HAL
open science

An ontology-based approach to manage conflicts in collaborative design

Moisés Lima Dutra

To cite this version:

Moisés Lima Dutra. An ontology-based approach to manage conflicts in collaborative design. Other [cs.OH]. Université Claude Bernard - Lyon I; Universidade nova de Lisboa, 2009. English. NNT : 2009LYO10241 . tel-00692473

HAL Id: tel-00692473

https://theses.hal.science/tel-00692473v1

Submitted on 30 Apr 2012

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

THESE

PRESENTEE DEVANT

L'UNIVERSITE CLAUDE BERNARD LYON 1

POUR L'OBTENTION DU DIPLOME DE

DOCTORAT EN INFORMATIQUE
(ARRETE DU 7 AOUT 2006)

PRESENTEE ET SOUTENUE PUBLIQUEMENT LE **27 NOVEMBRE 2009**
PAR

M. MOISÉS LIMA DUTRA

AN ONTOLOGY-BASED APPROACH TO MANAGE CONFLICTS IN COLLABORATIVE DESIGN

(Une approche basée sur les ontologies pour la gestion de conflits dans un environnement collaboratif)

DIRECTEURS DE THESE : PARISA GHODOUS (UNIVERSITE CLAUDE BERNARD LYON 1)
RICARDO GONÇALVES ("ORIENTADOR PELA FCT/UNL", UNIVERSITE NOUVELLE DE LISBONNE)

JURY :
ARIS OUKSEL (RAPPORTEUR, PROFESSEUR DES UNIVERSITES, UNIVERSITE D'ILLINOIS, USA)
YVON GARDAN (RAPPORTEUR, PROFESSEUR DES UNIVERSITES, UNIVERSITE DE REIMS)
SHAW FENG (EXAMINATEUR, CHERCHEUR, NATIONAL INSTITUTE OF STANDARDS, USA)
YAMINE AIT AMEUR (EXAMINATEUR, PROFESSEUR DES UNIVERSITES, UNIVERSITE DE POITIER)
PARISA GHODOUS (EXAMINATEUR, PROFESSEUR DES UNIVERSITES)
RICARDO GONÇALVES (EXAMINATEUR, CHERCHEUR)

AN ONTOLOGY-BASED APPROACH TO MANAGE CONFLICTS IN COLLABORATIVE DESIGN

MOISÉS LIMA DUTRA

DISSERTATION SUBMITTED TO THE
UNIVERSITÉ CLAUDE BERNARD LYON 1
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY IN COMPUTER SCIENCE

ADVISORS: PARISA GHODOUS (UNIVERSITY OF LYON 1)
RICARDO GONÇALVES ("ORIENTADOR PELA FCT/UNL", NEW UNIVERSITY OF LISBON)

COMMITTEE: ARIS OUKSEL (UNIVERSITY OF ILLINOIS, USA)
YVON GARDAN (UNIVERSITY OF REIMS, FRANCE)
SHAW FENG (NATIONAL INSTITUTE OF STANDARDS, USA)
YAMINE AIT AMEUR (UNIVERSITY OF POITIER, FRANCE)
PARISA GHODOUS
RICARDO GONÇALVES

To my Mother and Father,
Sister and Niece.

Acknowledgements

I would like to thank my advisors Parisa Ghodous and Ricardo Gonçalves, for their full support and encouragement during the development of this Ph.D. thesis. I am very fortunate to have had the opportunity to work under their supervision.

I also thank the committee members Aris Ouksel, Yvon Gardan, Shaw Feng, and Yamine Ait Ameur, for having evaluated this work.

I would like to thank Celson Lima as well, for the initial help and support during the begining of this work.

Then, I would like to thank my colleagues at the University of Lyon 1, Catarina Ferreira da Silva, Samer Ghafour, Patrick Hoffmann, Lionel Médini, Olivier Kuhn, Youssef Roummieh, Mahmoud Barhamgi, Nguyen Minh Tri, and Kamel Slimani. I also thank the Portuguese colleagues João Sarraipa, Carlos Agostinho, and Marco Delgado.

I thank my family, for everything that they have done for me in the past five years.

I wish to further thank Bill Siple, for the thesis review, and Alexandre Stauffer, for the previous reading of the thesis.

Finally, I would like to thank all the friends that I have made during my stay in France and Portugal, especially those from the French-Brazilian association BrasilParaTi, in Lyon.

Moisés Dutra

Résumé

De nos jours, les projets de conception complexes de produits exigent que les équipes de concepteurs se réunissent pour faciliter le partage de leurs compétences et expertises respectives afin de produire un ensemble de solutions de conception efficace. Dû au besoin croissant d'échanger les connaissances, les projets de conception modernes sont encore plus structurés pour travailler avec des équipes distribuées qui collaborent sur un réseau informatique pour accomplir une conception optimale de produit. Néanmoins, dans ce processus de conception collaborative, l'intégration d'équipes multidisciplinaires – qui implique l'échange et le partage des connaissances et compétences – génère fréquemment des situations conflictuelles. Les différents points de vue et perspectives des experts, les différentes façons de communiquer et collaborer au niveau de connaissances, rendent le processus difficilement maitrisable. Pour accomplir un scénario optimal, certains problèmes doivent d'abords être résolus comme la spécification et formalisation des besoins, l'intégration d'ontologies, la détection et la résolution des conflits.

Spécifier et formaliser les connaissances demandent un grand effort afin d'obtenir un modèle de représentation pour agréger plusieurs domaines différents des connaissances. Chaque expert pourrait s'exprimer afin que les autres comprennent leurs informations correctement. Il est donc nécessaire d'utiliser un modèle de représentation de données suffisamment clair et flexible pour accomplir cette tâche. Certains modèles actuels ne parviennent pas à fournir une solution efficace pour le partage des connaissances et pour la collaboration des projets de conception, car ces modèles n'intègrent pas les aspects géographiques, temporels, fonctionnels de la conception avec un modèle de représentation des connaissances flexible et générique.

Ce travail propose une architecture, pour la conception collaborative, qui ambitionne d'être synchrone, générique, orientée aux services, basée sur les agents, et basée sur les ontologies. Des modèles particuliers de représentation sont transformés en instances d'ontologie et sont fusionnés pour accomplir le model final de conception d'un produit. C'est une approche synchrone parce que le processus de fusion est entrepris en même temps que l'interaction entre concepteurs. C'est générique parce qu'elle permet aux utilisateurs de travailler avec deux approches pour l'intégration d'ontologies : celle qui utilise une ontologie générique et celle qui utilise un processus d'harmonisation. Notre proposition se concentre sur les conflits de la conception collaborative et fait usage de Web Ontology Language (OWL) et des Services Web, le premier comme langage pour représenter les connaissances et le dernier comme un support technologique pour la communication.

Mots-clés : Conception Collaborative, Ontologies, Services Web, Synchronicité, Généricité, Raisonnement et Harmonisation.

Abstract

Today's complex design projects require teams of designers to work collaboratively by sharing their respective expertise in order to produce effective design solutions. Due to the increasing need for exchanging knowledge, modern design projects are more structured to work with distributed virtual teams that collaborate over computer networks to achieve overall optimization in design. Nevertheless, in a collaborative design process, the integration of multidisciplinary virtual teams – involving exchange and sharing of knowledge and expertise – frequently and inevitably generates conflicting situations. Different experts' viewpoints and perspectives, in addition to several ways of communicating and collaborating at the knowledge level, make all this process very hard to manage. In order to achieve an optimal scenario, some problems must first be solved, such as requirement specification and formalization, ontology integration, and conflict detection and resolution.

Specifying and formalizing the knowledge demands a great effort towards obtaining representation patterns that aggregate several disjoint knowledge areas. Each expert should express himself so that the others can understand his information correctly. It is necessary, therefore, to use a flexible and sufficiently extensive data representation model to accomplish such a task. Some current models fall short of providing an effective solution to effective knowledge sharing and collaboration on design projects, because they fail to combine the geographical, temporal, and functional design aspects with a flexible and generic knowledge representation model.

This work proposes an information model-driven collaborative design architecture that supports synchronous, generic, service-oriented, agent-based, and ontology-based teamwork. Particular representation models are transformed into ontology instances and merged together in order to accomplish the final product design. It is a synchronous approach because the concurrent processes are undertaken at the same time that the interactions among designers take place. It is generic because it provides the users with two approaches for ontology integration: the use of a predefined generic ontology and the harmonization process. Our proposal focuses on collaborative design conflict resolution by using Web Ontology Language (OWL) and Web Services, the former as a tool for knowledge representation and the latter as a technological support for communication.

Keywords: Collaborative Design, Ontologies, Web Services, Synchronicity, Genericness, Reasoning, and Harmonization.

Contents

LIST OF FIGURES ..... **X**

LIST OF TABLES ..... **XII**

GLOSSARY ..... **XIII**

1 INTRODUCTION **1**

1.1 BACKGROUND AND MOTIVATION 2
1.2 GOALS 6
1.2.1 General Goals *6*
1.2.2 Specific Goals *7*
1.3 DISSERTATION STRUCTURE 7

2 COLLABORATIVE DESIGN **9**

2.1 WEB-BASED COLLABORATIVE DESIGN 12
2.2 AGENT-BASED COLLABORATIVE DESIGN 14
2.3 INTEGRATION OF WEB- AND AGENT-BASED APPROACHES FOR COLLABORATIVE DESIGN 17
2.4 OTHER APPROACHES FOR COLLABORATIVE DESIGN 18
2.4.1 Service-Oriented Architecture (SOA) *18*
2.4.2 Synchronicity *19*
2.4.3 Reuse and Optimization *20*
2.4.4 Sustainable Design *21*
2.5 SUMMARY 21

3 ONTOLOGY-BASED KNOWLEDGE REPRESENTATION **23**

3.1 ONTOLOGY TERMINOLOGY 25
3.2 SEMANTIC WEB 26
3.3 ONTOLOGIES AND INTEROPERABILITY 27
3.3.1 Formal Representation of Knowledge *28*
3.3.2 Standards for Interoperability *28*
3.3.3 Formalization of Standards *34*
3.3.4 Ontology Tools *35*
3.4 ONTOLOGY INTEGRATION METHODS 38
3.4.1 Mapping *38*
3.4.2 Merging and Aligning *39*
3.4.3 Harmonization *41*
3.4.4 Ontology Mismatches *43*
3.5 ONTOLOGY MANAGEMENT 45
3.5.1 Genericness *45*
3.5.2 Generic Ontology *46*
3.5.3 Natural Language *47*
3.5.4 Rules *48*
3.5.5 Other Approaches *48*
3.6 SUMMARY 52

4 CONFLICTS IN COLLABORATIVE DESIGN **54**

4.1 DEFINITIONS 54

4.2 TAXONOMY OF CONFLICTS 56
4.2.1 Technical Conflict Causes 57
4.2.2 Cognitive Conflict Causes 58
4.3 CONFLICT MITIGATION STRATEGIES 59
4.3.1 Terminological Resources: Ontologies and Thesauri 61
4.3.2 Prototyping 62
4.3.3 Design Rationale and Case-based Reasoning 62
4.3.4 Rule-based Reasoning 63
4.3.5 Negotiation 64
4.3.6 Supervision 65
4.3.7 Priority Management 66
4.4 CONSTRAINT MANAGEMENT 67
4.4.1 Definitions 67
4.4.2 Constraint Checking 68
4.4.3 Constraint Relaxation 69
4.5 SUMMARY 69

5 A PROPOSED FRAMEWORK FOR COLLABORATIVE DESIGN 71

5.1 GLOBAL ARCHITECTURE 72
5.1.1 Project 73
5.1.2 Agencies and Agents 74
5.1.3 Collaboration Levels 78
5.1.4 Publication 81
5.2 MAIN CHARACTERISTICS 82
5.2.1 Generic 84
5.2.2 Synchronous 85
5.2.3 Service Oriented 87
5.2.4 Agent Based 88
5.2.5 Ontology Based 88
5.3 SYSTEM MODULES 89
5.3.1 Management Module 89
5.3.2 Blackboard 91
5.3.3 Ontology Module 93
5.4 SYSTEM ROLES 94
5.4.1 System Administrator 95
5.4.2 Simple User 96
5.4.3 Project Manager 97
5.4.4 Agency Coordinator 98
5.4.5 Ontology Expert 99
5.5 COLLABORATIVE SERVICES 99
5.6 SUMMARY 102

6 AN ONTOLOGY-BASED METHODOLOGY TO DEAL WITH CONFLICTS . 104

6.1 FORMALIZATION PROCESS 104
6.1.1 A Proposed Mapping for EXPRESS and OWL 108
6.1.2 EXPRESS Schemas and Simple Types 108
6.1.3 Entity Types, Attributes, and Inheritance Relationships 110
6.1.4 Constructed Types 113
6.1.5 Aggregated Types 114
6.1.6 EXPRESS Rules 117

6.1.7 EXP2OWL Prototype ... *118*
6.2 ONTOLOGY-GENERIC METHOD ... 119
6.2.1 STEP-Based Product Model ... *120*
6.2.2 FBS Ontology ... *121*
6.2.3 Ontology Reasoning ... *122*
6.2.4 Constructing a Generic Ontology ... *123*
6.3 HARMONIZATION METHOD ... 125
6.4 APPLYING AN INFERENCE ENGINE ON A MECHANICAL PIECE MODEL ... 129
6.4.1 Description Logics-Based Reasoning ... *129*
6.4.2 Graph-based Representation of Concepts ... *130*
6.4.3 Defining Axioms ... *131*
6.4.4 Modeling a Small Gadget ... *132*
6.5 SUMMARY ... 136

7 THE CONFLICT RESOLUTION METHOD ... 138

7.1 AUTOMATIC SOLVING ... 140
7.1.1 Analogy-Based Resolution ... *141*
7.1.2 Deduction-Based Resolution ... *141*
7.2 INTERACTION AND NEGOTIATION ... 143
7.3 SUPERVISION ... 145
7.4 SUMMARY ... 146

8 PROTOTYPE ... 148

8.1 MODELING AN ELECTRICAL CONNECTOR ... 148
8.2 SCOOP – SYSTEM FOR COOPERATIVE WORK ... 152
8.2.1 Background Tools ... *157*
8.2.2 Conflicting Situation ... *158*
8.3 SUMMARY ... 161

9 CONCLUSIONS ... 162

9.1 PERSPECTIVES AND FUTURE RESEARCH ... 166
9.1.1 Improvement of Scoop ... *167*

REFERENCES ... 169

APPENDIX A – EXPRESS SCHEMAS ... 180

APPENDIX B – OWL CLASSES (AFTER TRANSLATION) ... 182

APPENDIX C – CONNECTOR 1 OWL REPRESENTATION ... 190

APPENDIX D – CONNECTOR 2 OWL REPRESENTATION ... 196

List of Figures

Figure 2.1 – The Single-model Approach (Ghodous, 2002) 11
Figure 2.2 - The Multi-model Approach (Ghodous, 2002) 11
Figure 2.3 – Engineering Portal (Huang et al., 2007) 13
Figure 2.4 – A Framework for Collaborative Design (Wang et al., 2009) 16
Figure 3.1 – An Example of Ontology 24
Figure 3.2 – Semantic Web Stack 32
Figure 3.3 – STEP-based Harmonization Famework 34
Figure 3.4 – Number of Concepts Used (Falconer et al., 2007) 37
Figure 3.5 – Ontology Language Usage (Falconer et al., 2007) 37
Figure 3.6 – Ontology Mapping Use Cases (Falconer et al., 2007) 38
Figure 3.7 – The Reference Ontology Building in MENTOR (Sarraipa et al., 2008) 42
Figure 3.8 – Conceptual Mismatches (Chungoora et al, 2008) 44
Figure 3.9 – Aggregation-Level Mismatches (Chungoora et al, 2008) 44
Figure 3.10 – Upper Ontologies (Pirró et al., 2009) 46
Figure 3.11 – Ontoling Architecture (Pazienza et al., 2005) 47
Figure 3.12 – A System Architecture for Ontology Management (Bloehdorn et al., 2009) 50
Figure 4.1 – A Typology of Conflicts (Matta et al., 1996) 57
Figure 4.2 – Decomposition of the Conflict Management Meta-process (Klein, 2000) 59
Figure 4.3 – A Conflict Taxonomy (Slimani, 2006) 66
Figure 4.4 – Constraint Management Module (Slimani, 2006) 69
Figure 5.1 – Global Architecture 72
Figure 5.2 – Problem Space 73
Figure 5.3 – Project Shared Workspace 74
Figure 5.4 – Multiagent Architecture 75
Figure 5.5 – Client Agency 76
Figure 5.6 – Design Agency 77
Figure 5.7 – Shared Workspaces 80
Figure 5.8 – Collaboration Levels 81
Figure 5.9 – Ontology Instance Publication 82
Figure 5.10 – The Sequential Design Approach 86
Figure 5.11 – The Synchronical Design Approach 87
Figure 5.12 – Management Module 89
Figure 5.13 – Blackboard Structure 91
Figure 5.14 – Ontology Reasoning 93
Figure 5.15 – Ontology Integration 94
Figure 5.16 – System Administrator Use Case Diagram 95
Figure 5.17 – Simple User Use Case Diagram 96
Figure 5.18 – Project Manager Use Case Diagram 97
Figure 5.19 – Agency Coordinator Use Case Diagram 98
Figure 5.20 – Ontology Expert Use Case Diagram 99
Figure 6.1– Formalization Process 106
Figure 6.2 – EXPRESS Statements Mappable to OWL 108
Figure 6.3 – Approach for Aggregated Types 114
Figure 6.4 – EXP2OWL Prototype 118
Figure 6.5 – Function getOWLEnumeration 119
Figure 6.6 – Function getOWL_Array 119

Figure 6.7 – STEP-based Product Ontology 121
Figure 6.8 – FBS Ontology 122
Figure 6.9 – Designer's "Pen" Product Instance 124
Figure 6.10 – Harmonization Process 125
Figure 6.11 – Manufacturer's "Pen" Product Instance 127
Figure 6.12 – Final "Pen" Product Instance 128
Figure 6.13 – Graphical Representation of Connector 1, Arcs Representing Compositional Properties 132
Figure 6.14 –Representation of the Connector 1 Classes in Protégé 133
Figure 6.15 – Properties Defined for Connector 1 133
Figure 6.16 – Graphical Representation of Connector 2, Arcs Representing Compositional Properties 134
Figure 6.17 – Representation of the Connector 2 Classes in Protégé 134
Figure 6.18 – Ontology Inconsistencies Detected with Protégé and Pellet 135
Figure 6.19 – Ontology Axioms that Caused the Inconsistency 136
Figure 7.1 – Conflicting Situation 139
Figure 7.2 – Blackboard's Conflict Area 140
Figure 7.3 – Conflict Notification Example 143
Figure 7.4 – Conflict Resolution Method Activity Diagram 146
Figure 8.1 – Electrical Connector 149
Figure 8.2 – Consumer's Requirements 150
Figure 8.3 – Manufacture's Requirements 150
Figure 8.4 – Connector's Function Model 151
Figure 8.5 – Scoop Login 152
Figure 8.6 – System Administrator's Workspace 153
Figure 8.7 – Project Manager's Workspace 153
Figure 8.8 –Attribution of Agencies 154
Figure 8.9 – Thermal Engineer's Private Workspace 154
Figure 8.10 – Client Agent's Specification Model 155
Figure 8.11 – Manufacturer Agent's Specification Model 155
Figure 8.12 – Mechanical Engineer's Proposed Models 156
Figure 8.13 – Thermal Engineer's Proposed Models 156
Figure 8.14 – Project P01's Shared Workspace 157
Figure 8.15 – Ontology Publication Sequence Diagram 159
Figure 8.16 – Thermal Engineer's Plug Model 159
Figure 8.17 – Mechanical Engineer's Plug Model 160
Figure 8.18 – Conflict Detected 160
Figure 8.19 – Former Version of Scoop 161

List of Tables

Table 2.1 – Summary of Collaborative System Features 22
Table 3.1 – Summary of Integration Methods 52
Table 5.1 – WSDL Specification File 102
Table 6.1 – Simple Types 109
Table 6.2 – Defined Data Type 109
Table 6.3 – Entities 110
Table 6.4 – Supertypes 111
Table 6.5 – Redeclared Attributes 112
Table 6.6 – Inverse Attributes 112
Table 6.7 – Enumeration 113
Table 6.8 – Select 114
Table 6.9 – Set 115
Table 6.10 – Array 115
Table 6.11 – List 116
Table 6.12 – Bag 117
Table 6.13 – “Product” Table 123
Table 6.14 – “Composition” Table 124
Table 6.15 – Correspondence Table for the “Pen” Design Project 128

Glossary

ANED – Argumentative Negotiation Framework for Engineering Design

AI – Artificial Intelligence

AIP – Agent Interaction Protocol

AP – Application Protocol

API – Application Programming Interface

CAD – Computer-aided Design

CAM – Computer-aided Manufacturing

CBR – Case-based Reasoning

CDFO – Common Design Feature Ontology

CE – Concurrent Engineering

CSCW – Computer Supported Cooperative Work

CSCWD – Computer Supported Cooperative Work in Design

CSP – Constraint Satisfaction Problems

CPD – Collaborative Product Development

DAML – DARPA Agent Markup Language

DARPA – Defense Advanced Research Projects Agency

DICE – Distributed and Integrated Collaborative Engineering Design

DL – Description Logics

EDI – Electronic Data Interchange

ERP – Enterprise Resource Planning

EXPRESS – Information modeling language (ISO 10303-11:1994)

FBS – Function-Behavior-Structure

FIPA – Foundation for Intelligent Physical Agents

FOL – First Order Logic

HTML – Hypertext Markup Language

HTTP – Hypertext Transfer Protocol

ICT – Information and Communication Technologies

IFC – Industry Foundation Classes

ISO – International Organization for Standardization

JADE – Java Agent Development Environment

JESS – Java Expert System Shell

JSP – JavaServer Pages

KB – Knowledge Base

KBR – Knowledge-Based Reasoning

KM – Knowledge Module

KQML – Knowledge Query and Manipulation Language

MAS – Multi-agent System

MENTOR – Methodology for Enterprise Reference Ontology Development

MV – Multi-Versioning

OWL – Web Ontology Language

PDM – Product Data Management

PHP – Hypertext Preprocessor

PLM – Product Lifecycle Management

RDF – Resource Description Framework

RDFS – RDF Schema

RBR – Rule-based Reasoning

RIF – Rule Interchange Format

RuleML – Rule Markup Language

SDK – Software Development Kit

SOA – Service-oriented Architecture

SOAP – Simple Object Access Protocol

SPARQL – Query Language for RDF

SQL – Structured Query Language

STEP – Standard for the Exchange of Product Model Data (ISO 10303)

SWOOP – Semantic Web Ontology Editor

SWRL – Semantic Web Rule Language

UI – User Interface

UML – Unified Modeling Language

UPON – Unified Process for Ontology Building

URI – Uniform Resource Identifier

W3C – World Wide Web Consortium

WordNet – Lexical Database for the English Language

WS – Web Services

WSDL – Web Services Description Language

WWW – World Wide Web

XMI – XML Model Interchange

XML – Extensible Markup Language

XSD – XML Schema

Chapter 1

1 Introduction

The design of a product involves, even more, different kinds of competencies and skills, joining efforts of several researchers and engineers, who are not necessarily located at the same place. We might ask, "What is the product that can, nowadays, be designed, developed, made and sold by a single person?" In fact, traditional ways of work and communication tools only permit a reduced level of interactivity. An interactive project means having engineers, researchers and all the involved people put together their own work and results – even if they are from different domains – in order to generate a common solution accepted by everybody. The technological progress in high speed networks along with new communication and interaction tools have enhanced the way that collaborative design is done.

Traditional product design systems use a sequential mode of design generation, which breaks a design task into a number of sub-tasks that can be sequentially executed in a predefined workflow. Such a sequential design mode has been found to be brittle and inflexible. It often requires numerous iterations, which make design expensive and time-consuming, and also limits the number of design alternatives that can be examined. On the other hand, sequential design is usually practiced with a downstream information flow. Information feedback from downstream operations (e.g., process planning and manufacturing at the shop floor) to the upstream design is usually performed by human interactions. It may also cause insufficient design evaluation/optimization and hence inefficient product development due to the absence of manufacturability checks at the design stage, based on available resources.

Mohammed et al. (2008) stress that traditional design processes account for 75% of the total production costs. Traditional design departmentalizes as the engineers develop the product design and any working and detail drawings associated within the conceptual design. The design is then passed to other departments within the organization (manufacturing for process and material selection, marketing, purchasing, etc.). The design may then be sent back to the design team for rework and revision. The traditional design process tends to be exceedingly tedious and resource inefficient. This tedious and inefficient nature can lead to increased lead time and loss of profit.

The impact of globalization on business has forced most industries to become more innovative and implement newer strategies to retain market leadership and growth with the desired profitability. The retention of corporate engineering knowledge and rules is also critical because of the aging workforce. Regarding this scenario, collaborative design plays a very important role in this situation, and this means that the companies have to utilize a variety of experts in collaboration. As Klein et al. (2003) put it very well, almost all complex

artifacts nowadays, including physical artifacts such as airplanes, as well as informational artifacts such as software, organizations, business processes, plans and schedules, are defined via the interaction of many, sometimes thousands of participants, working on different elements of the design.

With the advent of the Concurrent Engineering (CE), an emphasis is placed on the integration of product design with manufacturing and other disciplines. CE is a systematic approach to integrate the design of products with related manufacturing processes using software packages and computing techniques in a computer environment (Ghodous et al., 2002). Within CE, a designer can consider and evaluate the downstream manufacturing processes of the product life-cycle in the initial design phase.

As the product design process in a CE environment requires input from various disciplines, collaborative efforts become critical in product design. Collaborative product design is defined as groups, or communities, working together to optimize product design tasks, and all this process can be very complex. Complexity here arises from the need to synthesize different perspectives of a problem, to manage large amounts of information relevant to a product design task, and to understand the product design decisions in the long term. Manufacturing corporations have become more product-oriented, aiming at decreased lead times from design to manufacturing, minimal work-in-process, just-in-time flow of materials, and high efficiency and flexibility of manufacturing capacity utilization.

According to Alvares et al. (2008), the global competition and the fast changes in the consumers' demands have been causing significant changes in the production strategies of manufacturing companies. These companies have to work frequently with: (a) an increase in the competition, which is now global, and (b) the consumers' demands being modified continually. They think that these companies should be capable of foreseeing changes in the market, and to answer these changes satisfactorily. Traditional manufacturing systems, with a centralized structure, have difficulty satisfying such demands. According to them, in recent years, significant changes are being applied to the manufacturing strategy of those companies, particularly in those that work together seeking to stay competitive internationally in a volatile market.

With the trend of global competition and rapid advances of Internet technologies, collaborative design approaches are moving towards supporting distributed applications, in which geographically dispersed users, systems and resources can be integrated in an Internet/Intranet environment beyond the traditional boundaries of physical location and time zones. That context can be considered as Internet-based manufacturing, and its main characteristics are the use of distributed information, integration of processes, and remote manufacturing. The Internet has been providing a tremendous potential for the integration and remote cooperation in world-wide manufacturing applications, because it has become the world platform for sharing all types of information.

1.1 Background and Motivation

A collaborative design process may result in a very complex product – or system – such as a space shuttle or a car. The resultant artifacts may involve various techniques. Qin et al. (2006) say, for instance, that a rocket may have 40,000 electronic components and devices, and 4000 parts related to 150 pieces of equipment. Technically, designing a complex artifact requires

knowledge of various sciences, e.g., economics, sociology and management, and engineering technologies such as nuclear, control, mechanical and manufacturing engineering. Various tools and techniques need to be integrated in a collaborative design process. People working in different scientific fields may use the same technical terms but mean different things, or vice versa. The technical complexity also presents a great challenge for design management of a multidisciplinary team.

Nahm et al. (2006) say that early approaches to Computer-aided Design (CAD) systems have been criticized due to the absence of a coordinated and strategic approach to systems development, the prevalence of applications tailored for very specific purposes, the limited integration between the engineering processes, and the insufficient considerations on the multidisciplinary collaboration. We can say that due to the increasing complexity of design problems and the necessity of different expertise from a variety of engineering fields, it is difficult for a single designer to propose a design solution which satisfies all of the product life cycle functions.

A large number of software applications offer forums to facilitate exchanges among actors and help actors during their design activities, but they are not often well adapted to the actors' work. However, they do not formalize inter-skill relationships and do not guide the actors towards the setting up of a common knowledge project. They can only support simple, predicable processes, but not the dynamically changing and complex processes that are present in many organizations. They are too predictive. The CAD tools are principally focused on the generation of the objects and not on their functional and behavioral aspects. Moreover, in the case of collaborative work, multiple viewpoints (technological, human, expertise, etc.) are rarely supported (Robin et al., 2007).

A lot of collaborative projects use multi-agent systems as basis technology. Within the collaborative and distributed design process, large quantities of information and knowledge are widely distributed across multiple agents, which may be human, artificial or both. According to Movahed-Khah et al. (2009), human designer agents spend 80% of their time generating and searching for their data on the one hand, and 93% of their time evaluating the information to a no quantitative level of abstraction on the other. During the design process, agents are continually engaged in performing design tasks, requiring these large quantities of information and knowledge. Tasks require agents to interact. Interacting during the design process indicates that the agents may be affected by other agents in pursuing their goals and executing their tasks. Interactions take place by carrying out an action that modifies the state of the design problem.

These interactions are not only necessary, but they really form collaborative and distributed design process and they exert a great influence on the development of the final product. The interactions between different agents let us confront and integrate their various and domain dependent points of view. Furthermore, the complexity of the collaborative and distributed design process is defined by interaction with its forms: communication, cooperation and coordination. Agents communicate in order to better achieve theirs goals or those of the system where they exist. Communication enables agents to coordinate their actions and behavior. The purpose of coordination is to achieve or avoid states of affairs considered as desirable or undesirable by one or several agents. To coordinate their goals and tasks, agents have to explicitly take into consideration the dependencies among their activities. Cooperation is a form of interaction in which agents work together and draw on the broad collection of their knowledge and capabilities to achieve a common goal. Through these forms of

interaction, agents are able to select useful information, make sense of it and store useful selections for future use.

Product assembly also plays a critical role in getting products with high quality. As for many large and complex products, designers generally need to assemble and debug all physical parts of products beforehand in the factory, then disassemble all parts for transportation, and at last assemble the parts again in the application place, which results in a great waste of both manpower and material resources. That is why we need a collaborative assembly design system for long-distance guidance and discussion to reduce the time of assembly. Meanwhile it is also propitious to make different domain designers participate in the assembly process planning (Meng et al., 2006).

We can also say that complexity in collaborative product design arises from the need to synthesize different perspectives of a problem. Specifically, dependency identification of the product design process, as well as integration and sharing of computing applications among the design teams that are critical for efficiency of manufacturability. Web services have also been more and more used on collaborative product design through the Internet. However, Web services alone are passive whereas agents can provide alerts and updates when new information becomes available. As we show in our state-of-the-art survey, an approach that combines agent technology with Web services may be more suitable and stronger to face greater challenges.

Most collaborative design support systems are developed with the primary goal of achieving seamless information flow among designers and engineering systems (Jin et al., 2008). Database systems and various communication and workflow tools have been developed to support information sharing, design change propagation, and process management. Few systems help designers negotiate decisions for the benefit of the overall design, and little work has been done to quantitatively assess how negotiation protocols and strategies may influence collaborative behaviors and design results.

Whenever we talk about collaborative design, it is crucial to highlight a specific task: dealing with conflict. Klein (2000) says that all collaborative design processes are subject to the generic ‘design conflict’. According to Lu et al. (2000), the process of collaborative engineering design is relatively complex, and often results in various conflicts due to technical and social factors. They also say that to understand the relationships between design process and design conflict is critical to improve the collaborative design productivity.

When discussing strategies to mitigate the arising of conflicts in early-stage design, it is prudent as well not to forget the human factor in the development process. Klein et al. (2003) say that the collaborative design process is typically expensive and time-consuming because strong interdependencies between design decisions make it difficult to converge on a single design that satisfies these dependencies and is acceptable to all participants. Zheng el at. (2007) believe that designers and the project leader can track bugs and discuss changes in front of computers instead of face-to-face. To them, a collaborative CAD system can be used to interactively demonstrate its main features to customers, or to train or get feedback from customers, as well.

We can summarize this topic by saying that a major objective of collaborative design should be to address the insufficient or even absent manufacturability checks concurrently by detecting and considering conflicts and constraints at earlier design stages. A solution could

be to try managing all the resources involved, including people, organizations, software tools, and equipment. Relevant research issues are collaborative workflow, conflict management, cost and task management, activity scheduling, and computing resource management.

In recent years, most project teams from engineering departments around the world have started to increase and re-use the related product and intellectual capital, previously accumulated. The design process had to be rationalized to manage knowledge, skills, and technological patrimony. Besides, since the last decade, the information technology boom has allowed companies to manage loads of information efficiently owing to powerful search capacity i.e. catalogs and on-line information systems. However, engineers have at their disposal too much information and prefer to rely on their own experience rather than these information systems. At the same time, during the design process, engineers underestimate some factors such as ergonomics. Mahdjoub et al. (2009) says that, as a result, many products surrounding us have not been designed to respond to end-user expectations, including their need for usability. Usability represents the product's ease of use. Indeed, it corresponds to the ability of a technical device to be used easily by a consumer, in order to achieve the task for which that object was designed. Nevertheless, we come across products which remain complex and unsuitable although made for the general public. Workplaces, where a part of manufacturing systems lies, are also inadequate and cause serious security and health problems.

Our work is also concerned with software supporting collaborative design, such as PLM (Product Lifecycle Management) systems, which are needed to help design team members in their project tasks. PLM systems are supposed to support the design team members in carrying out collaborative activities in product-process-design projects. We can find in the literature some projects that combine PLM and agent technologies. This kind of approach usually looks for autonomy, reactivity, pro-activity and social ability in collaborative design.

Moreover, a great number of projects have tried to answer the problem of multiple points of view, but results were often limited and disappointing, and now many projects are developing new approaches. This becomes more evident when we look at the complex product industry. For instance, a number of projects have been undertaken recently within the aircraft industry, e.g. the work of Ölvander et al. (2009). As a consequence, we conclude that the existent operational applications which support the actors' needs during collaborative design activities cannot utterly fulfill all the needs that have emerged these latest years in collaborative design area.

The use of Ontologies to represent knowledge is an approach even more used in collaborative product development. Recent modeling languages – like OWL[1] and SWRL[2] –, and improving techniques that facilitate the task of matching, merging, aligning the ontologies, provide us with a very powerful way of exchanging and sharing expertise and data. The increment of rule-based systems and tools also provides us with a reasoning approach to extract information from ontologies. Moreover, the reasoning can be used to detect – and to resolve – collaborative conflicts at the early stages of design too.

However, representing heterogeneous knowledge may become a very hard task to deal with. Because, as Gero et al. (2009) said, designers are capable of seeing the world differently than

[1] http://www.w3.org/TR/owl-features/

[2] http://www.w3.org/Submission/SWRL/

the way it was initially intended. Also, that research evidence suggests that designers get inspiration and ideas from their designs that they did not imagine in advance.

1.2 Goals

1.2.1 General Goals

To cope with all this heterogeneity and complexity, we propose here an approach to manage collaborative design conflicts. This approach takes into account all the expertise involved in the collaborative project. Besides, we intend to provide the users with a computational environment where the design will be done locally but made available to the others on computer networks. Our architecture is structured in three levels of collaboration, the higher ones being the public spaces. Each design participant (called an *Agent*) develops his/her own design model and publishes it into the higher levels. In our approach, designers are gathered together in clusters of expertise and interests. We call each one of these clusters an *Agency*. The agents have their own private workspace. The agency workspace is the first public workspace. But the agencies also collaborate; consequently, they are all linked to the *Project* workspace, the higher level workspace in our architecture.

We use an ontology-based approach to represent knowledge in this environment. We consider that ontologies – especially the OWL-based ones – can be a very efficient way to detect collaborative design conflicts, as they provide us with precise computational formalization (e.g. OWL-DL). This formalization enables us to build an inference engine, which will reason about the product design information. Thus, when we talk about a designer's model, we are actually referring to an ontology instance.

Combining this inference engine with a generic environment where designers can work synchronously, sharing information and data, and interacting with one another since the very first phases of design, is our goal in this work. This interaction aims to mitigate and – whenever possible – to avoid the early appearance of conflicts.

This collaborative architecture also provides the designers with tools to detect potential problems and to help them solve conflicting situations, such as: the analysis of past cases, the application of rules, the communication workspaces, etc. Two different approaches are used to manage the ontologies after a formalization process is used to harmonize the representation formats:

1) Take a generic ontology previously created to be used in the current project. In this case, the users have to take this generic ontology as the basis for their private models;
2) Try to harmonize the users' expertise, letting them represent content in the way they are used to.

This work begins with the collaborative architecture that has been developed by our collaborative team at the University of Lyon 1 (Ghodous, 2002; Ferreira da Silva et al., 2004; Slimani, 2006; Dutra et al, 2006; Kuhn et al., 2008; Dutra et al., 2009). It is also part of a doctoral co-tutelle program between the University of Lyon 1 and the New University of

Lisbon, where part of this work has been developed during our stay in Portugal (from October 2006 to May 2007). The work done in Lisbon is within the scope of the European projects ATHENA[3], Interop[4], and InnovaFun[5].

1.2.2 Specific Goals

1) To undertake an extensive survey on the relevant literature on the state of the art for this work.
2) To evaluate the technologies, methodologies, and tools chosen to support this work.
3) To propose a collaborative architecture for collaborative design.
4) To propose an ontology-based methodology for conflict management.
5) To build a prototype to evaluate the proposal.

1.3 Dissertation Structure

This dissertation comprises 9 chapters. In this one, we have described the context that our work belongs to, including the main existing problems and the description of possible scenarios and goals to be achieved. The remainder of this work is organized as follows.

Chapter 2 presents an overview of collaborative design. We compare some projects and methodologies, focusing on the interaction between agent- and Web-based approaches for collaborative design.

In Chapter 3, we turn to ontology-based knowledge representation and the Semantic Web, discussing ontologies within the scope of interoperability, existing integration methods, modeling languages, and ontology tools. We also present a review of the existing work in this area that is suitable for comparing with our architecture.

In Chapter 4, we examine conflicts in collaborative design. We present a conflict taxonomy, mitigation strategies for conflict handling, and a constraint management approach to mitigate conflicts.

In Chapter 5, we present our proposed architecture for collaborative design – the overall approach, the main characteristics, the system modules, the system roles, and the collaborative services developed as the infrastructure for communication, stressing our specific contribution.

Chapter 6 presents our ontology-based methodology to deal with conflicts in collaborative design. This chapter briefly depicts some approaches used to represent ontologies and proposes a formalization process of standards, which includes a proposed mapping and a

[3] http://www.athena-ip.org/

[4] http://www.interop-vlab.eu/

[5] http://www.funstep.org/

translator tool EXPRESS to OWL. It also depicts the generic ontology and the harmonization methods for ontology integration.

In Chapter 7, we talk about our proposed conflict resolution method, which comprises two sub-approaches: automatic solving and negotiation.

Chapter 8 presents our prototype and its internal organization, along with a small industrial scenario, in which we have tested it.

We conclude this dissertation in Chapter 9, with a summary of the work and a brief discussion of issues for future research.

Chapter 2

2 Collaborative Design

Collaborative Design is the process of designing a product through collaboration among multidisciplinary product developers associated with the entire product lifecycle. This includes those functions such as preliminary design, detailed design, manufacturing, assembly, testing, quality control, and product service as well as those from suppliers and customers (Sprow, 1992). Collaborative design refers to the coming together of diverse interests and people to achieve a common purpose of developing a product via interactions, information and knowledge sharing, with a certain level of coordination of the various implemented activities (Gzara Yesilbas et al., 2006).

Collaborative design is the result of researchers' work on design methodologies. It has become possible because of the evolution of computational technologies. To Klein (2000), collaborative design generally experiences fewer delays and other costs from design conflicts than does serial design. To some extent, collaborative design is a response to the growing need of data exchange. Another important objective of collaborative design is to address the insufficient or even absent manufacturability checks concurrently by detecting and considering conflicts and constraints at earlier design stages. To support collaborative design, information and communication technologies are used to augment the capabilities of the individual specialists, and enhance the ability of collaborators to interact with each other and with computational resources.

Participants in collaborative design projects may include artists, industrial designers, engineering designers, manufacturing engineers, business managers, suppliers, customers, vendors, etc. For example, artists can work at the conceptual design stage to perform aesthetical design of product. Industrial designers can produce product form design, ergonomics evaluation, human factors, and address environmental issues and branding strategies. Engineering designers can detail the product and make it realizable. Manufacturing engineers perform its manufacture and so on. All those participants may have differences in educational and cultural background, professional knowledge and skills, personalities, and etc. The large number of participants and their differences above raise the complexity in interaction and interfacing.

Collaborative design has some particular characteristics (e.g., diverse and complex forms of information, interdisciplinary collaboration, and heterogeneous software tools), which make interactions difficult to support. Traditional approaches to sharing design information among collaborators and their tools include the development of integrated tools and the establishment of common data standards. These approaches are not good at supporting effective

collaborative design because of the highly distributed nature of the design teams and engineering tools as well as the complexity and dynamics of design environments. Members on a collaborative team often work in parallel and independently using different engineering tools distributed at separate locations, even across enterprise boundaries and across various time zones around the world (Shen et al., 2003; Li et al., 2007). Moreover, collaborative design also involves various stakeholders with different intentions, backgrounds, and circumstances. The group design activities are influenced not only by technical decisions, but also by social interactions.

Thus, a successful implementation of collaborative design needs: a series of new strategies, including an efficient communication strategy for a multidisciplinary group of people from the design and manufacturing departments to share and exchange ideas and comments; an integration strategy to link heterogeneous software tools in product design, analysis, simulation and manufacturing optimization to realize obstacle-free engineering information exchange and sharing; and, an interoperability strategy to manipulate downstream manufacturing applications as services to enable designers to evaluate manufacturability or assemblability as early as possible (Li et al., 2007). On the other hand, the objective of a design team has multiple facets, for example, optimizing the mechanical function of the product, minimizing the production or assembly costs, or ensuring that the product can be easily and economically serviced and maintained. Achieving global satisfaction, cooperative strategy, such as negotiation, optimization and trade-off, is also an important research issue in collaborative design (Shen et al., 2008).

The study of design models points out the fact that, according to the design type, the design objects are different (Rose et al., 2007). When the resolution steps are known, the project is structured according to various activities that transform the product knowledge. In other cases, design could be considered as a creative or innovative design process, and activities do not structure the project. Design must be identified as a process that supports the emergence of solutions. In this case, the project is organized to favor the collaboration between the actors of the process and the project manager strives to create design situations facilitating the emergence of solutions. He/she decides on the adapted organization favoring collaborative work and supporting the sharing of information and knowledge (Rose et al., 2007).

Actors of a workgroup should agree upon gaps implicitly left by management and upon procedures to be set up in order to fill these gaps. They are supposed to gather their skills and finally establish a common understanding, in a shared context. These operations constitute a common repository essentially based on verbal and deliberate communication. Although in most design processes, coordination entails clear communication between designers, the real reason for this coordination is not for communication but for resolving dependencies between product specifications (Ouertani et al., 2007). Design is constraint oriented, and comprises many interdependent parts. A change in one part may have consequences for another part, and designers cannot always oversee these interdependencies and consequences.

To Ghodous (2002), collaboration between experts coming from different areas implies that the system which allows several people to work at the same time on the same product model, should be able to manage the eventual conflicting situations arisen in this collaboration. This system should avoid the loss and redundancy of information, as well as the coherence of the handled information.

Ghodous (2002) thinks that to solve this problem, it is necessary to represent in an adequate way the information from each activity involved in the collaborative process, including the viewpoints associated to them. This work evaluated several collaborative projects in the mechanical and assembly domain. She has detected and classified two different approaches for collaborative development:

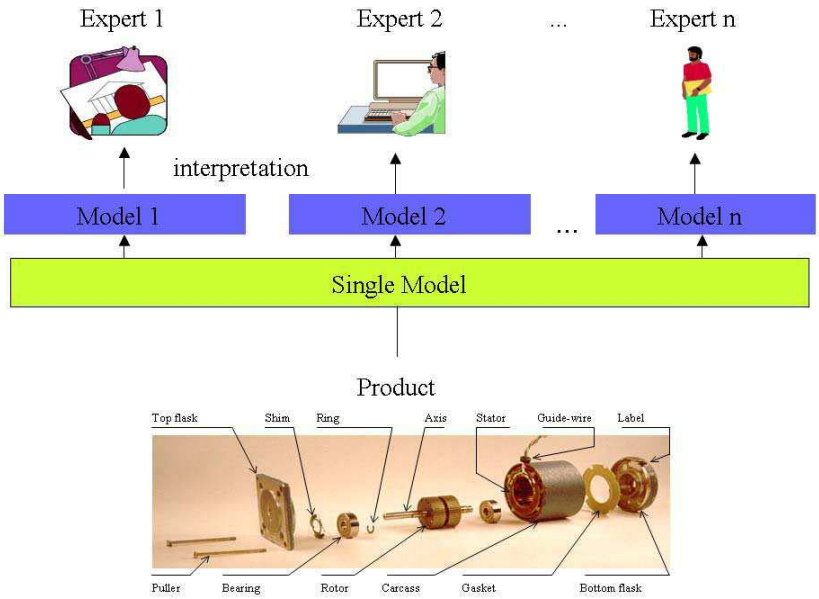

Figure 2.1 – The Single-model Approach (Ghodous, 2002)

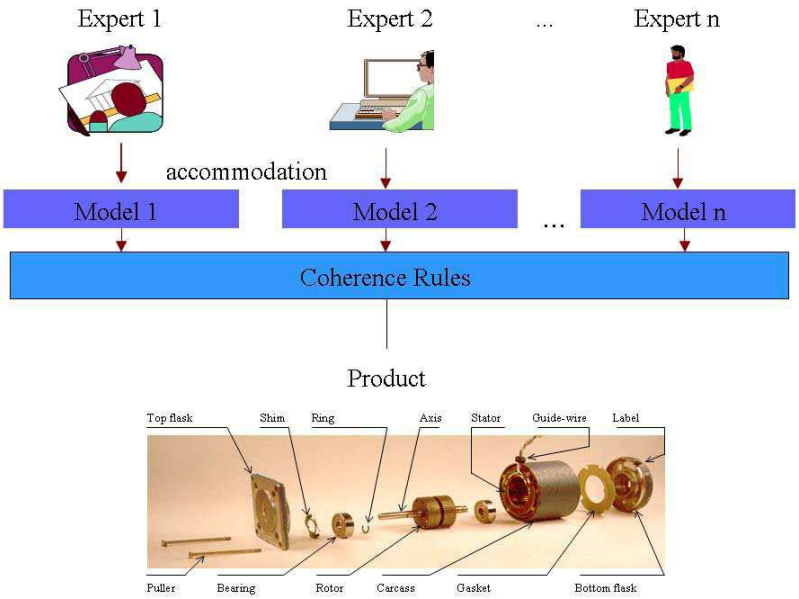

Figure 2.2 - The Multi-model Approach (Ghodous, 2002)

- The single-model approach. The objective here is to create a single and consensual model for the different viewpoints (Figure 2.1). This model should be able to represent all the information related to each viewpoint.
- The multi-model approach. This approach consists of permitting each expert to create his/her model from his/her own viewpoint, and then to integrate these different models, establishing relationships between them (Figure 2.2). In this kind of approach, several areas of expertise are involved, each one concerned with a specific aspect of the product's development. The representation of functional properties is, according to Ghodous (2002), the essential aspect of the multi-model approach. The notion of dynamics is closely related to the multi-model approach for collaborative design.

To Ölvander et al. (2009), collaborative design is a special form of problem solving where a set of frequently unclear objectives must be balanced without violating a set of constraints. By employing modern modeling, simulation, and optimization techniques, it is possible to achieve vast improvements, even at the conceptual stage of the design process. Ölvander et al. (2009) present a formal mathematical framework for the use of the morphological matrix in a computerized conceptual design framework. The authors consider that most designs are actually modifications of already existing products. Thus, this morphological matrix exploits this and encourages the designer to identify novel combinations of existing solution principles.

Klein et al. (2003) think that the key challenge raised by the collaborative design of complex artifacts is that the design spaces are typically huge, and concurrent search by the many participants through the different design subspaces can be expensive and time-consuming because design issue interdependencies lead to conflicts (cf. Chapter 4). We can say, hence, that in collaborative design environments, good communication and coordination among team members are the best ways to avoid conflicts.

In the following sections, we are going to depict some specific approaches for collaborative design. Section 2.1 presents Web-based collaborative design, and in Section 2.2, Agent-based collaborative design. In Section 2.3, we present some projects that use both approaches (Agent- combined with Web-based approach). In Section 2.4, some projects with other particular characteristics are presented, namely service-oriented architecture, synchronicity, reuse and optimization, and sustainable design. Section 2.5 contains the chapter's summary.

2.1 Web-based Collaborative Design

The Web was originally developed for information sharing within internationally dispersed teams and the dissemination of information by support groups. Proposed and developed early in the 1990's, the Web has quickly become a convenient medium for publishing and sharing information relevant to the design process, from concept generation and prototyping to virtual manufacturing and product realization. It has been adopted as the most popular implementation architecture of a collaborative product development (including design and manufacturing) tool. Despite the original role of the Internet and the Web as one of exclusively textual data exchange, a whole new paradigm has arisen from them, following the metaphor "work anywhere, at anytime."

Web-based collaborative design systems are developed by mainly adopting the WWW as a collaboration platform with the Web server working as a repository of design information that can be accessed through a Web browser over a network such as the Internet or an intranet. This type of system is very popular due to the ubiquity of the Web browser as a client and they are convenient for designers to take on activities including product design share and review, design discussion, customer surveys, etc. The ability to update and maintain Web-based systems without distributing and installing software on potentially thousands of client computers is a key reason for their popularity. However, the life-cycle of each collaborative design interaction between a client and a server is only a unidirectional information flow process. This characteristic hinders the efficient and effective broadcast of design changes made by a designer working with a client to designers working with other clients (Wang et al., 2009).

Web-based collaborative design systems use the client-server architecture. In order to support collaboration, Web-based design servers need to communicate the structure of the design representation so that users can pose queries about formal design concepts such as rationale and purpose, or the causality between physical and functional elements (Wang et al., 2002). To facilitate a viable design environment, Web servers must also engage users in a dialog-like interaction that encompasses a range of activities, such as geometric and semantic product modeling, design representation, user-interaction and design browsing and retrieval. However, the Web technology itself cannot satisfy these requirements. In other words, information access is not the only outstanding problem. In order to collaborate on a distributed project, remote engineers and designers need active help to coordinate their efforts. This coordination involves translation of terminology among disciplines, locating/providing generic analysis services, prototyping services, and product management (Wang et al., 2002).

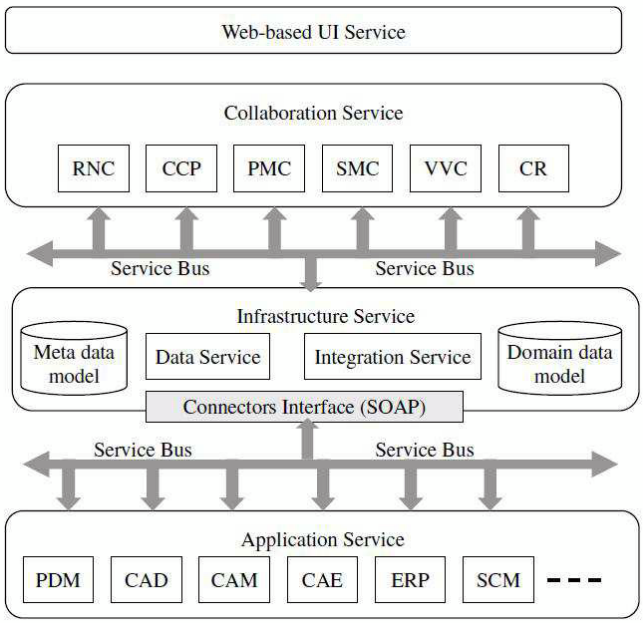

Figure 2.3 – Engineering Portal (Huang et al., 2007)

Huang et al. (2007) propose a workflow and Web-based engineering portal for collaborative product development (CPD). The workflow is used to model and execute the CPD processes, and the Web service is the implementing technology of the portal. They argue that the combination of workflow and Web service can fit the gap between business and information

system and achieve on-demand business. This portal (Figure 2.3) serves as a central point of access to a common CPD process model and allows collaborating partners to share product information stored in various application systems transparently via the Web.

They say that the use of modern technologies like Web services and workflow enables not only information collaboration from the IT perspective, but also process collaboration from the business perspective. Their architecture is structured in four layers (Huang et al., 2007):

- Application service layer: provides functionalities to access proprietary information sources of the enterprises involved in a collaboration, like PDM systems, ERP systems, etc.
- Infrastructure service layer: provides the basic data service and integration service for CPD. All data flows within the portal are supported by the infrastructure. The data service manages distributed data sources and offers a transparent view of the data for the portal services.
- The collaboration service layer provides the specific collaboration services. Five basic collaboration components are provided by the portal in the form of service, namely request and navigation, collaboration partner, process management, specification modeler, visualization and validation, and cultural repository component.
- User interface (UI) service layer: Service-specific UI service and generic UI service is provided for browsing available information resources or navigating through networked data structures. The UI services interact with the other services via HTTP or SOAP.

2.2 Agent-based Collaborative Design

Web technology alone is not the only solution to support collaborative design systems, although it makes communication physically viable through a common network. In order to collaborate on a distributed design project, remote engineers and designers need active supports to coordinate their efforts. This coordination involves the translation of terminology among disciplines, locating and providing engineering analysis services, virtual prototyping services, and project management (Liu et al., 2007; Mervyn et al., 2007). Web servers should not only be a repository of information but also provide intelligent services to help users to solve design problems. Such servers may be called software agents (Shen et al. 2008).

Multiagent systems (MAS) are a rapidly developing area of research that emerged from distributed artificial intelligence (Ferber, 1999). In agent-based collaborative design systems, software agents are mostly used for supporting cooperation among designers, enhancing interoperability between traditional computational tools, or allowing better simulations – particularly distributed simulations. From a historical point of view, the agent-based technology has been used to develop collaborative design systems even before the Web-based technology (López-Ortega et al., 2008). It can also be said that collaboration is facilitated by adding intelligence to agents.

A software agent is to be considered as an autonomous problem solver that is capable of proactively interacting and communicating with other agents on a high semantic, ontology-based level using agent interaction protocols (AIP). An AIP determines the typical pattern of

message exchange for agent communication combined with a common message content understanding based on a domain specific ontology. Accordingly, a multiagent system (MAS) is understood as a loosely-coupled network of several, autonomous and interacting problem solvers (software agents) that work together to solve some problems being above their individual capabilities. The sum of several interacting agents' capabilities in a MAS, therefore, is likely to exceed the sum of its individual parts (Ferber, 1999).

To Wang et al. (2009), agent-based collaborative design systems attempt to take full advantage of the emerging agent technologies and they are concerned with how a group of software agents can cooperate with each other or human designers to collectively manipulate design information and knowledge and solve design problems. The agent-based approach has received great attention in the past years as a promising alternative to the traditional client-server paradigm. The reasons often given for adopting an agent-based approach to develop distributed software systems are linked to their being proactive object systems and to the simplification of the system architecture. When used appropriately, applications of agent technology can lead to the desired modularity allowing flexible simulation and to better response and improved software reusability. Furthermore, they allow handling ill-structured or rapidly changing situations in a more economical way.

DICE (Distributed and Integrated Collaborative Engineering Design) (Sriram, 2002) is a relevant project in the collaborative design domain. It is a multi-agent and blackboard -based architecture. Its agents are called Knowledge Modules (KMs). These agents can call knowledge modules, encapsulate specialist systems for specific tasks or, still, encapsulate CAD tools, databases, etc. They can also represent human experts. Another DICE component is its blackboard, along with its control mechanism that executes two tasks: i) Evaluation and propagation of actions executed by KMs; and ii) Management of the negotiation process in case of a conflicting situation.

Within the more recent work of our team at the University of Lyon 1, we can cite Slimani (2006), who proposed a system for knowledge exchanging and sharing in collaborative design. This approach used a multi-agent system based on shared workspaces – through blackboards – and a multi-layer architecture to exchange data and information. In this architecture, a specific module to manage collaborative conflicts was built. However, this conflict module relies strongly on a constraint-based approach also proposed by Slimani (2006). In Chapter 4, we are going to discuss further about this subject. The architecture of Slimani is among those taken as a basis to our work. We have enhanced Slimani's approach through the introduction of Web services and ontology-based representation.

There are also some approaches to tackle with collaborative conflict. Jiao et al. (2006) say that a MAS environment may consist of a geographically distributed network supporting a large number of autonomous, heterogeneous agents. Obviously, to establish an effective conflict management mechanism for such a system is an extremely ambitious challenge. Jiao et al. (2006) believe that an integrated conflict management mechanism should include conflict avoidance, conflict detection, and conflict resolution. According to Barber et al. (2001), MAS implement distributed problem-solving, which provides many advantages including fast parallel computing, flexibility through partitioned expertise, narrow-bandwidth high-level communication, and increased fault tolerance. Many systems existing in the real world are naturally distributed due to their spatial, functional, or temporal difference; therefore, MAS can easily fit into such distributed environments. Through coordination, agents within such a system can work together to ensure coherent action. Conflict resolution is essential for

coordinated agent behavior (Barber et al., 2001). Coordination is the process by which agents reason about and manage the interdependencies among their behaviors and try to ensure that all members of the system act consistently.

Alvares et al. (2008) use multi-agent system with the internal organization as being of the blackboard type, and the architecture itself as being of the federated type using a facilitator to provide cooperative work. This approach intends to be cooperative and distributed, so that it can be used to solve many problems associated with CAD/CAM integration in a context of a community of agents.

Bilek et al. (2006) proposed a three-tier-based model for collaborative structural design. The three levels correspond to the complex coactions between human actors, software agents, and design specific resources. They implemented a hybrid, partial replicated data management strategy meaning that every designer is capable of designing structural elements in his private workspace. Supervisory control and synchronization of all partial product models, however, is carried out by the product model agent. Consequently, they had to implement the following services:

- Transaction service: Allows for creating, removing and modifying product model entities.
- Request service: Facilitates the transmission of requested product model entities.
- Synchronization service: Synchronizes the personal co-operation agents that are participating in a common product model construction process.
- Notification service: Notifies other registered agents about product model changes.

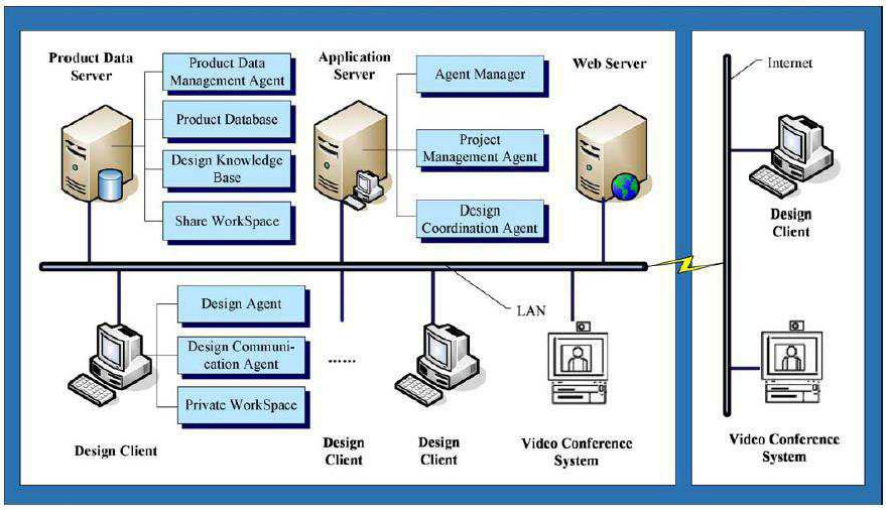

Figure 2.4 – A Framework for Collaborative Design (Wang et al., 2009)

The work of Wang et al. (2009) presents a framework for collaborative design (Figure 2.4). This framework adopts an agent-based approach and relocates designers, managers, systems, and the supporting agents in a unified knowledge representation scheme for product design. In order to model the constantly evolving design process and the rationales resulting from design collaboration, a collaborative product data model and a constraint-based collaborative design

process model were proposed to facilitate the management and coordination of the collaborative design process as well as design knowledge management.

Wang et al. (2009) describe the development of an agent-based environment capable of assisting designers in the management and coordination of the collaborative design process via the cooperation of a network of agents. The proposed system integrates AI based agents with traditional solid modeling systems such as AutoDesk Inventor to provide agents with 3D modeling. An example of designing a set of dining table and chairs is used to demonstrate how various agents can communicate with each other to provide support to the overall progress of the design project, by resolving potential conflicts which arise from the design collaboration.

2.3 Integration of Web- and Agent-based Approaches for Collaborative Design

Web services are increasingly being adopted as a ubiquitous means to access remote applications. They are self-contained, self-describing, modular applications that can be published, located, and invoked across the Internet. By means of Web services, computers and devices are connected with each other through the Internet to exchange data and combine data in new ways. In effect, agent-based technologies provide the mechanism for components to seek work, enter into cooperative agreements, and furthermore address the requirements of a dynamic, heterogeneous environment. Therefore, agent-based Web services can not only provide feasible system interoperation, but also increase the efficiency of distributed collaboration. Moreover, since agents communicate dynamically via message exchanges that conform to specify protocols/patterns, agent-based conversations are naturally compatible with Web services interactions (Huang et al., 2004).

Both the Web and agent technologies are very useful in implementing collaborative design systems. Shen et al. (2008) say that the attractiveness of the Web for propagating information makes it appropriate to integrate with agents for accessing and manipulating information automatically. The challenge is to build a Web-based environment that enables and supports seamless interactions between human designers, software agents, and Web servers using the available emerging technologies (Wang et al., 2003). A Web-based collaborative design system usually uses a client/server architecture, in which the interaction between components is predefined. This kind of approach is insufficient to support dynamic collaborative design environments, where tasks usually involve complex and non-deterministic interactions, producing results that might be ambiguous and incomplete.

Although agent technology has been recognized as a promising approach for developing collaborative design systems, they have so far been implemented in various prototype and industrial applications that are actually not very “intelligent”. In this view, agent applications in the Web-based collaborative design field are still facing many challenging questions. WebBlow (Wang et al., 2003) is an interesting attempt on the integration of the Web and software agents in implementing a multidisciplinary design optimization environment. Before the emergence of Web services, the concept of an active Web server was proposed to integrate the Web and agent technologies. Since the active Web servers have very similar features of Web services, it is natural for the ensuing work to implement collaborative design systems using Web services. During the past five years, more and more collaborative design

systems have been developed using Web services as well as Semantic Web and Grid Computing techniques. It is now easy to find hundreds of publications on this topic, e.g., there were more than 30 related papers presented at Computer Supported Cooperative Work in Design (CSCWD) 2006 and more than 40 at CSCWD 2007 (Shen et al., 2008).

Mahesh et al. (2007) propose a Web-based framework for distributed and collaborative manufacturing of engineering parts that use MAS. Different domains in manufacturing are represented as functional modules. All manufacturing agents communicate over the Internet via a bundle of Knowledge Query and Manipulation Language (KQML) messages to transfer data and information among each other. KQML is a common communication protocol used for negotiation and interaction between agents.

Huang et al. (2004) propose an agent-based Web services architecture for supporting collaborative design. This approach includes three steps:

i. Enhance the Web services with agent-based functionalities;
ii. Identify the dependency and explore the relationship among the sub-processes so as to assist the scheduling of the sub-processes of the collaborative product development; and
iii. Create a simple agent communication language, ASC (agent and services communication), to integrate the communication between Web services and agents.

2.4 Other Approaches for Collaborative Design

In the following sections, we have classified some works according to criteria that we believe are important to be implemented into our architectutre, namely service-oriented architecture, synchronicity, reuse and optimization, and sustainable design.

2.4.1 Service-Oriented Architecture (SOA)

The World Wide Web Consortium (W3C)[6] refers to the Service-Oriented Architecture (SOA) as a set of components which can be invoked, and whose interface descriptions can be published and discovered. According to Microsoft[7], the goal for SOA is a world-wide mesh of collaborating services that are published and available for invocation on a service bus (AlMellor et al., 2004). A SOA is a component model that inter-relates the different functional units of an application – the services – through well-defined interfaces and contracts between these services. These interfaces are defined in a neutral manner that should be independent of the hardware platform, the operating system and the programming language in which services are implemented. This allows services, built on a variety of such systems, to interact with each other in a uniform and universal manner.

Placing emphasis on interoperability, SOA combines the capacity to invoke remote objects and functions, i.e., the services, with standardized mechanisms for dynamic and universal

[6] http://www.w3.org/

[7] http://www.microsoft.com

service discovery and execution (Delgado et al., 2006). It offers flexible mechanisms that allow different technologies to be dynamically integrated, independent of the system's platform in use. This architecture promotes reusability, and it has reduced the time to make available and get access to new system's functionalities, allowing enterprises to dynamically publish, discover and aggregate a range of Web services through the Internet. According to Delgado et al. (2006), SOA encourages enterprises to be focused on their business and services, not constrained by the specificities of the applications and platforms. This is an essential requirement for organizations to achieve information technology independence, business flexibility, agile partnership, and to be seamlessly integrated in dynamic collaborative working environments and in digital ecosystems.

According to Pahl et al. (2007), the SOA approach creates some specific requirements for supporting software engineering methods. Firstly, services are used 'as is', with providers and consumers often coming from different organizations. They say this requires detailed descriptions of functional and non-functional aspects in a mutually agreed format. Secondly, service descriptions need to be based on rich modeling languages, in order to support their automated discovery and composition. Thirdly, reuse is a central aim of service technology that needs to be addressed on the level of individual services as well as service compositions Pahl et al. (2007). To Brambilla et al. (2007), one of the promises of SOA is the ease of integration of different, loosely coupled services across the Internet and Intranets. Still, the discovery and integration of a new service in an existing infrastructure is not automatic and requires a lot of human effort.

2.4.2 Synchronicity

Many collaborative CAD systems employ generic application-sharing technology that allows conventional single-user CAD systems to be transparently shared by multiple users for real-time collaborative design work. In addition, there are also collaborative systems that are built from scratch with advanced collaboration functionalities integrated (Zheng et al., 2008). However, they argue that due to the special characteristics of the CAD systems, these systems suffer from several significant drawbacks such as not allowing concurrent work and poor local response.

At the early stages of the design process, collaborative designers focus on accumulating various design ideas, especially creative ones. Under these circumstances, they expect the collaborative design system to obtain design ideas of individual designers, analyze them and find out the best ideas. At the later stages of a design process, however, a detailed design plan has already been worked out and design tasks have been well divided and assigned to collaborative designers. As a result, potential conflicts between designers during a collaborative process would be rare. A collaborative design system should help designers to complete their tasks efficiently and effectively by providing them with appropriate design functionalities and, if possible, synchronously. This approach can be very useful when dealing with conflicts.

Conflicts remain as one of the biggest challenges when the stakeholders work together in a real-time environment. To ensure data consistency, most existing systems adopt a pessimistic approach (e.g., locking and turn taking) to prevent conflicts from happening. However, the side effect of such an approach is that the design efficiency is greatly reduced and concurrent design can hardly be achieved. To attenuate this problem, Zheng et al. (2008) believe that

designers should be allowed to concurrently perform their own work while the system adopts non-pessimistic approaches to resolve conflicts.

Chen et al. (2005) have developed a synchronized collaborative design platform based on heterogeneous CAD systems to exchange procedural CAD models between heterogeneous CAD systems. According to them, the real-time exchange of a single operation was extended to the exchange of the complete procedural CAD model between the systems. Web services technology was adopted to encapsulate the procedural CAD model exchange functions, which are then released on the Internet as a standard interface and can be used by remote developers in their applications. A Web service for exchange of procedural CAD models between SolidWorks and Autodesk Mechanical Desktop was also realized.

We can also find real-time approaches for collaborative design, such as the one proposed by Meng et al. (2006), a synchronous collaborative model to satisfy real-time requirements (especially those concerning 3D streaming transmission). They focus on the collaboration aspect and on the assembly aspect as well, as they have decided that the representation of models should not only care about the network speed, but also should be convenient for assembly manipulations. A model tree structure is used in favor of assembly manipulation and the system only transmits commands and parameters changing the disassembly scene to the other clients, and then the clients parse and execute those orders by themselves.

2.4.3 Reuse and Optimization

As knowledge becomes explicit – and thereby available for further processing –, tools to retrieve and reason about models, simulations and their results allow us to exploit information acquired through previous processes to guide the execution of the current development process (Mühlenfeld et al., 2008). In particular multidisciplinary scenarios, where several teams work concurrently to achieve a suitable trade-off between different requirements, it is important to consolidate information managed in different teams and heterogeneous information systems to provide a consistent picture of the current state of the design processes, artifacts and models.

Mühlenfeld et al. (2008) have investigated how ontological engineering can improve existing design optimization scenarios. They used a method of modeling the design process via task and artifact ontologies to capture knowledge required for subsequent design optimization. According to them, by exploiting captured knowledge, it becomes possible to analyze and compare past modeling efforts with the current analysis scenario to identify potentially redundant simulations and to query and reuse information obtained from previous simulations. This approach uses the ISO 10303[8] standard and extends existing representations with additional ontologies to provide abstract views of concrete development artifacts. Explicit formalizations of essential analysis-specific properties of a design alternative provide the means for isolating and browsing similar compatible design optimization sub-processes. The resulting framework enables reuse of previous results in design optimization, hence speeding up the optimization processes by avoiding unnecessary time-consuming numerical simulations.

[8] ISO 10303-11:1994: Part 11: The EXPRESS language reference manual.

2.4.4 Sustainable Design

A research topic that has gained a lot of attention recently is Sustainable Design (Lilley, 2009). Sustainable design takes into account environmental, economic and social impacts that occur throughout the product lifecycle (Bhamra et al., 2007). Sakao et al. (2009) also talk about Eco-Design, an "environmentally conscious design."

These interrelated domains are often referred to as the three pillars or triple bottom line of sustainability. Whereas economic and environmental concerns are generally well defined and understood, the social sphere of sustainable design is less so and as such warrants further explanation. In its broadest terms it can encompass personal responsibility, quality of life, health, well-being and happiness, democratic participation and cooperative behavior (Lilley, 2009).

Lilley (2009) says still that designers shape the development of products and services which directly impact upon society and the environment. The application of sustainable design strategies can greatly reduce lifecycle impacts. Impacts which occur during use, however, are often determined by consumer behavior. Influencing user behavior can be challenging. In spite of over a decade of campaigns exhorting consumers to behave differently and greater product efficiency, consumers are slow to adopt more sustainable behaviors and behavioral changes made are often short-lived. Designers are in a position to reduce use impacts by purposefully shaping behavior towards that which is more sustainable.

2.5 Summary

Product design today requires new interaction forms between the various stakeholders involved in this specific process. Nevertheless, the sharing of expertise, knowledge and know-how jointly linked with the development of communication means generating a great number of pieces of information and knowledge.

To Klein et al. (2003), the collaborative design process is challenging because strong interdependencies between design decisions make it difficult to converge on a single design that satisfies these dependencies and is acceptable to all participants. Current collaborative design approaches are as a result typically characterized by heavy reliance on expensive and time-consuming processes, poor incorporation of some important design concerns – typically later life-cycle issues such as environmental impact –, as well as reduced creativity due to the tendency to incrementally modify known successful designs rather than explore radically different and potentially superior ones.

Li et al. (2005) think that it is important to establish a vertically seamless linkage between the upstream design and the downstream manufacturing processes through the creation of intelligent strategies for effective information interchange, and the horizontally interpersonal linkage of group work in the upstream design phases.

We can say that the integral system can support interrelated activities and share domain knowledge between designers and systems to improve design quality and efficiency. Modules for the hierarchical collaboration should be wrapped as services for remote invoking. Within the integral system, scheduling and coordination is becoming crucial and challenging, and

distributed intelligent algorithms and technologies such as agent-based systems, combined with Web-based approaches, can be used to enhance the integrated system.

FEATURE / WORK	Web-Based	Agent-Based	Web- and Agent-Based	Synchronous	Constraint-Based	Ontology-Based	Conflict Handling
Huang et al. (2004)			✓				
Bilek et al. (2006)		✓		✓		✓	
Slimani (2006)			✓	✓	✓		✓
Huang et al. (2007)	✓						✓
Mahesh et al. (2007)			✓				✓
Alvares et al. (2008)			✓		✓		
Wang et al. (2009)		✓			✓		✓

Table 2.1 – Summary of Collaborative System Features

Reuse and synchronicity are efficient strategies to mitigate conflicts in collaborative design. As we have seen, a number of systems invest in the synchronicity of their strategies. Others are more focused on avoiding collaborative conflicts, either by the use of constraints, or by the use of ontologies. However, we could not find a project that combines all of those characteristics together (see Table 2.1).

In this work we propose a generic, synchronous, Web-, agent-, constraint- and ontology-based approach for collaborative design, focusing on conflict handling. We have enriched our proposal with ontology-based knowledge representation, which we are going to explore further in the next chapter.

Chapter 3

3 Ontology-based Knowledge Representation

According to the Merriam-Webster Dictionary[9], an Ontology is "a branch of metaphysics concerned with the nature and relations of being; a particular theory about the nature of being or the kinds of things that have existence." An ontology is also a formal explicit description of concepts in a domain of discourse (classes, sometimes called concepts), properties of each concept describing various features and attributes of the concept (slots, sometimes called roles or properties), and restrictions on slots (facets, sometimes called role restrictions). An ontology together with a set of individual instances of classes constitutes a knowledge base. There is a fine line where the ontology ends and the knowledge base begins (Noy et al., 2001).

Simperl (2009) says that the term ontology was introduced to computer science as a means of formalizing the kinds of things that can be talked about in a system or in a context. She also states that with a long-standing tradition in philosophy, ontologies provide knowledge engineering and artificial intelligence support for modeling some domain of the world in terms of labeled concepts, attributes and relationships, usually classified in specialization/generalization hierarchies. With applications in fields such as knowledge management, information retrieval, natural language processing, e-Commerce, information integration or the emerging Semantic Web, ontologies are part of a new approach to building intelligent information systems: they are intended to provide knowledge engineers with reusable pieces of declarative knowledge, which can be – together with problem-solving methods and reasoning services – assembled into high-quality systems in an economical fashion (Simperl, 2009).

Ontologies have been realized as the key technology to shaping and exploiting information for the effective management of knowledge and for the evolution of the Semantic Web and its applications (Kotis et al., 2004). In such a distributed setting, ontologies establish a common vocabulary for community members to interlink, combine, and communicate knowledge shaped through practice and interaction, binding the knowledge processes of creating, importing, capturing, retrieving, and using knowledge. However, it seems that there will always be more than one ontology even for the same domain (Kotis et al., 2004). In such a setting where different conceptualizations of the same domain exist, information services must effectively answer queries bridging the gaps between their formal ontologies and users' own conceptualizations.

[9] http://www.merriam-webster.com/

Bloehdorn et al. (2009) say that ontologies are considered a key technology enabling semantic interoperability and integration of data and processes. They also believe that we are now entering a phase of knowledge system development, in which ontologies are produced in larger numbers and exhibit greater complexity. Ontology management infrastructures are needed for the increasing development of semantic applications especially in the corporate Semantic Web, which comprises the application of semantic technologies in an enterprise environment.

The continuing diversity of ontologies is partly related to ontologies being aligned with particular views of the world, hence resulting in biases and subjective features. Since the definition of concepts in design and manufacture is dependent of the context or view being taken, this clearly identifies that an all-embracing common basis for ontology construction to be adopted by all parties can prove to be very difficult and time-consuming to realize (Chungoora et al., 2008).

Formally speaking, an ontology is considered to be a pair $O=(S, A)$, where S is the ontological signature describing the vocabulary (i.e. the terms that lexicalize concepts and the relations between concepts) and A is a set of ontological axioms, restricting the intended meaning of the terms included in the signature (Kalfoglou et al., 2003; Kotis et al., 2007). In other words, A includes the formal definitions of concepts and relations that are lexicalized by natural language terms in S. In this definition, conforming to the description logics' terminological axioms, inclusion relations are ontological axioms included in A.

Figure 3.1 shows an example of ontology:

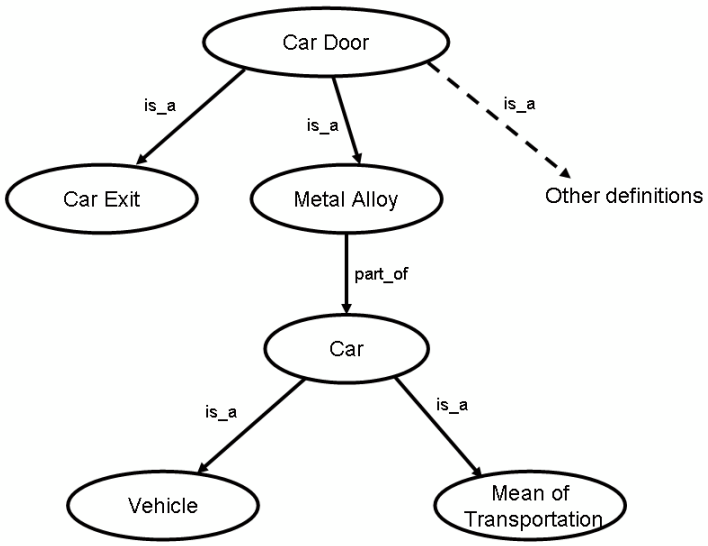

Figure 3.1 – An Example of Ontology

These features of ontologies underscore the main trends that distinguish semantic-integration research in the ontology community. First, since the underlying goal of ontology development is to create artifacts that different applications can share, there is an emphasis on creating common ontologies that can then be extended for more specific domains and applications. If

these extensions refer to the same top-level ontology, the problem of integrating them can be greatly alleviated. Second, since ontologies are developed for use with reasoning engines and semantics of ontology languages are specified with reasoning in mind, inference and reasoning take center stage in ontology-integration approaches.

The remainder of this chapter is organized as follows. In the next section, we present some ontology terminology. Section 3.2 contextualizes the use of ontologies in the Semantic Web. In Section 3.3, we talk about ontologies and interoperability. Section 3.4 presents some integration methods for ontologies. In Section 3.5, some other methods for ontology management are described. Finally, in Section 3.6, we give a short summary of the whole chapter.

3.1 Ontology Terminology

There are many different understandings of the ontology terminology in the literature. In Predoiu et al. (2006) the following terms are used:

- **Ontology:** A widespread definition for the notion of ontology is an explicit specification of a conceptualization. In the following an ontology is considered to be a 4-tuple $<C,R,I,A>$, where C is a set of concepts, R is a set of relations, I is a set of instances and A is a set of axioms.
- **Instance Base:** Although instances are logically part of an ontology, it is often useful to separate between an ontology describing a collection of instances and the collection of instances described by the ontology. We refer to this collection of instances as the Instance Base. Instance bases are sometimes used to discover similarities between concepts in different ontologies.
- **Ontology Language:** Semantic Web ontology languages can be split up into two parts: the logical and the extra-logical parts. The logical part usually amounts to a theory in some logical language, which can be used for reasoning. The logical part basically consists of a number of logical axioms, which form the class (concept) definitions, property (relation) definitions, instance definitions, etc. The extra-logical part of the language typically consists of non-functional properties (e.g. author name, creation date, natural language comments, multi-lingual labels) and other extra-logical statements, such as namespace declarations, ontology imports, versioning, etc. Non-functional properties are typically only for the human reader, whereas many of the other extra-logical statements are machine-processable. For example, namespaces can be resolved by the machine and the importing of ontologies can be achieved automatically by either (i) appending the logical part of the imported ontology to the logical part of the importing ontology to create one logical theory or (ii) using a mediator, which resolves the heterogeneity between the two ontologies.
- **Ontology Mediation:** Ontology mediation is the process of reconciling differences between heterogeneous ontologies in order to achieve inter-operation between data sources annotated with them and applications using these ontologies. This includes the discovery and specification of ontology mappings, as well as the use of these mappings for certain tasks, such as query rewriting and instance transformation. Furthermore, the merging of ontologies also falls under the term ontology mediation.

- **Ontology Mapping:** An ontology mapping M is a specification of the semantic overlap between two ontologies. The correspondences between different entities of the two ontologies are typically expressed using some axioms formulated in a specific mapping language. Mappings can be unidirectional (that specify how terms in one ontology can be expressed using terms from the other ontology) or bidirectional (that work in both ways, i.e. terms from one ontology are expressed using terms from the other and the other way around).
- **Matching:** Ontology matching is defined as the process of discovering similarities between two source ontologies. The result of a matching operation is a specification of similarities between two ontologies. Ontology matching is done through application of the Match operator. Any schema matching or ontology matching algorithm can be used to implement the Match operator. The specification of similarities typically serves as an input to the ontology mapping.
- **Merging Ontologies:** Merging refers to the creation of a final ontology from two or more source ontologies. The new ontology will unify and in general replace the original source ontologies.
- **Aligning Ontologies:** Specifying how the concepts in the different ontologies are related in a logical sense. This means that the original ontologies have not changed, but that additional axioms describe the relationship between the concepts. Leaving the original ontologies unchanged often implies that only a part of the integration can be done, because major differences may require adaptation of the ontologies. This concept is very similar to ontology mapping; however it is a more general concept: two ontologies can be aligned by creating an ontology mapping.

Section 3.4 will provide further details on these integration methods.

3.2 Semantic Web

The Semantic Web (Berners-Lee et al., 2001) is an extension of the current World Wide Web in which information is given precise meaning, making it easy to exchange, integrate and process data in a systematic, machine-automated manner. Using standardized languages, published as W3C recommendations, Semantic Web data can explicitly describe the knowledge content underlying HTML pages, specify the implicit information contained in media like images and videos, or be a publicly accessible and usable representation of an otherwise inaccessible database. The recent popularity of Semantic Web technology has renewed interest in building open, dynamic, and adaptive systems, with a high degree of automation, which supports flexible coordination and collaboration on a global scale (Zhang et al., 2008).

According to Correndo et al. (2008), due to the highly distributed and decoupled nature of the information present on the Web, the vision of the Semantic Web is moving towards a scenario where the task of creating and maintaining ontologies that formalize data semantics is going to be handed to the community that actually uses them. Such an effort must be supported by tools and methodologies that allow latent models to emerge as a product of a collaborative effort and dialogue. To this end, much research and development has focused on building tools and capabilities for ontology and Knowledge Base (KB) construction.

There has been a significant body of recent work on languages for specifying ontologies, software environments for editing ontologies, algorithms for reasoning with, aligning, and merging ontologies. However, the lack of collaborative environments for construction, sharing, and usage of ontologies is a major hurdle to the large-scale adoption and use of ontology-based approaches to sharing of information and resources, which is needed to realize the full potential of the Semantic Web.

3.3 Ontologies and Interoperability

To achieve competitiveness in today's global environment, companies need to engage in more sophisticated business networks and improve collaboration. However, establishing these business networks is an ongoing challenge that companies are facing, due to the diversity and heterogeneity of systems, software applications, and technologies used by the participants. For instance, with so many different modeling and implementation languages being used in systems development, interoperability problems can arise when the chosen product model is described using particular technologies different from the ones the system is required to be integrated with (Agostinho et al., 2006).

Interoperability can be defined as a state in which two application entities can accept and understand data from the other and perform a given task in a satisfactory manner without human intervention (Gulla, 2008). Interoperability characterizes the ability of product model data to accurately represent objects in all collaborative engineering environments. Interoperability and standardization have been playing important roles in lowering costs related to production, sales and delivery processes, which results in reduced final prices and increased competitiveness. Delgado et al. (2006) say that enterprise and system interoperability is frequently associated with the usage of several dedicated reference models, covering many industrial areas and related application activities, from design phase to production and commercialization.

We often distinguish between syntactic, structural and semantic interoperability (Yu et al., 2008; Gulla, 2008):

- Syntactic interoperability denotes the ability of two or more systems to exchange and share information by marking up data in a similar fashion (e.g. using XML).
- Structural interoperability means that the systems share semantic schemas (data models) that enable them to exchange and structure information (e.g. using RDF).
- Semantic interoperability is the ability of systems to share and understand information at the level of formally defined and mutually accepted domain concepts, enabling machine-processable interpretation and reasoning.

Figay et al. (2008) stress that the ATHENA research program on interoperability of enterprise applications highlighted difficulties to use simultaneously solutions coming from different communities when trying to integrate them within piloting activities. For example, semantic mediation prototypes were based on RDF schemas, while service-oriented execution components were based on messages structured according to XML schemas. So it required to

work on the mapping of schema definition languages. Similar issues exist for model interchange format (XML Model Interchange, XML Process Definition Language, etc).

3.3.1 Formal Representation of Knowledge

There are a large number of competitive programs which are utilized in collaborative design, each operating with different file formats and internal languages, i.e. scripting languages and programmatic interfaces. Intricate cooperation relationships among companies around the world complicate this situation, as they introduce security and data management issues (what can be shared with whom?). The question of how to pass product information to partners with whom collaborative protocols are not established and who may at other times be competitors is a big issue for companies. To support this operating environment and to strengthen product information management for the total product lifecycle, enhanced product representations are needed.

Ding et al. (2009) state that in todays' global market the challenge is not only how to utilize information management policies, but also how to develop product representation methods to meet the new demands including platform/application independence, support for the product lifecycle, assisting generation of viewpoint-specific representations, rapid sharing of information between geographically distributed applications and users, and protection of commercial security (intellectual property).

To Hepp (2006), the most obvious process that would benefit from a machine-readable representation of products and their properties is the search for potential matches between buyers and sellers and finding the best one (even if there is no perfect match), which is known as matchmaking. We can also add that a machine-readable representation of products provides us with a very useful manner of detecting collaborative conflicts in early stage design. This formalization also enables the process of automatic resolution of conflicts by reasoning (Chapters 6 and 7).

3.3.2 Standards for Interoperability

A proven approach to integrate heterogeneous systems is to enable interoperability, leveraging the adoption of data standards and enabling the integration of systems across organizations. Interoperability and standardization activities are playing an important role in lowering costs and prices, and increasing competitiveness of organizations. To help attain enterprise and the desired systems interoperability, several dedicated reference models covering many industrial areas and related application activities – from design phase to production and commercialization – have been developed.

- Standard for the Exchange of Product Model Data (STEP): STEP (ISO10303) is a multi-part open data exchange standard for the computer-interpretable representation of product information and for the exchange of product data under the manufacturing domain. The objective of STEP is to provide a means to describe product data throughout the life cycle of a product that is independent from any particular computer system. The nature of this description makes it suitable not only for neutral file exchange, but also as a basis for implementing product databases and retrieving data

(Agostinho et al., 2006; Dutra et al., 2007). STEP Application protocols (APs) are information models that capture the semantics of a specific industrial requirement and provide standardized structures, within which, data values can be understood by a computer implementation; STEP contains more than forty APs. These are described using ISO10303-11, which represents STEP's modeling language, commonly known as EXPRESS (STEP Part 11, 2002).

Many kinds of information standards exist that cope with different needs like product data modeling, electronic business documents, information exchange, etc. Dutra et al. (2007) have identified some of the most important standards that can leverage STEP's usage and adoption:

- XML Schemas (XSD): XSD is the XML modeling language, broadly used on the definition of technology architectures and structures, over the Internet and on Internet-based applications. It has quickly become the language of choice for e-business interoperability. XML is simultaneously human and machine readable. This feature makes it very valuable because, by providing a way for a wide range of data to be exchanged directly by computer systems in a human readable format, it dramatically lowers the cost of development. In addition, XML is easily extensible and is supported by numerous software tools. Many applications developed today that import or export data use or support some form of XML format. In order to take advantage of XML's massive adoption and flexibility, ISO has developed a standard for representing EXPRESS schemas and instances in XML, namely ISO10303-28 (Part 28).
- Unified Modeling Language (UML): UML is widely used by application developers, and as Agostinho et al. (2006) emphasize, has become a "de facto standard" for software engineering. It's a language for specifying, visualizing, constructing, and documenting the artifacts of software systems, as well as for business modeling and other non-software systems. UML represents a collection of the best engineering practices that have proven successful in the modeling of large and complex systems. Compared to STEP, UML related tools abound, which facilitates the task of many organizations that want to use it. UML provide mechanisms, like class diagrams, that can facilitate the modeling process. As with XML – ISO10303-25 (Part 25) –, is an ISO standard which defines the rules for mapping EXPRESS schemas to UML models.
- W3C Semantic Web Standards: Semantic content is not only restricted to the Semantic Web. EXPRESS information models also contain rules and constraints that applications can use to test data sets for correctness through conformance checking mechanisms. These prevent the propagation of incorrect data from one application to another. The emerging Semantic Web technologies, like OWL and RDF, have brought new possibilities for the exploitation of STEP Application Protocols at the knowledge level and on the Web (Dutra et al., 2007).
- Information Exchange Standards: One of the principal foundations for interoperability is to enable the exchange of information between ICT systems in an efficient and automatic way (Delgado et al., 2006). As mentioned before, STEP data – i.e. an instance population of an EXPRESS schema – is traditionally exchanged using ISO10303-21 (Part 21 of STEP). ISO10303-28 (Part 28) has become more popular in the last years due to the use of XML instead of ASCII based files (Agostinho et al., 2006). Other widespread information exchange standards are Electronic Data

Interchange (EDI), and the W3C SOAP. XMI[10] (XML Metadata Interchange) is another neutral format, described in XML, and used for the exchange of UML models.

The Semantic Web architecture defines at its bottom a simple, and at the same time, extremely flexible data model, the Resource Description Framework (RDF)[11]. Based on RDF data, which can be used to annotate Web pages and export data from legacy sources, ontologies and rules represent the two main components in the Semantic Web vision – the heart of the Semantic Web –, which shall enable integrating and making new inferences from existing data. While there are already standard languages for ontologies recommended by W3C – like RDFS[12] and OWL[13] – there is no standard for a rule language available yet (Eiter et al., 2008). Many rule languages and systems have been proposed, and they offer varying features to reason over Semantic Web data. To mitigate the situation, the Rule Interchange Format (RIF)[14] working group of the W3C is currently developing a standard exchange format for rules on the Web that takes language features into account, but is less concerned with committed semantics. Let us review the basics of RDF, SPARQL, RDFS, Description Logics, OWL, and SWRL:

- Resource Description Framework (RDF): RDF defines the data model for the Semantic Web. Driven by the goal of least possible commitment to a particular data schema, the simplest possible structure for representing information was chosen – labeled, directed graphs. An RDF dataset – i.e., an RDF graph – can be viewed as a set of the edges of such a graph, commonly represented by triples – or statements – of the form:

  Subject Predicate Object

  where
  - the edge links *Subject*, which is a *resource* identified by a URI or a *blank node*, to *Object*, which is either another resource, a blank node, a *datatype literal*, or an *XML literal*;
  - *Predicate*, in RDF terminology referred to as *property*, is the edge label.

- SPARQL Query Language for RDF: SPARQL is the W3C standard language for querying RDF data. A query in this language can be roughly divided in two parts: (i) the retrieval part (*body*) and (ii) the result construction part (*head*). The first part of a SPARQL query consists of namespace prefix declarations, which are used in the `where` part in the body to shortcut IRI literals. The body part of a SPARQL query offers the following features. An RDF `dataset`, i.e., the set of source RDF graphs used as input data, is specified using multiple `from` or `from named` clauses. Merging multiple RDF sources specified in consecutive `from` clauses is a crucial feature of SPARQL, which complements the lack of this possibility in plain RDF format. The `where` part lets us match parts of the RDF dataset at hand, by specifying a graph

[10] http://www.omg.org/technology/documents/formal/xmi.htm

[11] http://www.w3.org/RDF/

[12] http://www.w3.org/TR/rdf-schema/

[13] http://www.w3.org/TR/owl-features/

[14] http://www.w3.org/2005/rules/wiki/RIF_Working_Group

pattern possibly involving variables (variable symbols are prefixed with a question mark sign). This pattern is given in a Turtle-based syntax, in the simplest case by a list of consecutive triple patterns, i.e., triples containing variables, IRIs, blank nodes, and RDF literals. More involved patterns allow unions of graph patterns, optional matching of parts of a graph, matching of named graphs selected in `from named` clauses, etc. Finally, variable bindings matching the `where` pattern in the source graphs can be ordered, but also other solution modifiers such as `limit` and `offset` are allowed to restrict the number of solutions considered in the result. In the head portion, SPARQL lets you specify one of four query forms. Each one is associated with a specific result format representing a view over the solutions of the pattern matching mechanism. The three most-used query forms are `construct`, `select`, and `ask`; the `describe` query form can be used to get RDF graphs describing resources, but no formal semantics are defined for this operator and the output depends on the used SPARQL implementation, hence we omit a discussion here. The `construct` query form takes a template as parameter, which consists of a list of triple patterns in Turtle syntax, possibly involving variables that carry over bindings from the `where` part. This operator can be used to translate between different RDF formats. The `select` query form is used to retrieve the bindings for variables mentioned in the graph pattern of the `where` part. The `ask` query form returns true, if there is a binding for the supplied graph pattern, or false otherwise. An example for a simple SPARQL `select` query is

```
PREFIX humans: <http://www.univ-
lyon1.fr/2007/01/07/humans.rdfs#>
SELECT display xml *
WHERE
{
?x humans:hasSpouse ?y
?x rdf:type humans:Male
}
```

which retrieves all ?x of `Male` type, and their spouses (?y).

- RDF Schema (RDFS): RDFS is a semantic extension of basic RDF. In a nutshell, by giving special meaning to the properties `rdfs:subClassOf` and `rdfs:subPropertyOf`, to `rdfs:domain` and `rdfs:range`, as well as to several types – like `rdfs:Class`, `rdfs:Resource`, `rdfs:Literal`, `rdfs:Datatype`, etc. –, RDFS lets you express simple taxonomies and hierarchies among properties and resources, as well as domain and range restrictions for properties. The axiomatization of RDFS can, to a large extent, be approximated by a set of sentences of first-order logic.

- Description Logics: The next layer in the Semantic Web stack (Figure 3.2) serves to formally define domain models as shared conceptualizations, also often called ontologies, on top of the RDF/RDFS data model. In order to formally specify such domain models, the W3C has chosen a language which is close to a syntactic variant of an expressive but still decidable Description Logic (DL), namely SHOIN.

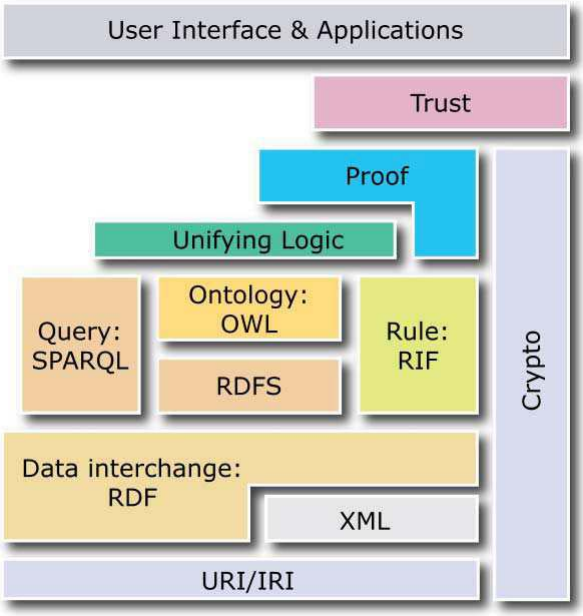

Figure 3.2 – Semantic Web Stack

- The Web Ontology Language (OWL): 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 as in RDFS, 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; 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 unique in that it 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. One of the main benefits of OWL is the support for automated reasoning, and to this effect, it has formal semantics based on Description Logics (DL). OWL is a family of richer ontology languages consisting of OWL Lite, DL and Full. They augment RDF Schema and are based on the descriptions logics (DL) paradigm. OWL Lite is the simplest of these. It is a limited version of OWL DL, enabling simple and efficient implementation. OWL DL is a richer subset of OWL Full for which reasoning is known to be decidable so complete reasoners may be constructed, though they will be less efficient than an OWL Lite reasoner. OWL Full is the full ontology language, which is however undecidable. While RDFS itself may already be viewed as a simple ontology language, OWL adds several features beyond RDFS' simple capabilities to define hierarchies (`rdfs:sub- PropertyOf`, `rdfs:subClassOf`) among properties and classes. As for properties, OWL lets you specify transitive, symmetric, functional, inverse, and inverse functional properties. Moreover, OWL allows the specifications of complex class descriptions to be used in `rdfs:subClassOf` statements. Complex descriptions may involve class definitions in terms of union or intersection of other classes, as well as restrictions on properties. Finally, OWL lets you express explicit equality or inequality relations between individuals by means of the `owl:sameAs` and `owl:differentFrom` properties.

- Semantic Web Rule Language (SWRL): SWRL was recently proposed to increase the power of OWL-based ontology. The SWRL rules provide procedural knowledge, which compensates for some of the limitations of ontology inference, particularly in identifying semantic relationships between individuals (Horrocks et al., 2004). This language is a non-hybrid coupling approach to rules and ontologies. Since SWRL is an extension of the OWL ontology language, it is restricted to unary and binary DL-predicates. Furthermore, it does not support non-monotonic inference (Eiter et al., 2008). SWRL utilizes the typical logic expression "Antecedent→Consequent" to represent semantic rules. Both antecedent (rule body) and consequent (rule head) can be conjunctions of one or more atoms written as "$atom_1$ ^ $atom_2$..^ $atom_n$". Each atom is attached to one or more parameters represented by a question mark and a variable (e.g., ?x). The most common uses of SWRL include transferring characteristics and inferring the existence of new individuals (Chi, 2009). The addition of rules is also the main cause why reasoning in SWRL is in general undecidable, but decidable fragments are known, like DL-safe rules (Eiter et al., 2008). Combining OWL data from outside ontologies is only possible through `owl :import` constructs. A SWRL ontology is composed of ordinary OWL axioms and SWRL rules. The rules are constituted of atoms. Atoms may be OWL class expressions, property definitions, or built-ins. Usually, SWRL rules are part of an OWL ontology encoded in XML or in abstract syntax. For instance, take the creation of a SWRL rule that expresses the variable *Y* as being the father of the variable *X*. Still, *Y* is brother of *Z* and *Z* is *X*'s uncle. This rule demands the existence of the properties *hasFather*, *hasBrother* and *hasUncle* in our OWL ontology. In human readable SWRL syntax this rule is writen like this:

$$hasFather\ (?x_1, ?x_2) \wedge hasBrother\ (?x_2, ?x_3) \rightarrow hasUncle\ (?x_1, ?x_3)$$

Its execution lets us deduce *hasUncle* relations among the defined instances in the knowledge base represented by the given ontology. In SWRL computational syntax, which is a combination of the OWL XML Presentation Syntax with the RuleML[15] XML syntax, this rule would look like this:

```
<ruleml:imp>
  <ruleml:_rlab ruleml:href="#example1"/>
  <ruleml:_body>
    <swrlx:individualPropertyAtom  swrlx:property="hasFather">
      <ruleml:var>x1</ruleml:var>
      <ruleml:var>x2</ruleml:var>
    </swrlx:individualPropertyAtom>
    <swrlx:individualPropertyAtom  swrlx:property="hasBrother">
      <ruleml:var>x2</ruleml:var>
      <ruleml:var>x3</ruleml:var>
    </swrlx:individualPropertyAtom>
  </ruleml:_body>
  <ruleml:_head>
    <swrlx:individualPropertyAtom  swrlx:property="hasUncle">
      <ruleml:var>x1</ruleml:var>
      <ruleml:var>x3</ruleml:var>
    </swrlx:individualPropertyAtom>
  </ruleml:_head>
</ruleml:imp>
```

[15] http://ruleml.org/

Another powerful functionality of SWRL is its capacity to support a great number of complements – defined by the user –, the built-ins. These complements significantly expand the expressivity of SWRL. Some of them are based on comparison functions (*equal*, *lessThan*, etc.), on mathematical functions (*add*, *multiply*, *sin*, etc.), on Boolean functions (*booleanNot*), on character string processing functions (*stringLength*, *substring*, etc.), among others. The following example shows us a SWRL rule with the comparison complement *greaterThan*, to determine if an individual has an elder brother. We can notice the use of the namespace "swrlb" before the complement.

$$hasBrother\ (?x_1, ?x_2) \wedge hasAge\ (?x_1, ?age_1) \wedge hasAge\ (?x_2, ?age_2)$$

$$\wedge\ swrlb{:}greaterThan\ (?age_2, ?age_1) \rightarrow hasElderBrother\ (?x_1, ?x_2)$$

3.3.3 Formalization of Standards

Data integration and exchange systems offer a uniform interface to a multitude of data sources and the ability to share data across multiple systems. These systems have recently enjoyed significant research and commercial success (Dong et al., 2009). Current data integration systems are essentially a natural extension of traditional database systems in that queries are specified in a structured form and data are modeled in one of the traditional data models (relational, XML, etc). In addition, the data integration system has exact knowledge of how the data in the sources map to the schema used by the data integration system.

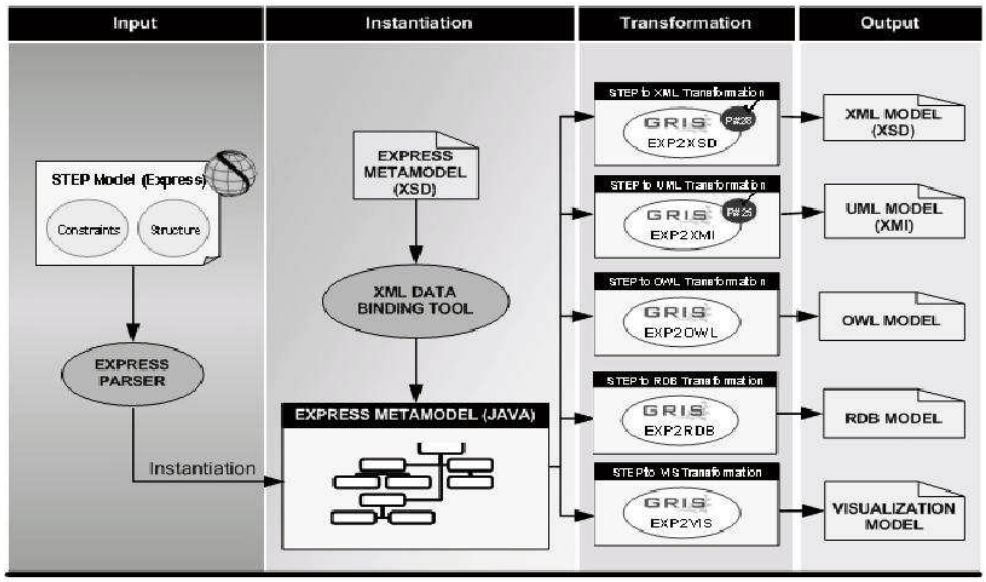

Figure 3.3 – STEP-based Harmonization Famework

Agostinho et al. (2006) have proposed a framework for the harmonization of STEP conceptual models with more popular technologies. It allows the transformation of STEP AP models, originally described using the EXPRESS modeling language, into some of the different technologies presented in Section 3.3.2. The methodology followed to achieve the desired harmonization comprises four different phases which correspond to layers of the framework: input, instantiation, transformation and output (Figure 3.3).

They argue that the knowledge contained in STEP standards can be transposed to complementary modeling and implementation technologies and made available worldwide, as it could ease and reduce the costs related to interoperability. This framework allows the transformation of STEP AP (Application Protocols) models, originally described using the EXPRESS (ISO 10303- 11) modeling language into different technologies (e.g. XML, UML, etc.). It was designed with extensibility in mind, so one can easily plug in more translators according to specific needs, as long as their implementations follow specific rules. Currently it integrates six translator tools that implement the following morphisms:

- EXP2XSD – Translates a STEP model into XML Schema format, according to the mapping rules defined by Part 28 of STEP.
- EXP2XMI – Translates a STEP model into XMI format, according to the mapping rules defined by Part 25 of STEP.
- EXP2OWL (Dutra et al., 2007) – Translates a STEP model into OWL format. The standard rules for performing this translation are currently under development.
- EXP2RDB – Translates a STEP model into a SQL schema format.
- UML2JAVA – Translates a UML model into a set of JAVA classes.
- XML2JAVA – Translates an XML Schema model into a set of JAVA classes.

The EXP2OWL translator was developed within the scope of this thesis, during a seven-month stay in Portugal. In Chapter 6, we are going to present this tool in detail, and how it can be useful to our collaborative architecture.

3.3.4 Ontology Tools

DLs are typically a decidable subset of First Order Logic (FOL), being restricted to the 2-variable fragment of FOL and including counting quantifiers, and are formalisms tailored towards knowledge representation, i.e., they are suitable for representing structured information about concepts, concept hierarchies and relationships between concepts. The decidability of the logic ensures that sound and complete DL reasoners can be built to check the consistency of an OWL ontology, i.e., verify whether there are any logical contradictions in the ontology axioms. Furthermore, reasoners can be used to derive inferences from the asserted information, e.g., infer whether a particular concept in an ontology is a subconcept of another (a.k.a. concept classification), or whether a particular individual in an ontology belongs to a specific class (a.k.a. realization). 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[16] provides a formal definition of ontology consistency.

[16] OWL Web Ontology Language Semantics and Abstract Syntax (http://www.w3.org/TR/owl-semantics/)

- 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.

Popular existing DL reasoners in the OWL community include (Mei et al. 2004; Description Logics Reasoners, 2009):

- Pellet: is a free open-source Java-based reasoner with simple datatypes. 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 and incomplete support of ABoxes. 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.

In addition to reasoners, numerous OWL ontology browsers/editors such as Protégé[17], KAON,[18] and Swoop[19] have been built to aid in the design and construction of OWL ontology models. The latter – Swoop – has been developed as part of the work of Kalyanpur (2006). Most of these OWL tools have expanded their functionality beyond basic editing to include features such as change management and query handling, and in a lot of cases included a reasoner for consistency checking of the ontology. For example, Swoop has integrated Pellet for reasoning and additionally provides the ability to automatically partition, collaboratively annotate and version control OWL ontologies (Kalyanpur, 2006).

Ontology discovery is facilitated by the use of URIs and Web-accessible ontological sources, which are core principles of the Semantic Web. Semantic Web search engines such as

[17] http://protege.stanford.edu/

[18] http://kaon2.semanticweb.org/

[19] http://code.google.com/p/swoop/

Swoogle[20], Watson[21] or OntoSearch[22] and ontology repositories such as the DAML ontology library, the Protege OWL library, the SchemaWeb directory, or the FIPA ontology service are equally important (Simperl, 2009). Falconer et al. (2007), in the ontology user survey undertaken by them, provide us with relevant information concerning the number of concepts used by the users (Figure 3.4), ontology type usage (Figure 3.5), and ontology mapping use cases (Figure 3.6).

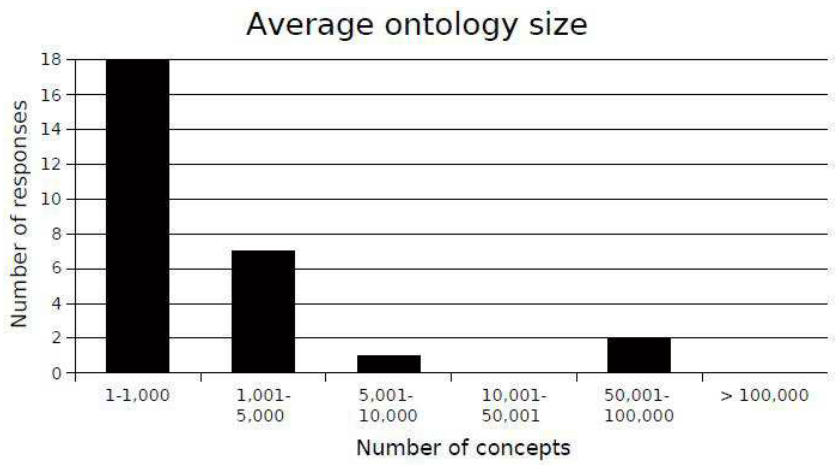

Figure 3.4 – Number of Concepts Used (Falconer et al., 2007)

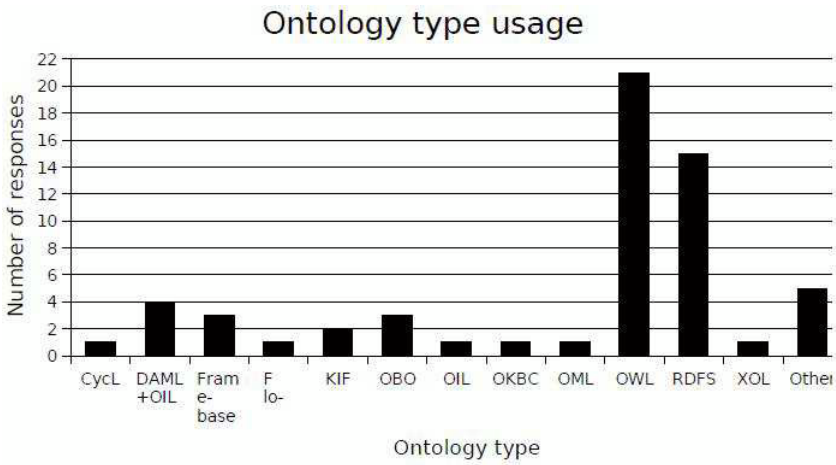

Figure 3.5 – Ontology Language Usage (Falconer et al., 2007)

[20] http://swoogle.umbc.edu/

[21] http://watson.kmi.open.ac.uk/WatsonWUI/

[22] http://www.ontosearch.com/

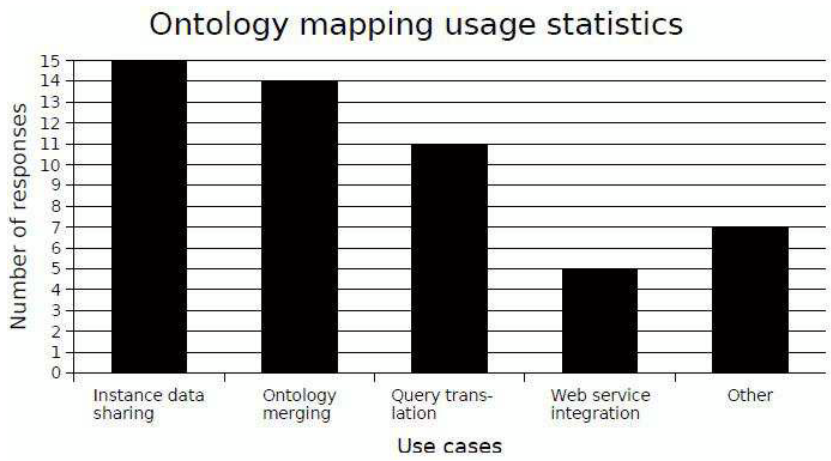

Figure 3.6 – Ontology Mapping Use Cases (Falconer et al., 2007)

3.4 Ontology Integration Methods

Ontologies enable shared knowledge and reuse where information resources can be communicated between human or software agents. Semantical relationships in ontologies are machine readable; in such a way that they enable making statements and submitting queries about a subject domain due to the use of a conceptualization, which describes entities and their relationships. This conceptualization enables software agents with a vocabulary to represent and to communicate knowledge. The usefulness of ontologies in agent based systems can be briefly summarized as they enable knowledge-level interoperation. In other research areas, ontologies support shared understanding, interoperability between tools, systems engineering, reusability and declarative specification.

The next two sections will discuss in detail the ontology mapping, merging, and aligning approaches. Section 3.4.3 will depict the harmonization method for ontology integration.

3.4.1 Mapping

Ontology mapping from ontology $O1 = (S1, A1)$ to $O2 = (S2, A2)$ is a morphism $f{:}S1 \rightarrow S2$ such that $A2 \models f(A1)$, i.e. all interpretations that satisfy $O2$'s axioms also satisfy $O1$'s translated axioms (Kalfoglou et al., 2003). Instead of a function, we may also articulate a set of binary relations between the ontological signatures. Such relations can be the inclusion ($\sqsubseteq$) and the equivalence ($\equiv$) relations. Then we have indicated an alignment of the two ontologies. Instead of aligning two ontologies directly through their signatures, we may specify the alignment of two ontologies O1 and O2 by means of a pair of ontology mappings from an intermediate source ontology O3 (Kalfoglou et al., 2003). Then, the merging of the two ontologies can be considered as the minimal union of ontological vocabularies and axioms with respect to the intermediate ontology where ontologies have been mapped. The merging process takes into account the mapping results in order to resolve problems concerning name conflicts, taxonomy conflicts, etc between the merged ontologies. Therefore, the merging of ontologies can be defined as follows: "Given two source ontologies O1 and O2, find an alignment between them by mapping them to an intermediate ontology, and finally merge

them by getting the minimal union of their vocabularies and axioms with respect to their alignment" (Kotis et al., 2007).

Predoiu et al. (2006) distinguish the following phases (in temporal order) in the mapping process:

- Import of ontologies: Ontologies can be specified in different languages, which indicate a need to convert them to a common format in order to be able to specify the mapping. Furthermore, the ontologies need to be imported into the tool, which is used to specify the mapping.
- Finding Similarities: Many systems use the Match operator to automatically find any similarities between schemas or ontologies. For any two source ontologies, the Match operator returns the similarities between the ontologies. This phase is distinguished in the mapping process only when the similarities are determined in an automatic fashion. If the mapping process is completely manual, this phase is skipped.
- Specifying Mapping/Merging: After (potential) similarities between ontologies have been found, the mapping between the ontologies needs to be specified. This specification is usually a manual process, but it can be aided by a tool. PROMPT (Noy et al., 2000); for example, comes up with concrete proposals for merge operations, so that for many operations the user only needs to say "execute", instead of having to specify the complete operation.

Most mapping languages allow defining the correspondences of mappings between ontology elements of the same type, e.g. classes to classes, properties to properties, etc. These mappings are often referred to as simple or one-to-one mappings. While this already covers many of the existing mapping approaches, there are a number of proposals for mapping languages that rely on the idea of view-based mappings and use semantic relations between queries to connect models, which leads to a considerably increased expressiveness (Bloehdorn et al., 2009).

3.4.2 Merging and Aligning

Structural matching involves matching the neighborhoods of ontology concepts (structure of nodes), providing evidence for the similarity of the nodes themselves. In this way, the similarity between two concepts in a tree-like structure is computed based on the similarity of their descendants in the tree structure, i.e. two non-leaf elements are structurally similar if their immediate children sets are highly similar (Kotis et al., 2007). Semantic matching explores the mapping between the meanings of concept specifications by exploiting domain knowledge. Semantic matching also specifies a similarity function in the form of a semantic relation (hyperonym, hyponym, meronym, part-of, etc) between the intension (necessary and/or sufficient conditions) of concepts. Semantic matching may rely on external information found in lexicons, thesauruses or reference ontologies, incorporating semantic knowledge (mostly domain-dependent) into the process. An example is the exploitation of semantic knowledge in the WordNet[23] lexicon by mapping senses to ontology concepts using information retrieval techniques (Kotis et al., 2007).

[23] http://wordnet.princeton.edu/

Ontology alignment is the process of discovering similarities between two source ontologies. This process is generally described as the application of the so-called Match operator (Rahm et al., 2001). The input of the operator is a set of ontologies and the output is a specification of the correspondences between the ontologies. There are many different algorithms which implement the match operator. These algorithms can be generally classified along two dimensions. On the one hand there is the distinction between schema-based and instance-based matching. A schema-based matcher takes different aspects of the concepts and relations in the ontologies and uses some similarity measure to determine correspondence – e.g., Noy et al. (2000). According to De Bruijn et al. (2006), an instance-based matcher takes the instances which belong to the concepts in the different ontologies and compares these to discover similarity between the concepts. On the other hand there is the distinction between element-level and structure-level matching. An element-level matcher compares properties of the particular concept or relation, such as the name, and uses these to find similarities. A structure level matcher compares the structure (e.g., the concept hierarchy) of the ontologies to find similarities. These matchers can also be combined, e.g. Anchor-PROMPT, a structure-level matcher, takes as input an initial list of similarities between concepts. The algorithm is then used to find additional similarities, based on the initial similarities and the structure of the ontologies.

Existing efforts on ontology mapping, alignment and merging vary from methodological and theoretical frameworks, to methods and tools that support the semi-automatic coordination of ontologies. Kotis et al. (2004) consider that only latest research efforts "touch" on the mapping/merging of ontologies using the whole breadth of available knowledge. Based on this statement, they propose the HCONE approach on ontology merging. HCONE is based on (a) capturing the intended informal interpretations of concepts by mapping them to WordNet – a lexical database for the English language – using lexical semantic indexing, and (b) exploiting the formal semantics of the concepts' definitions by means of the description logics' reasoning services.

To a greater extent than existing approaches to coordinating ontologies, the HCONE approach gives much emphasis on "uncovering" the intended informal interpretations of concepts specified in an ontology. Linguistic and structural knowledge about ontologies are exploited by the Latent Semantics Indexing method (LSI) for associating concepts to their intended, informal, human-oriented interpretations realized by WordNet senses (Kotis et al., 2004). Using the concepts' intended semantics, the proposed method translates formal concept definitions to a common vocabulary and exploits the translated definitions by means of the description logics' reasoning services. The goal is to validate the mapping between ontologies and find a minimum set of axioms for the merged ontology.

Generally, the similarity among concepts can be defined in ways that range in a continuum from simple string matching to more elaborated semantic matching approaches. As it is done in other surveys, we distinguish between lexical, structural and semantic matching depending on the kind of knowledge used in the computation of a similarity function i.e. lexical, structural, or semantic, respectively. Lexical matching involves the matching of ontology concept names (labels at nodes), estimating the similarity among concepts using syntactic similarity measures. Minor name variations can lead the matching result astray. For instance, considering the matching between labels "TechReport" and "Technical Report", although they both lexicalize the concept "technical report", a matching may not be established due to the failure of name matching algorithm to identify the similarity (Kotis et al., 2007).

3.4.3 Harmonization

In an open, distributed, and dynamic setting such as the World Wide Web, it is often the case that neither a reference ontology nor a representative set of instances are present. On the other hand, the humans' intended meaning of ontology concepts must always be captured in order for semantics to be exploited during the mapping process. Automating the process is still an open research issue.

Noy (2004) identifies two major architectures for mapping discovery between ontologies.

- In the first one, the vision is that a general upper ontology is agreed upon by developers of different applications, who then extend this general ontology with concepts and properties specific to their applications. As long as this extension is performed in a way consistent with the definitions in the shared ontology, finding correspondences between two extensions can be facilitated by this common "grounding."
- The second set of approaches comprises heuristics-based or machine learning techniques that use various characteristics of ontologies, such as their structure, definitions of concepts, and instances of classes, to find mappings.

Noy (2004) clarifies the difference between ontology merging and ontology aligning. When merging two ontologies, a single coherent ontology is created that is a merged version of the two original ontologies. When aligning two ontologies, the two original ontologies persist, with a number of links established between them, allowing the aligned ontologies to reuse information from one another. The alignment of ontologies is usually part of the ontology merging process. Solutions can be further classified along two dimensions: a run-time and a design-time dimension. The run-time dimension is concerned with the way the user views the data in the system during operation. The design-time dimension is concerned with the way the models of the disparate data sources are integrated. In the run-time, or user-centered dimension, Predoiu et al. (2006) distinguish two approaches: (1) the local model and (2) the global model approach. The difference between these two approaches is whether, in interactions with the system, the user can use his/her own local data model, or whether the user needs to conform to a global model when interacting with the system:

- The Local Model: In this case the user is represented by an agent in the system and this agent represents the user with its own local data model. The agent performs the translation between the user's local model and either the global model or other local models in order to allow interaction with multiple data sources in the system.
- The Global Model: The user views the system through the global data model using a mediator, which is a system that supports an integrated view over multiple information sources. Note that in the local model approach, a user agent will in most cases also contact a mediator in order to allow interoperation with the system, which contains multiple information sources.

Sarraipa et al. (2008) propose MENTOR (Methodology for Enterprise Reference Ontology Development), a methodology to support the development of a common reference ontology

for a group of enterprises sharing a business domain. MENTOR addresses all the referenced categories, i.e. ontology building from scratch; ontology reengineering; cooperative building, and merge methods. MENTOR seeks a more comprehensive knowledge representation life cycle for use in existent semantic interoperability problems, inside a given domain. A tool based on the Protégé Java-based Application Programming Interface (API) was used to implement MENTOR methodology functionalities.

MENTOR provides several step methods as semantic comparisons, basic lexicon establishment, mappings among ontologies and other operations on KB representations. This methodology is composed of two phases: the Lexicon Settlement, and the Reference Ontology Building with three steps each.

The Lexicon Settlement phase (Phase 1) represents a domain knowledge acquisition which compared to the human language apprentice phase could be represented in computer science as an organized semantic structure with definitions. The thesaurus can represent such words as a structure of associated meanings and thus should be built in order to establish the lexicon of a specific domain. This phase has three steps: Terminology Gathering, Glossary Building, and Thesaurus Building. These steps were defined based on the UPON (Unified Process for Ontology Building) based on workflows of iterations related to the ontology building requirements, analysis, design, implementation and testing (De Nicola et al., 2005), which defines a set of workflows that establishes a thesaurus of the domain before starting the ontology building.

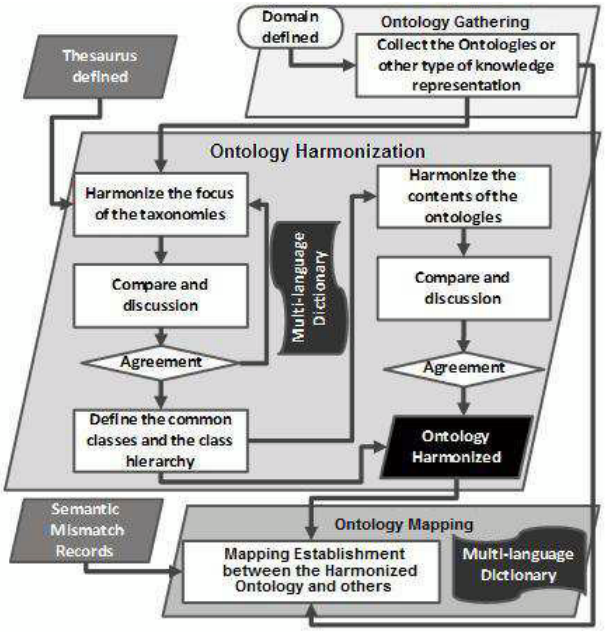

Figure 3.7 – The Reference Ontology Building in MENTOR (Sarraipa et al., 2008)

The Reference Ontology Building phase (Phase 2) is the phase where the reference ontology is built and the semantic mapping between the organizational and reference ontologies is established (Figure 3.7). The first step includes ontology gathering in the domain defined. Another type of knowledge representation could be used as input for the harmonization ontologies process together with the thesaurus defined in the previous phase. The harmonization method for building ontologies proposes the development of a single

harmonized ontology in two cycles where first the structure is discussed until having agreement on it and then the same process for the ontology contents definition. From this process new semantic conflicts could be found. After agreement, the resolution could be recorded in the MENTOR Ontology for further mapping establishments. With all the agreements accomplished, the harmonized ontology is finalized together with the mapping tables, describing the ontological relationships between the harmonized ontology and each one of the individual ontologies. Semantic difficulties related to the natural language of the potential users of the harmonized ontology are likely to happen. To assist with this, the ontology is complemented with a multi-language dictionary where a set of normalized tokens gives the reference to the corresponding concepts and definitions in different native languages (Sarraipa et al., 2008).

3.4.4 Ontology Mismatches

Michel Klein (2001) identifies two levels of mismatches between ontologies. The first level of mismatches is the ontology or model level. The second one is the ontology language or meta-model level. These mismatches include syntactic mismatches, differences in the meaning of primitives in the different languages, and differences in the expressivity of the languages.

The languages can differ in their syntax, but, more important, constructs available in one language (e.g., stating that classes are disjoint) are not available in another. Even semantics of the same language primitives could be different (e.g., whether declaration of multiple ranges of a property has union or intersection semantics). The normalization process therefore often precedes ontology-matching and translates source ontologies to the same language, resolving these differences. However, even for ontologies expressed in the same language, possible ontology-level mismatches abound. A partial list of ontology-level mismatches includes using the same linguistic terms to describe different concepts; using different terms to describe the same concept; using different modeling paradigms (e.g., using interval logic or points for temporal representation); using different modeling conventions and levels of granularity; having ontologies with differing coverage of the domain, and so on (Noy, 2004).

Different types of mismatches can occur between different ontologies. It is important to identify which kind of mismatches can and do occur between ontologies, in order to resolve these mismatches in the mapping or the merge of ontologies. The classification of ontology mismatches is also important to denote which kind of mismatches can be resolved with a particular mapping formalism and which kind of mismatches can be detected with a particular matching algorithm (Predoiu et al., 2006).

Chungoora et al. (2008) say that conceptualization mismatches (Figure 3.8) occur as a consequence of having two or more conceptualizations of a certain domain. These conceptualizations can potentially differ in the way they are defined as ontological entities or in the way they are related within ontologies. According to them, conceptualization mismatches involve:

- Class Mismatches. The different classes and subclasses present in ontologies.
- Categorization Mismatch. This takes place when in two ontologies the same class has been defined but the class possesses different subclasses.

- Aggregation-Level Mismatch. This takes place if in both ontologies the same class has been defined but the latter has varying levels of abstraction (Figure 3.9).
- Relation Mismatches. This type of mismatch is concerned with relations or properties present in ontologies. They involve, for instance, the hierarchical relations between two classes or the assignment of attributes to classes.
- Structure Mismatch. This is likely to happen when in two ontologies; experts have used the same set of classes but have structured the classes differently using relations/properties.
- Attribute-Assignment Mismatch. This form of mismatch is found when the same relation is defined in two separate ontologies, but differs in the way the particular relation is attributed to classes in both conceptualizations.
- Attribute-Type Mismatch. This takes place when in two ontologies, the same relation has been defined, but has varying value types, which consequently affects the range of possible values for the instances in both ontologies, for example, a same relation "hasDimension" can be defined as an object property in one ontology and as a datatype property in another ontology, hence resulting in attribute-type conflicts.

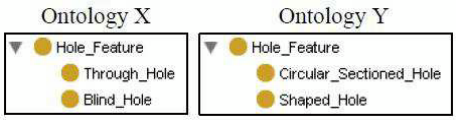

Figure 3.8 – Conceptual Mismatches (Chungoora et al, 2008)

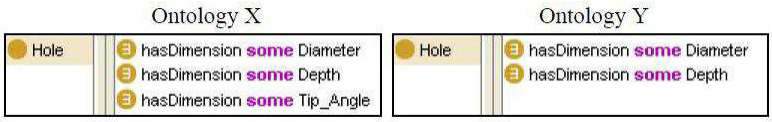

Figure 3.9 – Aggregation-Level Mismatches (Chungoora et al, 2008)

We will now go over the above mentioned language level mismatches and describe what the (potential) issues are with the current systems and techniques:

- Certainly, differences in syntax would be resolved by any such translation to an internal representation, since both ontologies then use the same syntax.
- Differences in logical representation occur when syntactically different, but logically equivalent statements are used to represent the same thing. An example of this is the way disjointness is expressed in the OWL Lite species of the Web Ontology Language OWL (McGuinness et al., 2004), compared to the way disjointness is usually expressed in the OWL DL species. Arguably, this is not really an issue with the language itself, but rather an issue with the use of the language. However, when a language allows the user to model the same thing in different ways, it is easy for a user to mistakenly model certain things in an inconvenient way and it is harder for a user to understand the model created by a different user or indeed created by himself/herself in the past. When the ontology language used by the technique/tool/system allows

such different logical representations of equivalent statements, this mismatch still needs to be taken into account in the ontology mapping process. In order to overcome these issues, one could think of a normalization step before the start of the mapping process or reasoning during the mapping process in order to detect equivalence in logical expressions.

- When the semantics of primitives is different in different ontology languages, i.e. a syntactically equivalent construct has a different meaning in the different languages, the translation to the common representation needs to take this into account. Fortunately, this problem can be resolved in the translation to the common representation. If both ontologies already use the common representation and this common representation does not allow ambiguous statements, this mismatch does not occur.
- Differences in expressivity of the languages are resolved in the translation to the common representation language. However, if the expressivity of the common representation language is not a superset of the language of the source ontology, some semantics might get lost in the translation.

3.5 Ontology Management

In this section, we are going to present some works in the ontology research area. They were classified according to the criteria that we think that are useful to our collaborative architecture.

3.5.1 Genericness

Chungoora et al. (2008) believe that adopting an all-embracing ontology as a basis for sharing meaning, and as a foundation over which to build up information and knowledge exchanges, remains a very unlikely scenario, since in practice, multiple ontologies and schemas will be developed by independent entities. Furthermore, with the widespread distributed use of ontologies, different parties inevitably develop ontologies with overlapping content. These factors, although targeted at the more general problem of ontology genericness, bring evidence of the existence of multiple ontologies developed to suit different functional domains and this is very likely to happen in the design and manufacture stages of the product lifecycle.

As a prerequisite to solving the ontology interoperability issue, it is first vital to understand how varied ontological concepts can be and in which ways ontology and semantic mismatches take place, which impede achieving seamless interoperability. Semantic mismatches can be interpreted from perspectives such as knowledge elicitation, databases and knowledge representation.

Heterogeneous modeling terms should be semantically processed both by design collaborators and intelligent systems. Ontologies in the Semantic Web can explicitly represent semantics and promote integrated and consistent access to data and services. Thus, if an ontology is used in a heterogeneous and distributed design collaboration, it will explicitly and persistently represent engineering relations that are imposed in an assembly design. Design intent can be

captured by reasoning, and, in turn, as reasoned facts, it can be propagated and shared with design collaborators (Kim et al., 2006).

Heterogeneous tools and multiple designers are frequently involved in collaborative product development, and designers often use their own terms and definitions to represent a product design. Thus, to efficiently share design information among multiple designers, a designer's intentions should be persistently captured and the semantics of the designer's terms and intents should be interpreted in a consistent manner. For this purpose, a standardized data format is a prerequisite. Furthermore, the appropriate design knowledge should be provided during all design processes in order to make proper design decisions.

3.5.2 Generic Ontology

Pirró et al. (2009) propose a collaborative architecture that relies upon a generic ontology, previously defined; they call it Upper Ontology (Figure 3.10). This architecture use P2P to enable distributed and collaborative work. It provides a multi-layer ontology framework to enable semantics-driven knowledge processing. The ontology framework allows organizational knowledge to be modeled at different levels. An Upper Ontology is exploited to establish a common organizational knowledge background. A set of Workspace Ontologies is designed to manage, share, and search for knowledge within communities by the establishment of a contextual – related to the aim of a group – understanding.

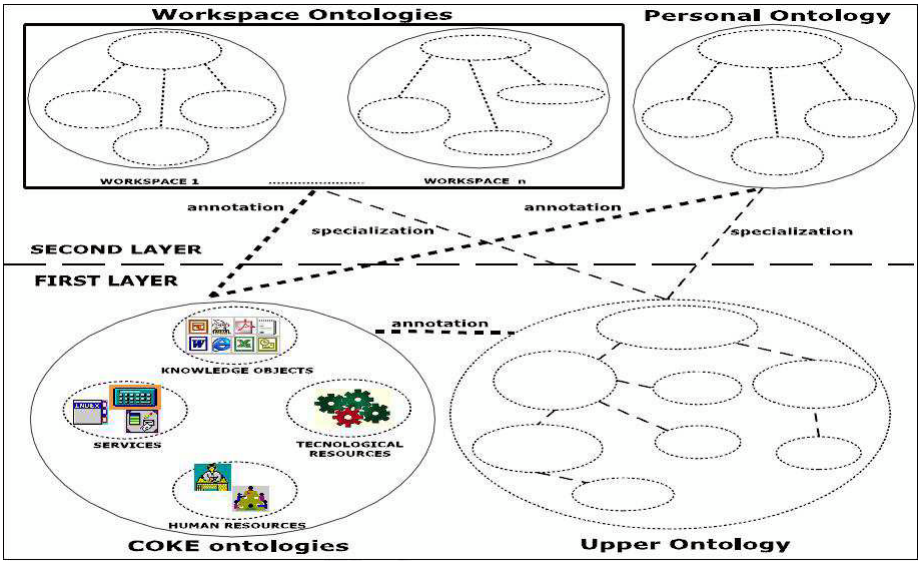

Figure 3.10 – Upper Ontologies (Pirró et al., 2009)

Knowledge engineers have long realized that, for two or more information systems (databases, agents, peers, software components, expert systems, etc.) to be both syntactically and semantically interoperable, they will need to commit to a shared conceptualization of the application domain. Commonly, this is achieved by producing an explicit specification of this conceptualization — what has come to be known in computer science as an ontology — and by defining each system's local language in terms of the ontology's vocabulary (Schorlemmer et al., 2008).

3.5.3 Natural Language

Natural language interfaces are found in many applications for example, database query and information retrieval systems, language translation systems, Web search engines, dictation systems, and command and control systems. The construction of natural language interfaces to computers continues to be a major challenge. The need for such interfaces is growing now that speech recognition technology is becoming more readily available, and people cannot speak those computer-oriented formal languages that are frequently used to interact with computer applications (Frost, 2006).

Pazienza et al. (2005) present an approach to ontology mapping, which is strongly based on the exploitation of (monolingual and multilingual) linguistic resources for content publishing and discovery, and on human intervention for supervising the process and assessing semantic links between mapped resources.

In fact, ontologies structure their content according to the formal semantics of their underlying model as well as to the domain conceptualization which developers are acquainted with. As this conceptualization may change from person to person, the inner meaning of their formal content is inherently bound to the specific applications which rely on them. Mapping the content of two ontologies is thus like committing to a shared interpretation of the two semantic structures, which is a model for both and which is compliant with the applications which insist on them. If the inner meaning of independently developed ontological knowledge is thus a shadowed and ambiguously identifiable concept, the same should not hold for its surface expression, which could at least provide strong evidence for recognizing identities and similarities across different information sources. This is indeed the task which can mostly be supported, being related to the discovery of objective and measurable clues and facts, and thus being easily managed in an automatic approach.

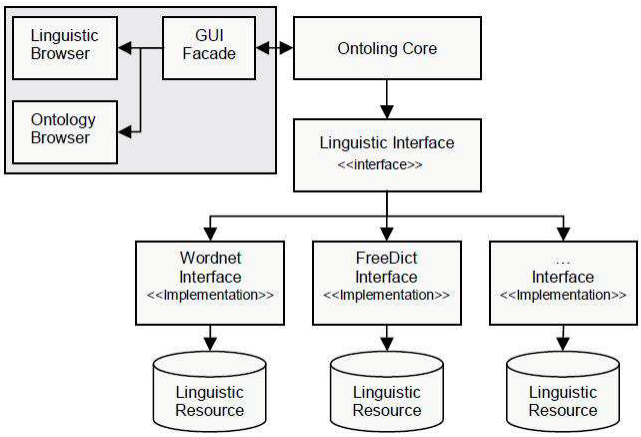

Figure 3.11 – Ontoling Architecture (Pazienza et al., 2005)

Figure 3.11 shows Ontoling (Pazienza et al., 2005), a tool dedicated to assisting ontology developers and users in the process of enriching ontologies with information coming from available linguistic resources. It has been developed as a plug-in for the popular ontology editing tool Protégé.

3.5.4 Rules

The issue of having rules on top or beside ontologies written in OWL is an important milestone on the W3C agenda for completing the Semantic Web architecture (Eiter et al., 2008). Despite theoretical issues that arise, due to the complementary nature of existing ontology and rules languages, an abundance of rule-based systems have been developed over the last years, driven by the need for rule-based integration of constantly growing Semantic Web data; currently, the W3C is designing a unifying exchange format – the Rule Interchange Format (RIF) – for the various existing languages. Ontology languages on their own cannot fulfill all the prescribed requirements; rule languages should overcome at least some of the known obstacles. But such a combination of rules and ontologies, which integrates well with current W3C standards, is not a simple task due to a number of reasons (Eiter et al., 2008).

Chi (2009) demonstrates how an ontology combined with semantic rules can be used for searching for complete business partners in a supply network. Rather than being fully self-sufficient, individual enterprises are often part of a supply chain. Involving partners and understanding the importance of their activities is essential for enterprise agility. The question is how far-reaching and in what capacity a supply network is needed. Partner tracing becomes more difficult if search tasks involve potential partners or conform to future production planning. This study utilized OWL and SWRL technologies to develop supply network ontology and a problem-solving ontology, respectively.

Paschke et al. (2007) present a Rule-based pragmatic agent Web model for virtual organizations. They propose a declarative rule-based service-oriented methodology and a scalable architecture to operationalize a distributed rule-based approach where event-based communication and rule-based use of meaning plays a central role in connecting the various resources and Web-based services/agents in virtual organizations and teams. The addressed application domain of virtual organizations and rule-based services is of high industrial relevance.

They have followed a constructivistic design science research methodology (Paschke et al., 2007) and implemented an improved rule-based agent technology based on a distributed rule management service and a modern enterprise service middleware providing enhanced usability, scalability, and performance, as well as less costly maintenance in engineering and deploying agent/service-oriented architectures. Their rule responder system allows externalizing and publishing rules on the Web, and managing them in various modules deployed as online services/agents which are then weaved into the main applications at runtime.

3.5.5 Other Approaches

Bloehdorn et al. (2009) present a generic system architecture for an ontology management infrastructure that supports the construction of semantic applications. Such an ontology management infrastructure intends to be generic and flexible in its design, comprehensive in its functionality in order to meet the requirements of a wide spectrum of semantic applications. According to them, the following requirements need to be met by such an ontology management infrastructure:

- Integration of Heterogeneous Knowledge Sources: A common limitation is that different knowledge sources use different taxonomies and different metadata schemes to describe the stored content, which requires a user to switch between different representations depending on the back-end system searched. A particular challenge is the integration of structured knowledge sources (e.g., document metadata) and unstructured sources (e.g., document fulltexts). They propose then to access heterogeneous knowledge sources via a single, unified interface that integrates different metadata schemes as well as different topic hierarchies. This integration of heterogeneous knowledge sources is realized using ontology mappings.
- Automatic Content Extraction and Classification: They argue that some support for automatically capturing the content of new documents added to the library seems necessary. Typically, the content of a knowledge source is not static, but changes over time. New documents come in, but also documents may be removed from the document base. They use techniques of incremental ontology learning to automatically extract relevant knowledge and to classify the new content.
- Dealing with Inconsistent Information: In many cases, the information derived from diverse sources leads to inconsistencies. This is especially the case if information is derived using automatic knowledge acquisition tools such as wrappers, information extraction systems, or tools for automatic or semi-automatic ontology learning that aim at the semi- or even fully automatic extraction of ontologies from sources of textual data. Ontology-based applications which rely on learned ontologies have to face the challenge of reasoning with large amounts of inconsistent information resulting from automatic ontology generation systems (Bloehdorn et al., 2009).
- Support for Structured Queries Against Metadata and Documents: They require the capability to pose structured queries to the underlying digital library which can be evaluated against the article's metadata as contained in the knowledge base. In general, they propose to allow the retrieval of facts as well as the retrieval of documents. For metadata queries, current interfaces either offer preselected attribute fields – which are interpreted as conjunctive queries over an attribute-specific fulltext search – or they require some kind of formal query language. In order to provide intuitive access and not to impose the burden on the user of learning a formal query language, they propose allowing the user to use an intuitive query language, ideally arbitrary natural language queries.

Figure 3.12 illustrates the architecture proposed by Bloehdorn et al. (2009). The core of this architecture is the ontology model module. The ontology model, typically an in-memory representation, maintains a set of references to the ontologies, their respective contents and corresponding instance data sources. From a logical perspective, it can be seen as a set of uninterpreted axioms. The ontology model can typically be modified by means of an API or via well-defined interfaces. Ontology editors and ontology learning systems usually access and modify the contents of the ontology model. As a separate component, the ontology metadata store maintains metadata information about the ontologies themselves, for example author, version and compatibility information.

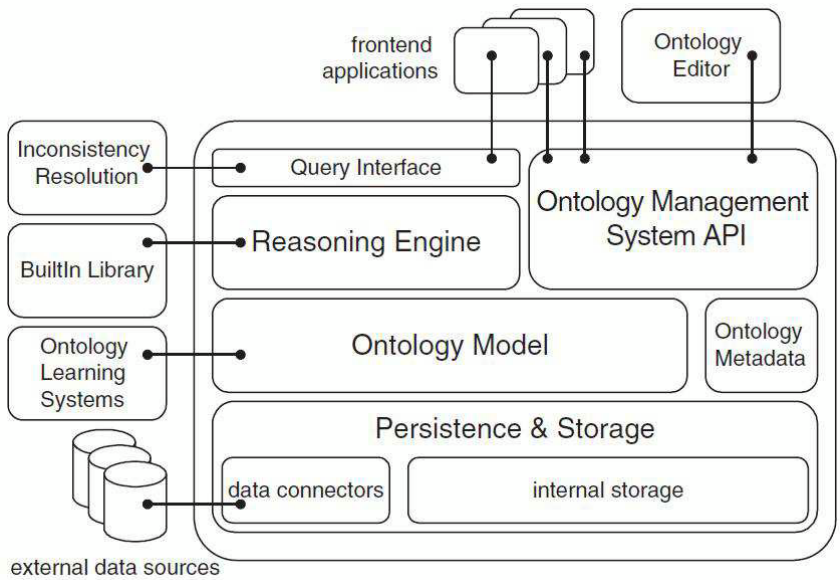

Figure 3.12 – A System Architecture for Ontology Management (Bloehdorn et al., 2009)

While the ontology model – or parts of it – is usually kept in-memory during runtime, the underlying layer is responsible for persistence and storage functionalities. Often, these functionalities will be internal serialization and import/export. However, it may also include connectors to external data sources, e.g. classical relational databases that are interpreted in the context of the ontology model they are embedded in. The reasoning engine operates on top of the ontology model, giving the set of axioms an interpretation and thus deducing new facts from the given primitives. The behavior of the reasoning engine depends on the supported semantics, e.g. Description Logics, in the case of OWL reasoning engines. The interaction with the reasoning engine from outside takes place either by means of a defined API or by means of a query interface. The query interface needs to support a standardized query language, e.g. SPARQL for OWL/RDF data.

Most design models and design ontologies focus on the artifact or object of design. The FBS ontology distinguishes between three aspects of a design object (Gero et al., 2004), function (F), behavior (B) and structure (S) (Kannengiesser et al, 2009):

1. Function (F) of an object is defined as its teleology ("what the object is for"). For example, some of the functions of a window include "to provide view", "to provide daylight" and "to provide rain protection". Function represents the usefulness of the object for another system.
2. Behavior (B) of an object is defined as the attributes that can be derived from its structure ("what the object does"). Using the window example, behaviors include "thermal conduction", "light transmission" and "direct solar gain". Behavior provides operational, measurable performance criteria for comparing different objects.
3. Structure (S) of an object is defined as its components and their relationships ("what the object consists of"). The structure of physical objects includes their form (i.e., geometry and topology) and material. More generally, form can be viewed as a description of an object's macro-structure, and material can be viewed as a shorthand description of the micro-structure. In the window example, macro-structure (form)

includes “glazing length” and “glazing height”, and micro-structure (material) includes “type of glass”.

Ghafour (2009) proposes a method to ameliorate the semantic interoperability of collaborative systems that uses an ontology-based approach and combines OWL DL and SRWL languages. His work focuses on design features. An intermediate ontology has been developed, in order to provide data integration among different CAD systems, the Common Design Feature Ontology (CDFO). This approach is undertaken in two different stages. The first one consists of a homogenization of CAD model representation formats towards a common model, written in OWL DL. According to Ghafour (2009), this homogenization processes the syntactic heterogeneities between the given models. The second stage consists of a rule definition process, in order to find the semantic correspondence between the CAD application ontologies and their common ontology.

The method used by Ghafour (2009) to find the semantic correspondence is based on explicit definitions of axioms and correspondence rules, in order to align all the different ontologies’ entities. This approach is also based on the automatic recognition of semantic correspondences, through the use of the reasoning provided by DL-based inference engines. The semantic correspondence is also enriched by a semantic similarity calculating method – provided by OWL-DL – on the entities’ components, that analyzes their descriptions and contexts.

A semi-automatic methodology to help the discovery of semantic correspondences between different semantic resources is proposed by Ferreira da Silva (2007). She as well uses OWL-DL to provide DL-based inference engines. This work proposes creating equivalences between relations of different semantic resources. These equivalences are supposed to discover other semantic correspondences, and to reason over the concepts (Ferreira da Silva, 2007). The European construction sector was used as a scenario to validate this proposal’s prototype.

Benslimane et al. (2006) work with contextual ontologies. Contextual ontologies are ontologies that characterize a concept by a set of properties that vary according to context. Domain ontologies are developed by capturing a set of concepts and their links according to a given context. A context can be seen from different perspectives. For instance it could be about abstraction level, granularity scale, interest of user communities, and perception of ontology developer. Therefore, the same domain can have several ontologies, where each ontology is described in a particular context (Benslimane et al., 2006). Another approach for contextual ontologies is the work of Hoffmann (2008). He proposes a methodology to determine, model, and use contexts. This methodology contains three different uses of context in order to ameliorate the ontology reconciliation (integration):

- Disambiguate the possible pragmatic senses of concepts, comparing the “perspectives” that were used to develop these concepts.
- Personalize the agent contexts, through a pertinent selection of organization tasks and domains.
- Evaluate the pertinence of data associated to the concept, for the task that has given the interoperability requirement.

Correndo et al. (2008) describe a prototype for data integration based on collaborative and social ontology mapping, their system allows: aligning local ontologies to shared ones; exploiting social interaction and collaboration to improve alignment quality; reusing user ontology alignment information for enhancing future automated alignments; and querying heterogeneous data sources.

3.6 Summary

Traditionally, the geometric model was in the center of the product development process. Geometry has always played a great role since the early years of collaborative design. Nevertheless, the increasing need for semantic knowledge representation of the modern industrial systems has shed light on ontology-based methodologies and strategies.

A large number of projects have come about in the recent years in the ontology domain. Different kinds of proposals – ranging from static mapping of semantic correspondence to dynamic reasoning through the use of more and more sophisticated inference engines – have shown us the feasibility and advantages of using ontologies.

INTEGRATION METHOD / PROJECT	Mapping	Merging and Aligning	Harmonization	Reuse-Based	Generic Ontology	Rule-Based
Noy et al. (2000)		✓		✓	✓	✓
Pazienza et al. (2005)	✓	✓			✓	
Kotis et al. (2007)	✓	✓	✓			
Chungoora et al. (2008)	✓	✓			✓	✓
Correndo et al. (2008)	✓	✓		✓	✓	✓
Sarraipa et al. (2008)	✓	✓	✓	✓		
Chi (2009)		✓			✓	✓
Pirró et al. (2009)	✓			✓	✓	

Table 3.1 – Summary of Integration Methods

Table 3.1 shows us the most used methods for ontology management and integration. A huge gap can be noticed when considering the harmonization approach, in comparison with the use of a generic (or domain) ontology. We consider that a good solution for our architecture should necessarily combine both approaches, in order to enhance the system's outputs.

We consider the ontology-based representation a key characteristic in our architecture. We intend to merge together the designer's ontology instances in order to achieve a common product model. This process is done after a reasoning process, which tries to find collaborative conflicts in early stages of design.

Ontologies possess many advanced structures and functions that using ontologies is a very suitable way to mitigate collaborative conflicts, and to solve them whenever prevention is not possible. This point will become clearer in the next chapter, when we are going to present a state-of-the-art on collaborative design conflicts.

Chapter 4

4 Conflicts in Collaborative Design

In the literature, it is common to notice the recurrent appearance of development and design conflicts (Fulczyk et al., 2002). These conflicts become more explicit and more complex inside collaborative development environments, where design and development tasks and their associated knowledge are distributed among several actors, each one of them having his/her own constraints and viewpoints (Slimani et al., 2006).

Matta et al. (1996) state that, if on the one hand, it is difficult to detect conflicts and their nature precisely, on the other hand, it is complex to determine appropriate methods to solve them. They believe that, at least, two classes of conflicts can be handled in Concurrent Engineering. First, *conflicts between design and requirements* – in other words, when the design made by a participant does not satisfy the corresponding requirements; second, *disagreements between designers*. Such conflicts arise from problems caused by strategies used and propositions made by designers.

Usually, the gain that can be obtained in relation to cost, development delay, and more generally, the advantages provided by an integrated development approach, relies essentially on the knowledge and constraints from every development process actor, which should be taken into account from the very beginning of the process. These sets of individual knowledge, however, may lead to conflicting situations during the design process. These situations may ultimately slow down the whole development process. In this context, conflict management, i.e., the detection and resolution of conflicts – preferably as soon as possible – is a fundamental task to be done, in order to provide support to collaborative environments. Because usually design conflicts were often not detected until long (days to months) after they had occurred, it resulted in wasted design time, design rework, and even scrapped tools and parts. Design rework rates of 25-30% were typical (Klein, 2000). Klein also remarks that roughly half of the labor budget for the Boeing 777 program – which is measured in the hundreds of millions of dollars – was estimated to be due to changes, errors and rework, often due to design conflicts.

4.1 Definitions

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 designer's disagreement upon 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 at the same time in 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. Publishing a proposition means putting 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.

Ferreira da Silva et al. (2004) also consider a conflict as a disagreement between one or more actors involved in the realization of a given task, unsatisfactorily accomplished, during a design process. More precisely, a conflict is brought about by a disjunction between the current solution's state and the constraints brought in by some particular actor (designer, manufacturer, etc.), who evaluates them negatively. Thus, a conflict can be seen essentially as a viewpoint confrontation process, which leads ultimately to a consensus.

So, conflict attenuation is an important point to be considered. Conflicts can be extreme 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 parties, in order to put the situation under control as soon as possible. These three steps are called strategies of conflict attenuation (Matta et al., 1996).

During collaborative designs, conflicts and disputes arise regularly in decision-making processes such as goal selection, proposal exchanging, task co-ordination, role playing, limited resource allocation, etc. Unmanaged or improperly managed conflicts not only affect the productivity of the whole design team but also hamper achieving the design goal. Conflict resolution consists of at least five steps: conflict detection, conflict identification, negotiation team formation, solution generation and solution evaluation (Wang et al., 2002). Conflict resolution expertise consists of a taxonomy of design conflict classes in addition to associated general advice suitable for resolving conflict in these classes (Klein, 2000).

According to Wang et al. (2002), to provide supportive environments for collaboration, design systems must provide participants with facilities for sharing information, coordinating tasks, and solving conflicts. The key challenge raised by the collaborative design of complex artifacts is that the design spaces are typically huge, and concurrent search by the many participants through the different design subspaces can be expensive and time-consuming because design interdependencies lead to conflicts – whenever the design solutions for different subspaces are not consistent with each other. Such conflicts severely impact the design utility and lead to the need for expensive and time-consuming design rework (Klein et al., 2006).

Slimani et al. (2006) state that the design process – or the definition of solutions starting from requirements and considering the use of constraint sets – can be represented by a multidimensional space of the design's parameters. Such parameters comprise the product model, the project goals, and the designers' social-cultural references (which define their work methods). In this context, to propose a solution and to publish it mean to unify the

proposed solution with others, i.e., to integrate it with the already proposed and validated solutions. However, this solution is only feasible if there is no interference between the proposed solution – and its parameter set – and the validated ones. Concerning interference, one must understand that it is an inconsistency in collaborative design, detected by the system. For example, interference can be a non-respected constraint or a negative evaluation given by some designer. In the latter case, the proposed solution simply blocks the concerned designer, not allowing him/her to carry on the work. Thus, a conflicting situation can show up right after a solution proposition, due to non-respected constraints or requirements, or viewpoint divergences. The conflict may also concern a specific design parameter or some particular design goal, as well.

Regarding the conflict resolution, we can say that the conflict resolution process is essentially a process of reduction of the cognitive gap between designers. Nonetheless, it is important to highlight that not every disagreement is a conflict. In systematic design[24], an iterative exploration method, like negotiation, is commonly used to find acceptable alternatives for everybody (Easterbrook et al., 1993). Moreover, requirements and their constraints also play an important role in producing new conflicting situations. Actually, if there aren't any constraints to respect, we can say that there will be no conflict at all, even if the designers will not agree upon the proposed solution.

Conflict management – including prevention and resolution – requires first the identification of the origin of the conflict causes, in order to propose the most suitable management approach. We present below a taxonomy of conflicts and their management approaches, based on previous propositions (Matta et al., 1996; Ferreira da Silva, 2003; Slimani et al., 2006).

4.2 Taxonomy of Conflicts

A number of computational models have been developed to classify and resolve conflicts in concurrent engineering design. A good example is the work of Matta et al., (1996), in which a typology of conflicts is proposed (Figure 4.1). This typology has a hierarchical structure, which reveals and highlights the causes of various conflicts and their distinctive behavior, and it is based on two categories: conflicts of strategies and conflicts of propositions. The first concerns the management of the design process, whereas the latter comes from the misunderstanding or the refusal of the other propositions. In the same way, Klein (2000) suggests a taxonomy, available in a conflicts repository, based on belief conflicts and on conceptualization conflicts. Some of these conflict types are studied again in the classification suggested below.

Another classification of conflicts was proposed by Barber et al. (2001). They see generically three subtypes of conflicts:

1. Goal Conflicts*:* Goal conflicts are conflicts involved with a goal's property, which may or may not be represented as ordering constraints or conditions.

[24] Systematic design refers to a process of design that looks not only at the problem that needs to be overcome, but also at the surrounding environment, and other systems that are linked to the problem. Trial and error, and technological evolution are methods used to arrive at a solution appropriate for the system (Systematic Design, 2009).

2. Plan Conflicts*:* Plan conflicts are conflicts in which certain preconditions of an agent's intended actions – of its temporary plan – become invalid due to the postconditions of another agent's actions.
3. Belief Conflicts*:* Beliefs conflicts are conflicts that involve inconsistent beliefs. Since beliefs include propositions about facts and evaluations, belief conflicts can be inconsistent descriptions about facts or incompatible evaluation statements.

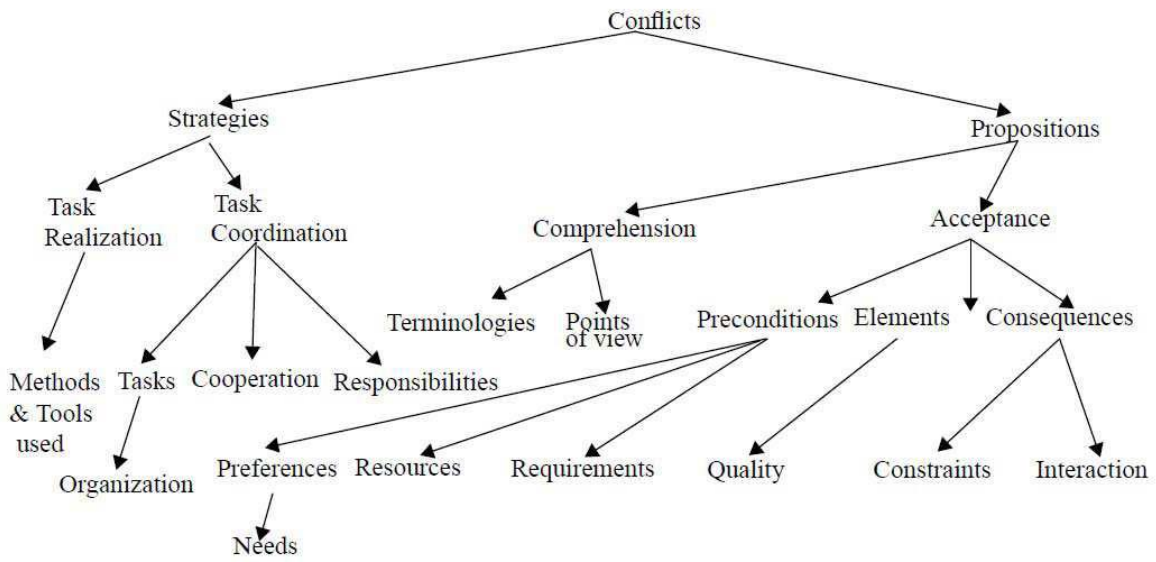

Figure 4.1 – A Typology of Conflicts (Matta et al., 1996)

Some other previous studies show that conflicts vary from each other in fundamental causes and behaviors, like in Klein (2000). He thinks that conflict resolution methods tend to be appropriate for certain kinds of conflicts. Thus, it is desirable to group the conflicts into different categories. The distinction between the eventual divergences of viewpoints over the product or over the design process does not take into account relational or psychosocial conflict causes. That is why this taxonomy makes a dichotomy between technical conflict causes and cognitive conflict causes. The former causes conflicts to which automatic processes can easily mitigate. The latter causes conflicts to which an automatic resolution is more difficult, less obvious, and which sometimes requires the intervention of human experts. The root of this taxonomy is the concept "incomplete specifications" (Ferreira da Silva, 2003). We show below how these two categories are organized.

4.2.1 Technical Conflict Causes

- **Linguistics:** In engineering design, agents – team members – use terms to represent the meaning of design entities such as variables and relations. Some conflicts emanate from the lack of comprehension between team members. They can be of two types:
    - **Terminological:** When design propositions use different terms to express the same concept. In this case, the design agents have different label-name relations or some variables in the design process.

- **Polysemic:** When a term has several meanings depending on the expertise domain where it is used. The main mitigation strategy here is the use of terminological resources (e.g. ontologies) to translate concepts used in several design domains.

- **Resource Sharing:** Designers are forced to use quantitatively limited resources, such as manufacturing facilities, materials, time and budget, etc. Considering each resource, even if, at the local level, one or several propositions do not exceed the imposed resource limit, at the global level, this constraint can be violated by the propositions set.
- **Requirements:** Inseparable from conflicts in engineering design, requirements may be grouped into three categories as follows:
  1. **Global Requirement:** A requirement imposed on a global variable whose value is determined by two or more independent variables from different modules or parts (Slimani et al., 2006).
  2. **Interface Compatibility:** Used to enforce the compatibility between module/component variables involved in an interface between modules.
  3. **Functionally Influenced Requirement:** This type of requirement is imposed on a module's variable, whose value is determined by another module variable's value, or several other module variable values.
- **Models and tools:** A multi-model approach is almost unavoidable in collaborative design. It enables designers to use tools for product instance representation (structure and behavioral models) adapted to the diversity of their viewpoints, tasks, and expertise domains.
- **Inadequate communication:** In this case, some agents fail to have sufficient communication necessary for coordinating their designs for a period of time, or keep on ignoring the other agent's requests and/or the design changes. Conflict resulting from inadequate communication has become an important topic in modern engineering projects with multiple companies or design groups involved, especially when they are located in different geographical locations.

4.2.2 Cognitive Conflict Causes

- **Conflicting Goals and Sub-goals:** In concurrent engineering design, each agent has his/her local goals. In some cases, these conflict with one another other, leading to a violation of the associated requirement. Human supervision of negotiation can help to solve these types of conflict.
- **Social-cultural Causes:** Design teams – distributed or not – often come from different engineering backgrounds (and thus have different cultures, knowledge, beliefs, working methods, experience, priorities, and criteria, etc.). Each team member has his/her own cognitive representation and this is composed of deeply ingrained assumptions, generalizations, or even pictures and images that influence how one understands the world and how one takes actions. Team members do not easily change their fundamental values and beliefs (Burgess et al., 1999). It is common that they may have different perceptions of the design entities, which may lead to conflict in design.

4.3 Conflict Mitigation Strategies

Because conflicts cause delay and consume resources, it is mandatory to enable prevention strategies that avoid their appearance. It is necessary to enable detection strategies as well, to identify, to categorize and to communicate them to the opposite parties, and to enable suitable resolution strategies, in order to return to a non-conflicting situation as soon as possible (Matta et al., 1996). These steps are called conflict mitigation strategies. It has been found that designer behavior varies widely from one conflict situation to another. A method to detect and resolve one conflict often does not work well for another type (Klein, 2000).

Several strategies have been proposed to solve conflicting operations. One of those strategies is to adopt a conflict prevention strategy based on pessimistic concurrency control mechanisms such as locking and turn-taking. In these systems, a user has to gain the ownership of the target objects before he/she can edit it, thus preventing other users from generating conflicting operations on the same object. Another alternative conflict resolution method is by means of serialization, which ensures that the effect of executing a group of concurrent operations be the same as if these operations were executed in the same order at all sites. If there is any conflict among concurrent operations, only the effect of the last operation (based on the total ordering) will be kept.

Figure 4.2 shows the decomposition of the generic *conflict management meta-process*, proposed by Klein (2000). He believes that the conflict taxonomy captures the range of possible conflicts and associated conflict handling processes, but does not specify which handlers should be used when for what exceptions.

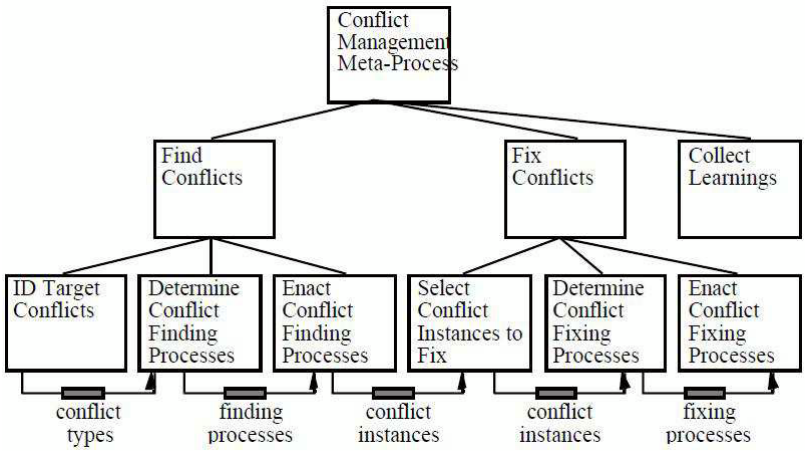

Figure 4.2 – Decomposition of the Conflict Management Meta-process (Klein, 2000)

Multi-Versioning (MV) is another technique for conflict resolution (Sun et al., 2002). The MV technique preserves all users' work by keeping multiple versions of shared artifacts. Such a strategy provides better feedback to the users, helps the users to better understand the nature of the conflicts, and to better adjust and coordinate their actions in the face of conflicts. However, MV is not suitable for design-oriented collaborative systems because keeping conflicting operations in different versions of shared objects does not help designers analyze conflicts as a whole or make use of conflicts to stimulate creative design.

Klein (2000) believes that conflict management is fundamentally a multi-disciplinary topic, and information in this area is scattered as a result across multiple disparate communities including computer science, industrial engineering, and management science, to mention just a few. Taking the conflict taxonomy as the starting point, some resolution methods can be applied to the different conflict causes. Conflicts are commonly grouped according to artificial intelligence and social science, such as fuzzy logic and linguistic knowledge (Fedrizzi et al., 1998). We present below the most common strategies used to deal with conflicts:

- **Terminological Resources (Ontologies and Thesauri):** These resources can mitigate linguistic conflicts. Ontologies can help to mitigate more complex conflicts, such as false beliefs, divergent cognitive representations, and incompatible socio-cultural customs. To do this, the required tools are translation for linguistic conflicts, conversion between models and languages, and matching between instances of the same model.
- **Prototyping:** Simulators and modelers enable the detection of conflicts due to inconsistencies between different product models. Simulators test the product behavior, while physical and virtual mock-ups help detect spatial interferences between their components.
- **Design Rationale:** Enables the enhancement of a model of the design decisions and the corresponding justifications when designers supply them. Inference engines and case-based reasoning can use them to suggest conflict solutions. Case-based reasoning is the process of solving new problems based on solutions for similar past problems. In this case, the most common past solutions are taken as the starting point to solve the new problem.
- **Rule-based Reasoning:** Rule-based reasoning takes predefined rules/statements as parameters to check the given model. It is quite similar to constraint checking (cf. further below), except that the rules defined by it are not design specifications, but instead they should be seen more like a to-do list, to be followed whenever a conflict appears.
- **Negotiation:** This strategy enables co-construction of a solution to a conflict by all team members. Each one can suggest a solution, supporting or rejecting the previous ones by offering a new proposition. The negotiation process involves direct interaction among designers, striving to find the best solution for everybody involved in the design project.
- **Supervision:** Negotiation time is expensive, so it must be controlled. It can be done automatically or one can rely on an authority. They supervise negotiation and decide on its solution and cost. Authority mission is also important to avoid deadlock situations. In all situations, a proposition that contains a solution must be adopted. To do so, several solutions are possible:
    - Choosing a proposition (by the authority);
    - Organizing a pool;
    - Automatically calculating the best solution, based on the different support and reject messages; or by
    - Establishing a priority viewpoint.

- **Viewpoint Priority:** This strategy applies to conflicts caused by resource competition. It consists of favoring priority expertise domains, whether they are consumers and/or their activities are more important to the global design process.
- **Constraint Checking:** Automatic task that needs a constraint model and targets to detect any conflict. Deciding the exact moment to trigger the checker that will verify consistency between different design propositions is not that easy. Constant alert can introduce system interference into design activity and lead to rejection by the designer. However, it is best to detect inconsistencies as early as possible; so one must find a balance. A good strategy may be triggering consistency checking after instantiating the product's structure model.
- **Constraint Relaxation:** Used for conflict resolution caused by shared resources. When some design parameters are flexibly defined, the system can suggest relaxing some values considered less important by designers. Parameters involved in a conflict are relaxed – that means they become less severe – for instance by widening the range allowed for its value attribution. A challenge consists in determining which constraints are to be relaxed to obtain optimal global design. This mechanism requires constraint propagation to update other parameters if those are interdependent.

Now let us look a little closer at each one of these approaches. Constraint strategies will be discussed in detail further down in Section 4.4.

4.3.1 Terminological Resources: Ontologies and Thesauri

These resources are used to prevent or to solve linguistic conflicts. While the thesaurus is suitable for dealing with synonymic or polysemic conflicts, the ontologies allow us to cope with more complex conflicts, through the use of precise definitions of the used terms, and by the establishment of relations between concepts and handled entities (Falquet et al., 2000). In this case, the conflicts are solved through the reasoning provided by the ontology information model (Chira et al., 2003; Qiu, 2005). There are two possible approaches for this method:

i. With previous definition. In this case, all the development process actors must work with ontologies and thesauri previously defined. The advantage of this solution is that it is easily implemented in a collaborative design system. As a drawback, the actors do not have flexibility, as they are obliged to use the given ontologies and thesauri. Besides, the use of predefined common concepts instead of the ones normally used by a designer – personal concepts that are usually refined and enriched during all his/her professional life – may contribute to reduce this method's effectiveness.

ii. Without previous definition. Here the designers are free to choose their tools, languages, models and terminologies. However, it is necessary to provide a system to guarantee the translations and exchanges of the users' personal terminological resources, in a transparent way. This approach is closer to the designers, because they keep using the terms that they are familiar with, but its implementation is much more complex.

4.3.2 Prototyping

Software applications for modeling and simulating are tools to detect conflicts generated by inconsistencies between functions, structures or behavior models (Sriram, 2002). Prototypes are also suitable to detect structural-level interferences, to test objects' predefined behaviors, as well as to detect possible conflicts (Toinard, 2001).

4.3.3 Design Rationale and Case-based Reasoning

Case-based reasoning (CBR) lets us reason from experiences and cases already encountered, in order to solve new problems: e.g. for maintenance of complex equipment, the collective memory of past incidents can be useful for taking a decision in case of a new breakdown. The retrieval of a similar past case can be used to suggest a solution to a new problem to be solved – this solution can be reused or adapted if need be. Improving representation of the cases, organization and indexing of the case base is important for enhancing efficiency of case retrieval.

A design rationale is a framework of the reasons behind decisions made when designing a system or artifact. Sycara (1994) had already suggested that CBR should be applied to conflict resolution, by adapting a previous conflict resolution to the current one. An understanding of the design rationale, or the justification for design decisions made throughout the design process, is necessary in order to understand, recreate, or modify a design (Hooey, 2006).

The work of Matta and Cointe (Matta et al., 1996; Cointe, 1998) is a good example in this area. A library of generic components is developed to store the conflict-related design information. Evaluation agents are designed to autonomously assess the design state information, looking for inconsistencies and identifying the conflict classes. The extension to the CommonKADS methodology suggested by Matta and dedicated to conflict mitigation in concurrent engineering includes mainly prevention, argumentation and negotiation methods. However, in the Matta and Cointe models, the clustering of information in the library is based on various design facets rather than different conflict behavior. In addition, the evaluation agent only uses the current value of design data, yet the history of these data is ignored.

Most of the systems are concerned with conflicts caused by spatial-temporal interferences. This happens when designers try, for instance, to place two elements of the product at the same place or to define two different behavioral states at the same time (Cutkosky et al., 1998; Cooper et al., 1998). SHARED-DRIMS (Sriram, 2002) uses design rationale to help to detect and solve these kinds of conflicts. However, mitigation of conflicts appearing in the earlier design stages is more difficult to solve and often disregarded. At this stage, product requirements and function models are less formalized. This leads to less obvious conflict detection and resolution, at those design stages. A few are concerned with resolution of social conflicts in the design process, such as NegotiationLens (Adelson, 1999), and their corresponding reasons.

Jiao et al. (2006) propose a knowledge-based system consisting of both machine-based and human designer agents based on exceptions. This work uses two approaches to deal with these exceptions. The first one is to classify exceptions into environmental exceptions,

knowledge exceptions, and social exceptions. The second one is to classify them into conversation exceptions, structural exceptions, logical exceptions, and language exceptions. They state that although different in criteria, exceptions in general can fall into two categories: those caused by agent conflicts, and those not. We here focus on the study of exceptions caused by agent conflicts with the belief that conflict management remains a complex task in MAS. Although being different in details, most existing exception handling approaches conform to the following paradigm: exception detection, exception diagnosis, and exception resolution. A key element underlying these approaches is the notion that generic and reusable exception handling expertise can be usefully separated from the knowledge used by agents to do their "normal" work. This principle is indeed developed from the conflict resolution rationale and the notion is also confirmed in the domain of collaborative design conflict management (Jiao et al., 2006).

The work of Ouertani et al. (2007) explores the linkages between the design process features and product specification dependencies, and suggests ways of identifying and managing specification dependencies to improve collaborative process performance. Using UML (Unified Modeling Language) specifications, it proposes a process traceability tool to track the design process in an ongoing manner. Based on the information captured, the dependencies between specifications involved in the tracked process are identified and inserted in a dependency network, which is maintained throughout the design process. A set of mechanisms is proposed to qualify the identified dependencies. It says that extracting and qualifying specification dependencies could be useful in many design situations; for example, during an engineering change management process to assess impacts and study change feasibility, or during a conflict management process to assist designers in resolving conflicts and maintaining the coherence of the design process.

Mohammed et al. (2008) say that CBR can be combined with rule-based reasoning (cf. next section) to lead to higher levels of automation. CBR uses past design knowledge to create new designs based on domain knowledge and newly refined inputs; thus, redesign is eliminated and time to market is shortened. CBR is also beneficial because performance of the case situation is known. A CBR algorithm uses a similarity algorithm to search for similar cases. Semantic similarity searches for similar names or part numbers. Function similarity may also be used. This similarity concept compares key dimensions. Parameter similarity uses input parameters from the user. Similarity searches may or may not use Boolean logic.

4.3.4 Rule-based Reasoning

Rule-based reasoning (RBR) intends to resolve conflicts by using a set of pre-defined rules. All constraints are transformed into primitive terms using rules (Yuan-Lung, 2007). Badr et al. (2002) invest in the development of control architecture for self adaptive software, combining conflict resolution and control strategies to resolve runtime conflicts. They built a service-based architecture prototype, implemented using Java[25] and Jini[26] technologies, to provide runtime monitoring and conflict resolution to support software self-adaptation. This approach is based on a control mechanism which monitors behaviors, detects and identifies conflicts, and formulates remedial action in the form of a resolution strategy. A service-

[25] http://java.sun.com/

[26] http://www.jini.org/wiki/Main_Page

oriented approach was adopted to develop the control mechanism and overall service-based architecture. The service-based architecture achieves self-adaptation by detecting, identifying, and resolving conflicts that occur at runtime. After detection, a conflict is identified and categorized according to its type before a resolution strategy is used to minimize the conflict. A monitor element then provides feedback to guide the conflict resolution tasks, which are used to implement the conflict resolution process.

According to Mohammed et al. (2008), rule-based and case-based reasoning also can lead to an object-oriented design environment. The initial computer aided design model or other geometric model is used as the class operator. Changes can be made to the class operator based on inputs and constraints applied by engineers. These changes create an instance (object) of the class operator. The instance will retain attributes associated with the class operator and will also contain its own attributes. An object-oriented environment is useful when only a small amount of changes need to be made to an already completed design. Repetitive tasks are eradicated from the design work, leaving time for idea development (Mohammed et al., 2008).

4.3.5 Negotiation

Negotiation is a process in which a joint decision is made by two or more parties. The parties first verbalize contradictory demands and then move towards an agreement through tradeoffs and/or searching for new alternatives (Jin et al., 2008). For collaborative design, negotiation can be a way for designers to exchange information, learn about others' perspectives and intents, and identify new opportunities based on the learned information and knowledge. Therefore, according to Jin et al. (2008), negotiation in collaborative design should not be merely a way for designers to reach agreements through simple give-and-take interactions. It should facilitate designers' exploration of a wider range of solution space through influencing each others' understanding of the problem, knowledge, perspective and judgments.

Besides computational models, the negotiation approach is a strategy widely recognized for conflict resolution in CE design (Matta et al., 1996; Cointe, 1998). When a conflicting situation is detected, the concerned system actors may choose to find a solution to it by negotiation. During the negotiation, each actor may suggest a solution that supports or rejects the previous one, through communication and publication of the new proposition. It is however advisable to determine in advance the implications of the new proposition, in order to avoid the arising of new conflicts.

Some researchers in the distributed artificial intelligence community have investigated the issue of negotiation by creating agent-based support systems that collect data from the participants and reconcile their disparities to achieve optimal decisions. Some have proposed negotiation processes that use case-based reasoning mechanisms together with restricted protocols to support agents resolving their goal conflicts. Jennings et al. (1998) proposed argumentation-based negotiation to support negotiation among distributed agents. Through argumentation, the parties can exchange various information pertaining to the negotiation situation, explore mutual option spaces and eventually arrive at an acceptable solution. Raiffa et al. (2002) proposed a taxonomy of group decision-making and suggested negotiation as a way to make joint decisions. Extending the multi-objective decision theory and game theory, he examined the dynamics of win-lose, win-win, and multi-party negotiations, and proposed novel approaches for successful negotiation.

Jiao et al. (2006) believe that while the conflicts between machine-based agents can be resolved by applying a repository of such meta-knowledge, the conflicts between human designer agents can be more effectively resolved by negotiations through some presumed method. Still, according to them, when the conflict happens between human design agents, the disputants can try to resolve it with four negotiation methods: inquiry, arbitration, persuasion, and accommodation. These four methods can be implemented along with a set of high-level protocols under the precondition of a global communication language, and knowledge-based agents.

Rose et al. (2007) depict a modeling framework to describe the work carried out by the different members of design teams, taking into account the knowledge exchanged to control the design environment. They analyze the collaboration exchanges' life cycle in design teams, and propose a collaboration ontology with different viewpoints. In that work, negotiation of the conflict via the resolution phases was first run by several iterations of explanation and argumentation, proposed to refine the conflict causes. When all the actors agreed and checked that the conflict causes were well identified, the designer proposed a first solution concerning the modifications to be made on the product. The mechanical calculation expert also proposed the solution comprising the launch of a much more important new study campaign to obtain a reliable and much more precise model. They also intended to help the actors in their interventions, by providing software with an inference function to allow them to consult the various exchanges that took place during the resolution of previous conflicts of the same type, through the use of an ontology. The provided arborescence permits one to quickly target the structured solutions or arguments/explanations provided with the ad hoc knowledge previously used in a similar case.

Jin et al. (2008) developed an Argumentative Negotiation framework for Engineering Design (ANED). ANED is composed of an argumentation model, a negotiation protocol, and a number of multilevel negotiation strategies, and it has been implemented as a computer tool to support engineering negotiation. In other words, they use an argumentation-based negotiation approach to support collaborative design. Their goal is to develop a negotiation framework that links designers and engineering systems together at the decision-level, facilitates understanding among them, and helps designers expand their search space and subsequently generate better alternatives.

4.3.6 Supervision

The conflict resolution by negotiation adds some cost to the development process and, because of that, it must be well planned and controlled. This control may be exerted in two different ways:

i) Automatically when it is needed, the main drawback is a possible indefinite duration; or

ii) Imposed by an "authority" (one of the actors would act as the project manager), and in this case, it is the authority that decides when to start and finish the negotiation process, and also chooses the suitable solution.

The authority role must also avoid deadlock situations in the system. He/she can impose one of the proposed solutions, organize a poll among everybody, define the best solutions based on some predefined criteria, or still, manage priorities.

Gzara Yesilbas et al. (2006) present a conflict resolution protocol that describes the decision-making process held to resolve the conflict after its automatic or manual detection. They use an arbitration process by guiding conflict resolution and using design history (through a collaborative model), in order to improve the resolution process effectiveness and preserve the conflict resolution memory.

4.3.7 Priority Management

Priority management is used when there is competition and resource-based conflicts. It must bias toward propositions, according to the actors who formulated them, their domains, their previous work, or by priority order. For example, high priorities may be attributed to transactions that need a lot of resources, or to the ones considered to be more important than others into the global design process. In this case, weights are attributed to transactions and activities. This attribution may be done by the project manager in the beginning of the project, during the constraint definition phase. It should be considered also that these priorities may be changed along the design process.

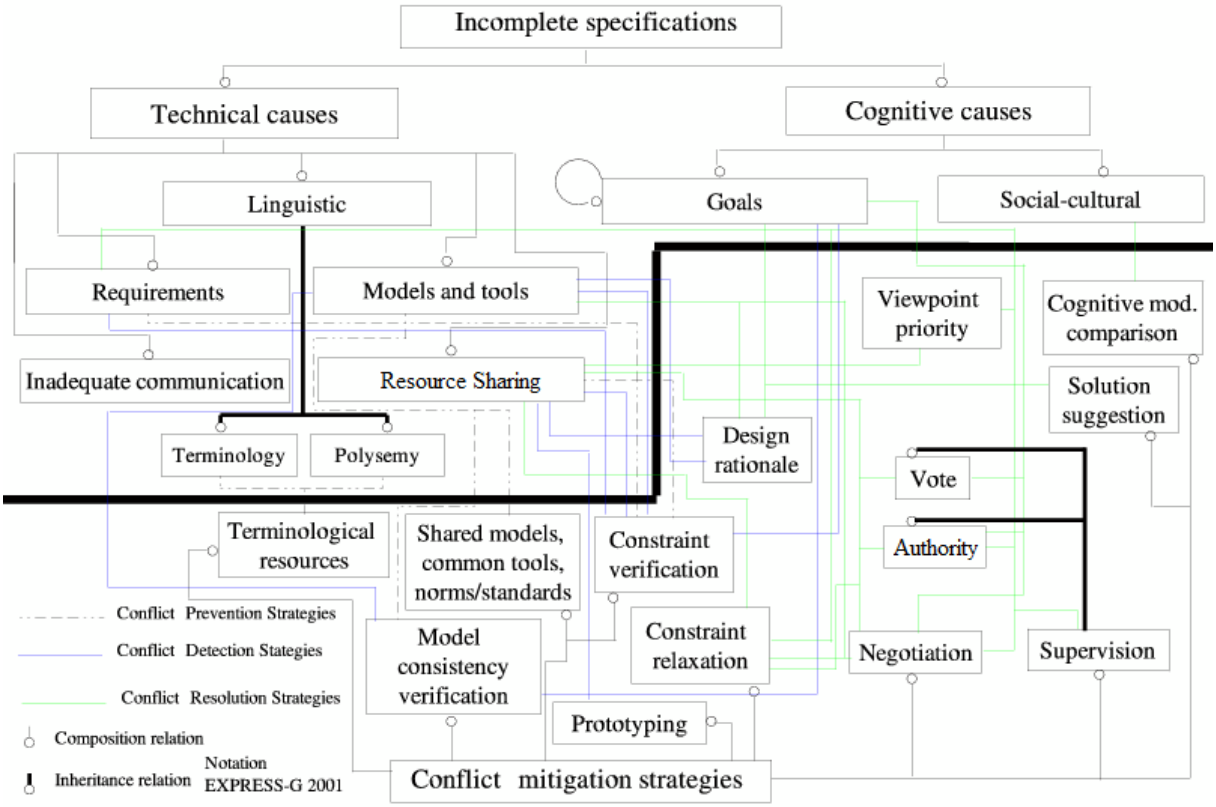

Figure 4.3 – A Conflict Taxonomy (Slimani, 2006)

Figure 4.3 shows the conflict classification used by Slimani (2006), which has been taken as basis for this work. The upper part represents the taxonomy, and the lower one depicts the

resolution strategies. As we can see, Priority Management is considered in the classification of Slimani (2006).

Another strategy to mitigate conflicts is the use of constraints. The next section explores this topic more widely.

4.4 Constraint Management

The tackling of engineering problems, more and more complex, along with tightened constraints (cost, delay, etc.), demand the design and development teams to be even more multidisciplinary and large enough to permit different design tasks to be done in parallel. Such requirements exploit the parallelism that is inherent in large projects. Unfortunately in collaborative design, to integrate different parts, specified and developed in a parallel way, is a huge task to cope with. This approach frequently leads to conflicts when the global specifications are not well observed.

This happens because each group of actors has the tendency to observe only a subset of constraints, the one related to their own domain. It is only when all the partial work is integrated that the whole set of constraints is taken into account. Instead, if the constraints were considered at the moment the tasks are done, then the system rollback – and its cost – would be considerably reduced, or even avoided (Sriram, 2002). It is then necessary to implement a method of constraint management. Considering a distributed and multidisciplinary context, inside a collaborative environment, it isn't a simple task to work with constraints (Sriram, 2002).

Constraints provide boundaries for representing solution spaces. Constraint satisfaction techniques are used to compute approximations of solution spaces of constraint satisfaction problems (CSPs). A CSP is defined by a set of variables, their domains (a priori possible values for each variable) and a set of constraints. In engineering, constraints are typically numerical relationships – equalities and inequalities – that influence feasible values of continuous and discrete variables (Lottaz et al., 2000).

4.4.1 Definitions

The methodology for constraint management used in this work was previously defined by Slimani (2006), which is based on concepts formalized by Brockman et al. (1996). In the following sections we are going to provide an overview on how this methodology is organized, and how constraints are generated and evaluated through it.

The design history, noted H_n, is defined by: a) a series of state-transition pairs noted $\{<e_i,t_i>, i = 1, \ldots, n-1\}$, where e_i represents the state of the design process at stage i, and t_i is the transition between e_i and $e_{i+=1}$; and b) the current state of the design process e_n.

An e_i state of the design process comprises three elements:

- Hierarchy of handled objects: Represents all the objects used in the design. They are organized by levels and are linked by composition or interfaced relations. Each object

is described by a set of variables, called properties, each one of them representing a specific object's characteristic. Taking some action implies the attribution of at least one value to one property. A property's value set represents all the possible alternatives that the designer can attribute to this property. These values can vary from different types, such as: text, number, complex object, and object aggregation, among others.

- Hierarchy of problems: Represents the set of formulated problems until this state. A design problem is described in terms of goals, objects to be designed, constraints and input data, and properties expected as output. Each problem has a state (a status: "solved", "waiting for solution", "being solved"…).
- Constraint network: A set $C_i = \{c_j, j = 1, \ldots, N_i^C\}$, where each c_j represents a constraint, and N_i^C the number of constraints in the state e_i. Constraints describe the relations that validate the involved properties, in order to achieve flawless design. Each constraint c_j is given by:
    - 1) the set $A_j = \{ a_k, k = 1, \ldots, N_j^A\}$ of the N_j^A properties affected by c_j;
    - 2) a relation ρ_j that links these properties; and
    - 3) a status indicating if the relation ρ_j is satisfied or not with the values given by the N_j^A properties.

The solution to a constraint network is the instantiation of the variables of the network constraints so that each constraint is satisfied. A transition t_i describes a state changing (from e_i to e_{i+1}). Each transition t_i is the result of a design stage E_i. A design stage is an action of problem resolution taken by the designer.

4.4.2 Constraint Checking

A constraint network is formally defined by $P = (X,D,C)$ where $X = \{x_1,\ldots,x_n\}$ is the variable set, $D = \{D(x_1),\ldots,D(x_n)\}$ the set of current domains (being $D(x_n)$ the finite set of possible values of x_i), and C the variables' constraint set. $D_0 = \{D_0(x_1),\ldots,D_0(x_n)\}$ represents the initial domains of P.

A constraint C upon the variable set $X(C) = \{x_{i1},\ldots,x_{in}\}$ is a subset of the Cartesian product $D_0(x_{11}) \times \ldots \times D_0(x_{1n})$, which specifies possible combinations of the variables $x_{11,\ldots,}\ x_{1n.}$ Each element of $D_0(x_{11}) \times \ldots \times D_0(x_{1n})$ is called a tuple of $X(C)$ and $|X(C)|$ is the number of arguments of C. Consequently, to check a constraint means to verify if a given tuple τ is allowed or not.

We consider that constraint checking is an automatic task, based on the specification and modeling of constraints. It is not always easy to decide when exactly to trigger the constraint checking mechanism. To check every proposition may slow down the development process (Slimani, 2006). Besides, this method isn't too well adapted to collaborative and distributed work, where designers spend some time working locally on their parts. It may sometimes generate "ghost" or "ephemeris" conflicts, which disappear in the sequence of the design. Thus, it is interesting to detect inconsistencies as soon as possible, to avoid propagations that may lead to a "snow ball" (Slimani, 2006). Also, verifications too spaced out from one another chronologically, may be lenient with one or more constraint violation, delaying the

whole conflict detection process. To solve this situation it will be necessary to undertake some rollback, in order to achieve a consistent state of the system, the one where all the constraints were consistent. Of course, this will add some additional cost in terms of development. It is important, then, to find a balance between these two approaches.

Hu et al. (2007) present an approach based on the constraints network to support collaborative design. They propose an algorithm to predict conflicts and refine the intervals for parameter coordination, which verifies the design stage early in the process and assists the designers in determining design variables to reduce the multidisciplinary iterations in collaborative design. They also establish a model for parameter optimization based on a constraint network.

4.4.3 Constraint Relaxation

Constraint relaxation is used to solve conflicts that concern resource sharing. Usually these conflicts arise when design parameters are defined with some degree of flexibility, and their values, at a given moment, do not respect the predefined constraints (Jussien et al., 1996). In this case, the system may suggest the application of constraint relaxation on these parameters. So, to solve a conflict caused by the violation of one or more constraints, the relaxation process may be done, e.g., to widen the set of acceptable values (Slimani et al., 2006). A great challenge here is to determine, inside this context, which constraints to relax, and to what degree to relax them, in order to obtain the optimal design.

Figure 4.4 shows the Constraint Management Module proposed by Slimani (2006):

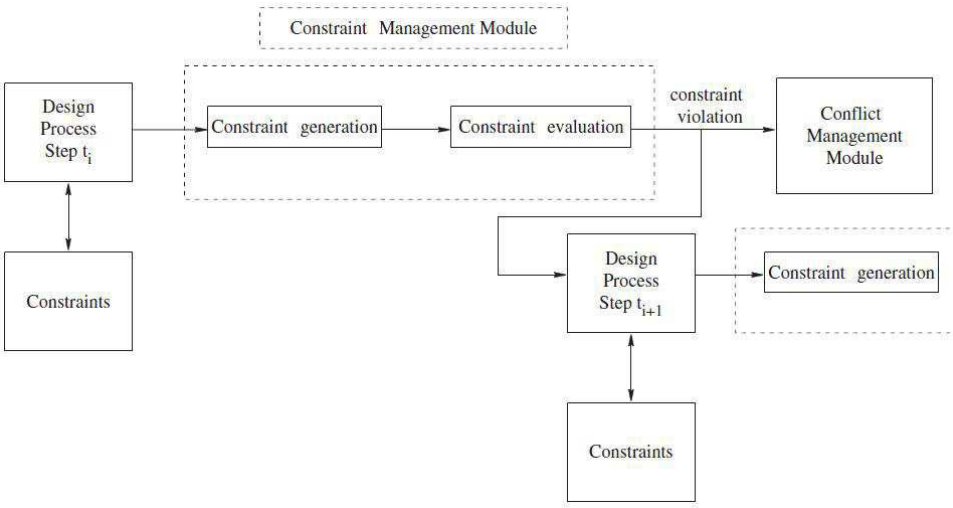

Figure 4.4 – Constraint Management Module (Slimani, 2006)

4.5 Summary

We have presented in this chapter a state-of-the-art on conflicts and conflict management in the context of collaborative design. Aiming to establish an effective conflict management strategy, we have analyzed several methods and projects (old and ongoing ones). We can say,

in general, that a good approach for conflict management consists of prevention – when possible, early detection, and resolution.

The majority of the works analyzed here point in the same direction, i.e., the combination of one or more techniques presented in this survey. The same occurs with the mitigation strategies, when considered that there are conflicts that can be easily categorized into more than one taxonomy item. As Barber et al. (2001) put well, classifying conflicts should not mean creating isolated classes, but instead, as they are closely related, transforming them into one another. Hence, we can imagine a scenario for conflict management composed of prevention techniques, and strategies for automated conflict resolution and for cognitive conflicts, putting some special emphasis on negotiation.

Also, the use of constraints is a very powerful approach to attenuate conflicting situations, as it lets us check a set of pre-defined values/rules in an automated manner, saving time for designers. Besides that, we can foresee the growing importance of lock mechanisms. In their recent survey, Shen et al. (2008) state that concurrency control is the foremost mechanism for organizing synchronous activities to avoid conflicts. Locking is a primary means in managing concurrency control to prevent people from colliding, and three types of locking – non-optimistic, optimistic and visual – have been developed and used in various applications. Negotiation can formalize and implement the mediation and facilitation functions among people to handle conflicts.

Moreover, if we recall Easterbrook et al. (1993), who said that human design agents prefer to communicate in natural language, rather than in the formal language, which is often considered hard to understand and thus avoided, we may conclude that the best approach to follow is a combination of several, which must be adaptable and suitable to each particular situation. In Chapter 7, we will present the conflict management method used in this work that focuses on the use of ontology-based representation.

Chapter 5

5 A Proposed Framework for Collaborative Design

Collaborative design refers to the coming together of various people to achieve a common purpose that is developing a product through interactions, information, and knowledge sharing, with a certain level of coordination of the various implemented activities. In that case, various resources set up around the design actors describe a real network around the design project. Unlike the traditional organizations where participants mainly share the results of their work, in collaborative work people share and build knowledge in order to create a common understanding of the situation.

In order to support interoperability among different engineering computer tools and coordinate the design activities of multidisciplinary engineering projects, it becomes a monumental task to develop an efficient collaborative design environment. Not only should this environment automate data exchange and the design process, but also be easy-to-use and widely accessible. Many organizations are dedicated to investigate frameworks for supporting optimized applications for multidisciplinary design. Besides, a number of projects have been proposed in this area in recent years. However, we consider that they fall short of providing an effective solution to effective knowledge sharing and collaboration on design projects, because they fail to combine the geographical, temporal, and functional design aspects with a flexible and heterogeneous knowledge representation model.

To overcome the weakness of the methods described in Chapters 2 and 3, we propose a framework for collaborative design that intends to be synchronous, generic, service oriented, agent based, and ontology based. It also focuses on some integration aspects like flexibility, portability, and reusability. This framework integrates several existing widespread technologies such as agents, Web services, and ontologies. It is primarily designed to allow the collaboration of different areas of expertise during a product design project.

5.1 Global Architecture

Figure 5.1 shows the system's global architecture.

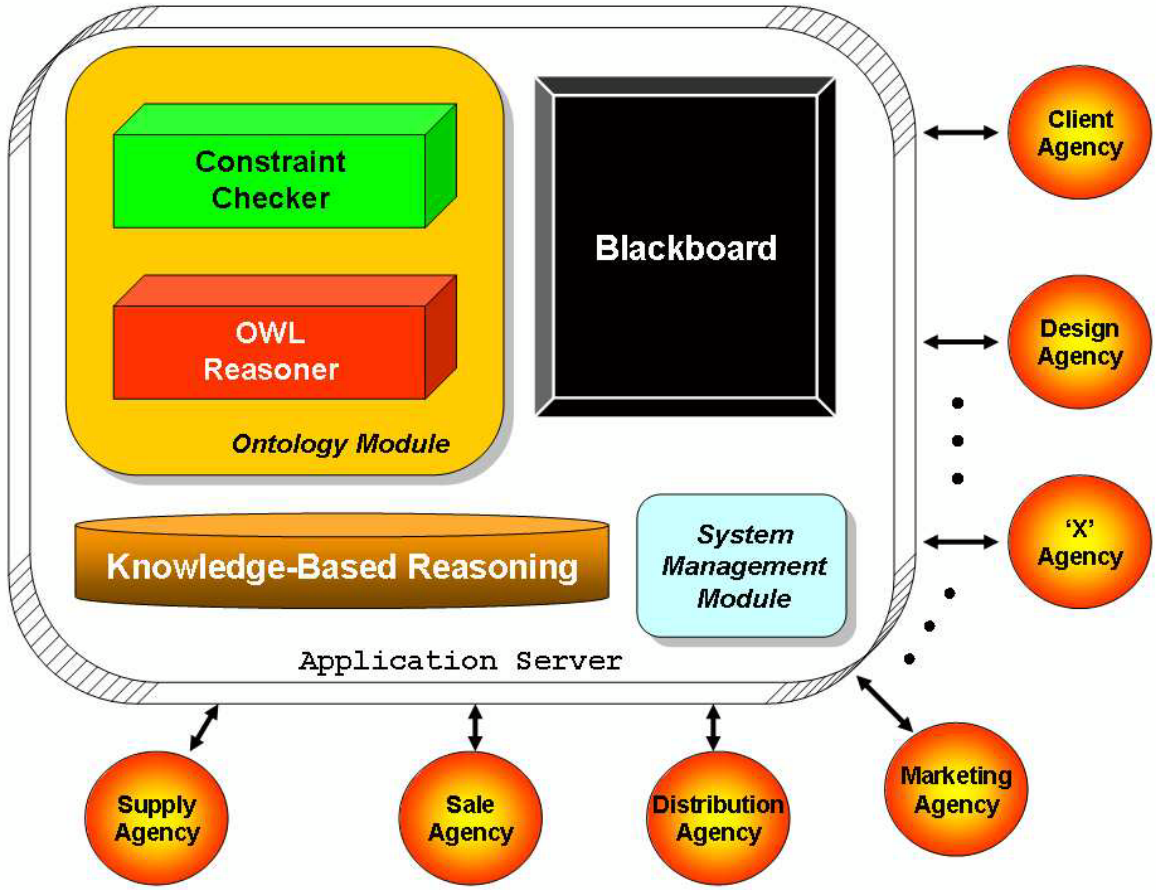

Figure 5.1 – Global Architecture

The different actors and knowledge areas of the collaborative design process must be represented within a collaborative modeling system. We consider the modeling activity as a multidimensional-space problem, in which dimensions represent activity aspects to be considered, such as different knowledge areas, different participants and their viewpoints, different stages of global activity, and different requirements. In Figure 5.2, we can observe the participants' different viewpoints, regarding different areas.

Clients express their requirements in a global and *ad hoc* way. These requirements are distributed among different knowledge areas and enable the definition of the specification model. During the modeling activity, each expert takes into account the client requirements in relation to his/her own area of expertise. Our global architecture defines a space that enables the design participants to extract knowledge concerning their personal areas of expertise, in order to help them construct their models. We call this space Knowledge-Based Reasoning (KBR).

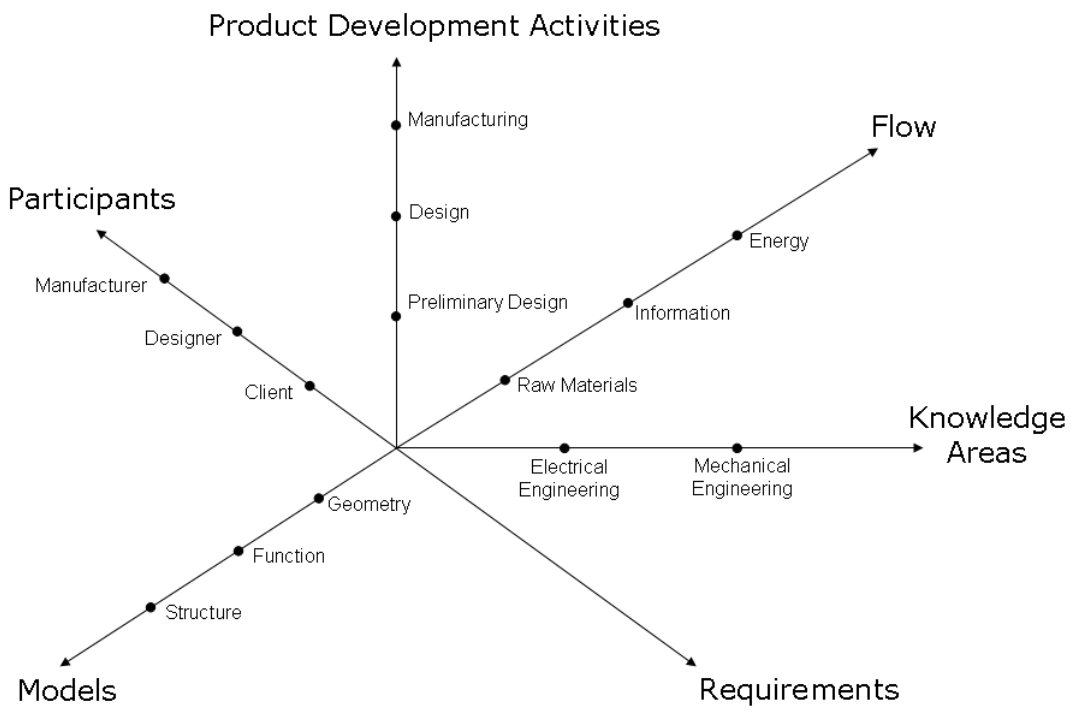

Figure 5.2 – Problem Space

5.1.1 Project

Organizations that are involved in the design and development of new products should adopt flexible methods of working to meet the numerous and varied demands of the global marketplace. Connected through information and communication technology (ICT), organizations are able to collaboratively design new products. The product development process covers several issues, including idea generation, concept development and testing, marketing strategy, business analysis, product development, test marketing, and commercialization. To facilitate the collaborative design, a computer-networked environment based on human aspect should be built so that members of the team are able to follow their experience to work together with other participants in geographically different locations without any difficulty.

In our architecture we have proposed a shared workspace for the project (Figure 5.3). This space is intended to provide the interaction between the different agencies involved in the project. Is within this space that the final product design is stored.

We think that various support services should be provided in shared workspaces. The knowledge representation should be mapped onto a distributed database environment which supports persistency and concurrent access in a distributed shared workspace. Product information must be structured to support various design activities, particularly the collaborative nature of design. These requirements are the following:

- Support for multiple levels of geometric representation.

- Support for multiple functional views.
- Support for representation and management of multiple levels of constraints.

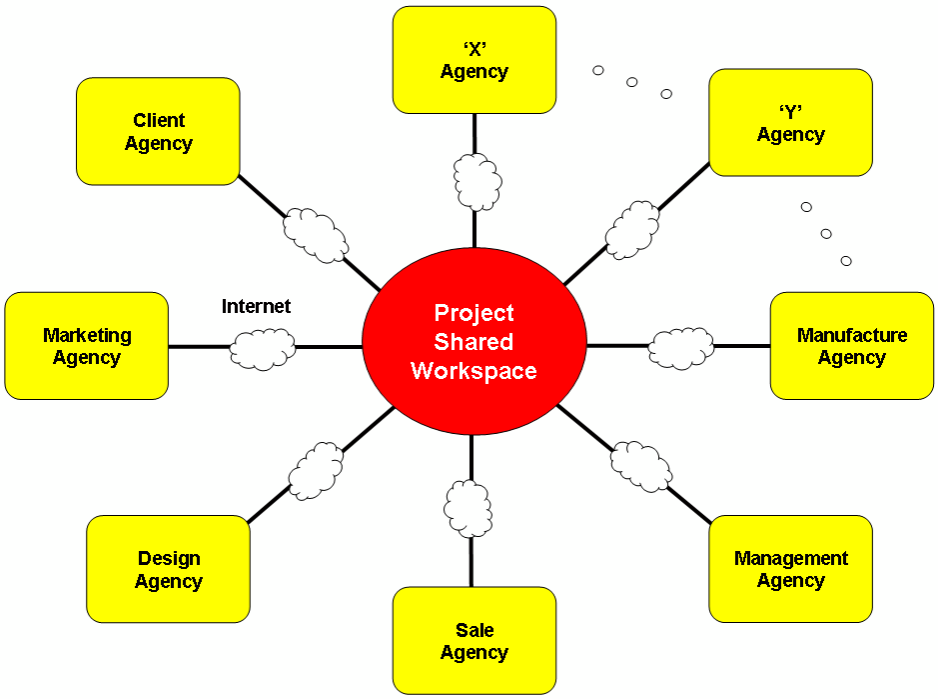

Figure 5.3 – Project Shared Workspace

We propose that each agency also has its own collaborative space, structured in a very similar way to the project's, as we are going to see below.

5.1.2 Agencies and Agents

An agency is a Web- and agent-based system that represents particular activities of a given domain/discipline. The agency definition lets us reproduce the global view of an activity, whereas the agents inside an agency represent the expertise to be considered for each participant. For example, within the client agency, only requirements are represented and handled. Within the design agency, we represent not only client requirements and the knowledge needed to distribute them among the existing knowledge areas, but also the knowledge needed to elaborate the function model, through the definition of different parts – related to each knowledge area – of the function model (Figure 5.4).

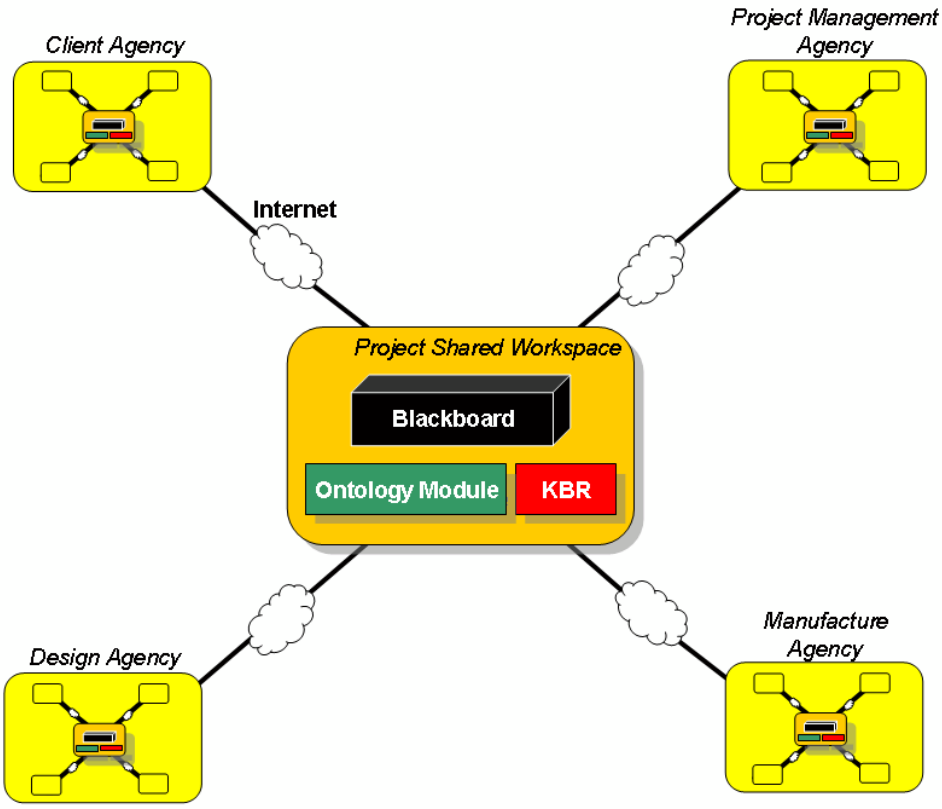

Figure 5.4 – Multiagent Architecture

If we zoom in on the client agency, we can see (Figure 5.5) that there are as many agents as there are knowledge areas. Each agent, which represents a design area, has a knowledge base that lets it extract requirements concerning its knowledge area. The requirements that are not related to any specific knowledge area are kept in a general blackboard, to be used later during the modeling process.

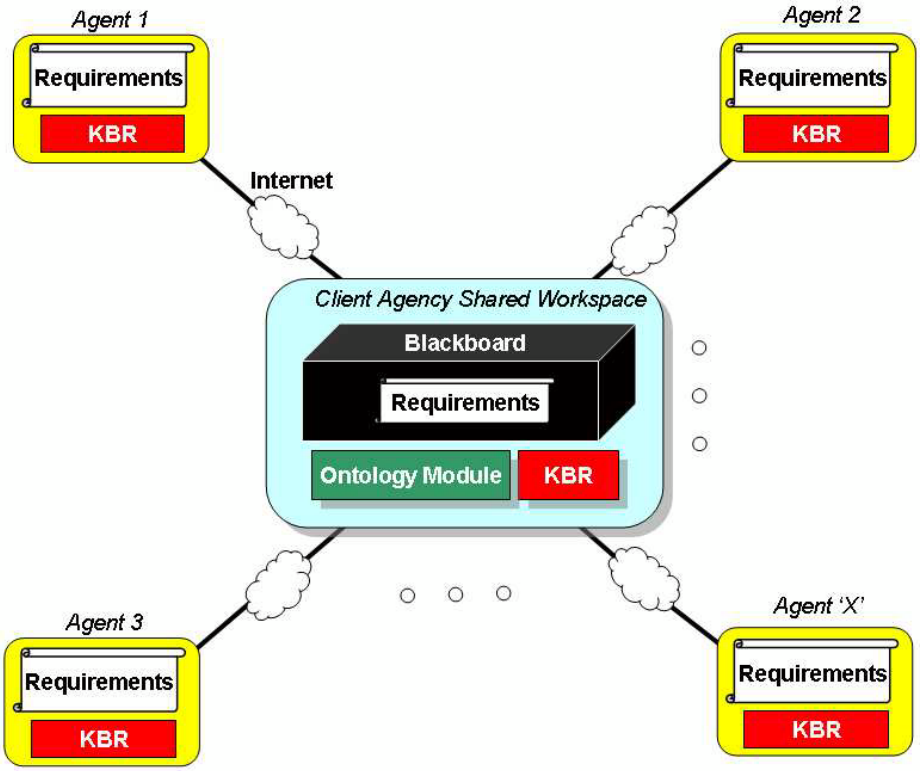

Figure 5.5 – Client Agency

The other agencies have the same structure. As the next figure shows, the design agency is composed of two parts (Figure 5.6). The first one expresses requirements and has the same structure as the client agency. The second part elaborates the function model, and comprises as many agents as existing design areas. We have chosen to represent three agents, which correspond to three design areas: civil engineering, hydraulic engineering, and thermal engineering.

Each agent has a knowledge base that lets it elaborate the function model. To create this model, the agents use the specification model and a function tree related to three flows: energy, information and raw material flow. The specification model is then handled and may evolve during the modeling acitivity of the product. The function model is elaborated progressivily and is accessible to all the participants.

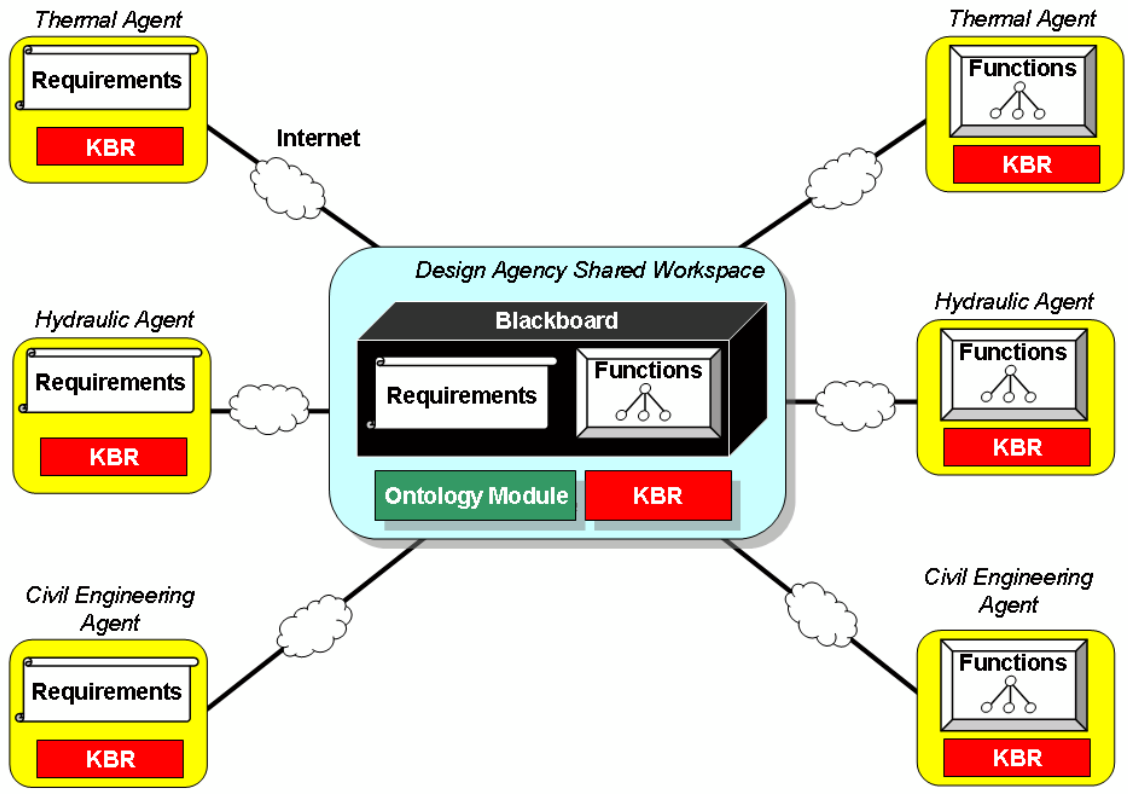

Figure 5.6 – Design Agency

An agency also provides services to support the collaborative design process, such as design process management and control, coordination, and cooperation. The design process for the multidisciplinary engineering problem community is typically iterative and parallel. Multiple design activities in a design process may sometimes need to execute in parallel while not affecting the correctness of results.

Within our framework, a design process is seen as a collection of many design activities that are logically related in terms of their contribution to the solution to the multidisciplinary design problem. Each activity in a design process has special properties that are either basic or extensive. Basic properties are composed of participants, activity variables, execution conditions, etc. Extensive properties include adapter and application identifiers, input and output design variables, the relation among design variables, as well as the relation between activity variables and design variables.

The agency definition provides us with an overall view of a specific domain, whereas the agents represent the activities to be considered in the domain. They can also be seen as particular functions within the system, e.g. the version management. Every tool or computational resource may be encapsulated by an agent. The agent category represents the human experts of the collaborative project. Each agent uses differents types of knowledge: the generic or domain knowledge – related to its specific design area – and the specific or experience knowledge, which represents each area of expertise.

During a collaborative work, the designer's tasks are performed in parallel and their results should converge to satisfy the design objectives. These objectives could be refined during the progress of the design project, thus understanding the collaborative design process to control

it is important. The efficiency of each collaborative work depends on the actor's capabilities to collaborate. It is the quality of the exchanged knowledge between design actors that will influence the evolution of the design environment. The results of the collaborative design activity directly depend on relationships between designers. It also depends on the methodology used to represent and to integrate the agents' personal models.

A major challenge is the integrated environment for coordinating and managing the worldwide-distributed design teams. It is necessary to develop an integrated environment while considering human requirements and the efficient tools for collaborative design. The literature review (Chapter 2) indicated that more effort is needed for enhancing such an integrated environment for collaborative design. Other limitations of the existing systems are that most of them deal with technical restrictions for collaborative design rather than considering human experience and human requirements during collaborative design. It is also apparent that not all aspects of the product life cycle such as the marketing analysis stage and human resources management were considered in these systems.

At the implementation level, each agency represents a specialty, a knowledge domain. An agency aggregates one or more agents. Each agent has a private workspace of data and information. In these private workspaces, the agents work on their propositions before submitting (publishing) them to other agents.

From an overall viewpoint, the architecture proposed here can be seen as a network of computational resources and human experts. It is organized as a set of autonomous agents (humans or software applications) that represent the several actors involved in a collaborative design process; and that integrate different tools (database, CAD and communication tools, etc.) within an open and generic environment. In the next section, we are going to detail the two collaboration levels that exist in our architecture.

5.1.3 Collaboration Levels

The proposed architecture comprises the following essential elements:

- The agents, which represent the actors involved in a design or development process. These agents are autonomous with regard to the others and undertake their respective tasks according to their own viewpoints. However, throughout the collaborative design process they will be compelled to exchange information. From the agents' perspective, the system is defined as a workstation set, geographically distributed across a network.
- The agencies, which are multiagent systems in which the agents represent activities related to a particular design area.
- A set of graphical interfaces to provide the agents with invocation of the different system modules. These interfaces are also used by the agents to initiate a design process, as well as to exchange data and information, and to negotiate solutions to conflicts, among others.
- The collaborative system and its internal components, which provide interaction among different agents in order to achieve a product design. This system is the core of the architecture, and is composed of a number of modules (depicted below). Each module is application domain independent, so that the architecture's structure remains

flexible and portable. For all intents and purposes, it is an open architecture, as agents can be taken out or new ones can be introduced without the need to modify the architecture's structure or restart the whole system. Modules also can be added or removed without affecting the system execution.

Regarding the architecture's data structure shown in Figure 5.1, three sub-levels can be considered:

1. The first one is the local level, which is associated with each expert. There is a local knowledge base, used by the agent to stock his/her own proposed models and expertise. These models and expertise will be exchanged and updated during the design process, to aggregate information/suggestions given by other agents. They can also be used later by the agent to work on another design project.
2. The second level is the agency level. Grouped by expertise and interests, these clusters provide the designers with a collaborative structure, where they can exchange their private models previously designed and stored in their local databases. It is composed by a blackboard accessed by everybody through special implemented services. The blackboard's structure will be depicted below.
3. The third level is the project level. This level comprises also a blackboard. It provides experience and information sharing among the agencies – in order to form a knowledge base – as well as coordination and synchronization of activities, aiming to achieve a common goal, the final product design.

Whereas in the project shared workspace, different knowledge areas collaborate, in the agency shared workspace, the agents themselves interact with one another (Figure 5.7).

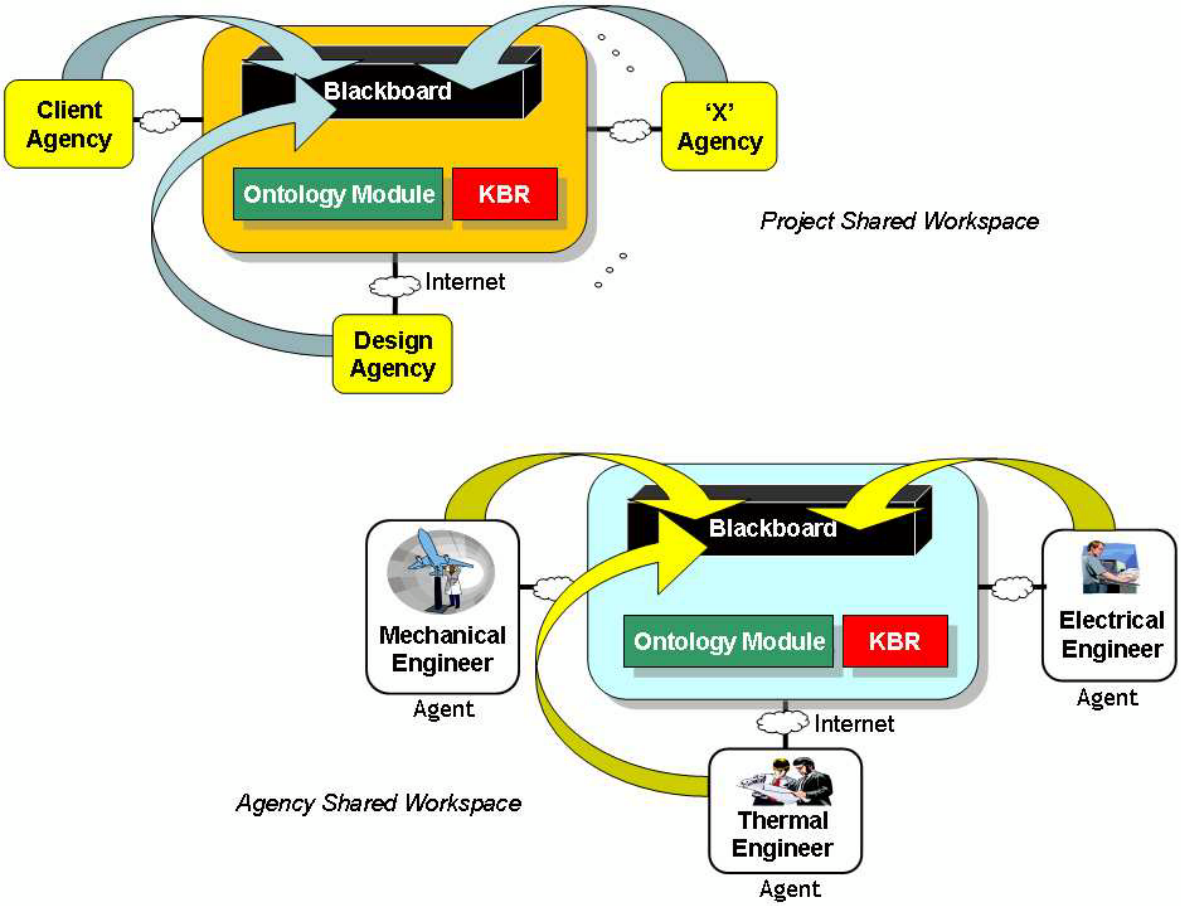

Figure 5.7 – Shared Workspaces

The several agencies involved in the design project communicate and collaborate through a Web application server, composed of three modules: a blackboard, an ontology module, and a management module (Section 5.3.1). The constant feedback between designers enhances the collaboration process, as it permits full-time interaction among them. Such architecture provides the detection of problems and potential conflicting situations even in early-stage design. Once the detailed design stage ends, it can be considered that there are no more adjustments to be made to the common model, and it is ready to be sent to the manufacturer to be produced.

Having both types of workspace – private and shared, this architecture enables different work modes, alternating from private/isolated sessions to collective ones. Inside an agency, as within the system's global level, engineering tools and development participant actors are connected through a network and communicate through a blackboard (as shown in Figure 5.7). Thus, cooperation among engineering tools and system actors is undertaken in the same way as in the global workspace. Agents can interact with one another through the agency shared workspace, as it has the same structures present in the agent private workspace – but in a higher level – because all data and information contained therein were previously agreed upon all concerning agents. In the same way, inter-agency communication is provided by the project shared workspace – a global workspace to store the final design models. This framework is thus a multi-layer collaborative architecture.

Inside the agencies, designers collaborate to achieve a common design model; this is called the *intra-agency* collaboration level. In the higher level, agencies communicate among themselves through an application server, the *inter-agency* collaboration level (Figure 5.8).

Ontologies are used to represent the designers' models in our methodology (Dutra et al., 2007b). The intra- and inter-agency collaboration provides the refinement of such models in order to merge them together in a final design solution. The goal is to have the final model tailored in a seamless way, i.e., to be consistent, free of conflicts and acceptable to all participants (Dutra et al., 2008). Considering the different expertise involved in the design, we can ensure that there will usually be heterogeneous design propositions/models.

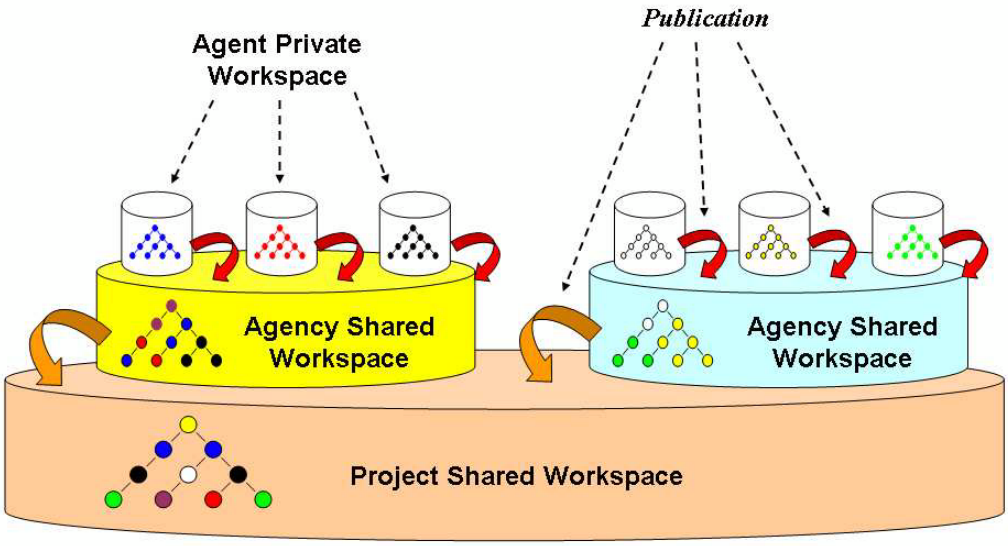

Figure 5.8 – Collaboration Levels

5.1.4 Publication

Ontology publication is an essential point to be considered in our architecture (Dutra et al., 2008). Publishing a proposition means merging different proposed instances of a product into the design space. This merging is only possible if there is no interference between the elements, i.e., if they are all consistent.

To meet specific requirements in the CAD community, a few real-time collaborative CAD systems have been developed, which allow designers to manipulate the same design documents at the same time. We think that to achieve high responsiveness in a networking environment, shared documents are usually replicated at each collaborating site so that editing operations can be performed at local sites immediately and then propagated to remote sites.

During a publication, constraint checking and an ontology reasoning process are done in order to guarantee the consistency of the published ontology instance (Figure 5.9), as described in Dutra et al. (2007b). It is during the publication phase that a conflicting situation might pop up.

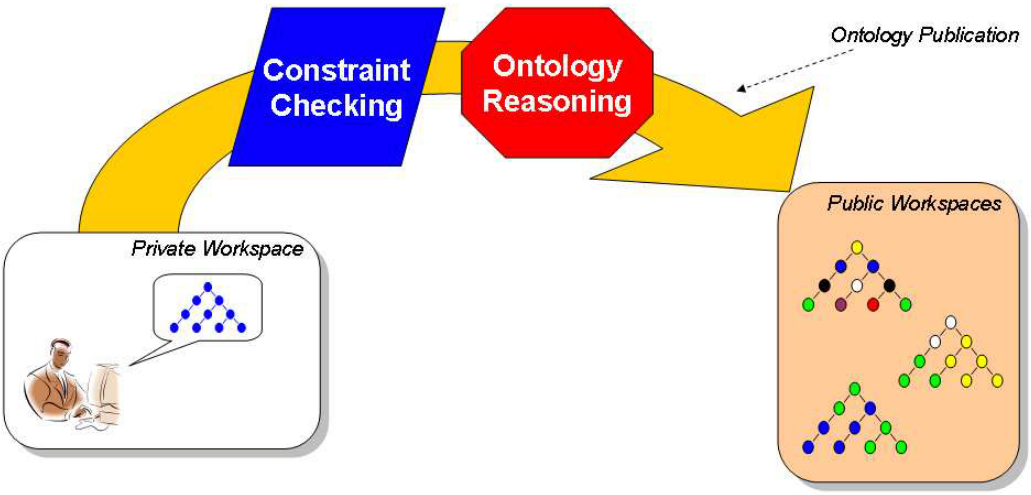

Figure 5.9 – Ontology Instance Publication

In the following sections, we introduce the main characteristics of the proposed architecture. Subsequently, we present the system modules and actors, and how communication and collaboration are done within this architecture.

5.2 Main Characteristics

Our framework embodies knowledge from human experts and computationally encodes them as ontology instances, aiming to represent the collaborative knowledge. While expert systems are typically powerful tools in static environments, they often fail when used under varying external constraints. Collaborative knowledge is not only purely technical. It actually does not concern only the product to be designed, but also the activities and the design process. We define four prerequisite components to characterize collaborative knowledge:

- The first component is similar to a lexicon, and its incoherence can result from the problems of synonymy or polysemy. Collaborative knowledge is a common language for various professions represented in the project. This allows a global and rapid understanding of each problematic point.
- The second component has the mission to collect the bricks of basic knowledge inherent to the other disciplines involved in the project.
- The third component relies on common experiences of actors having already worked together. This knowledge must include various past allegories or first-used metaphors, in order to be reused in the new collaboration context.
- The fourth component is originated from the capitalization of the appropriate information and traceability for the current project.

We define the general knowledge of an expert as being the meeting of the expert's in-depth knowledge and the collaborative knowledge handled by this expert. The in-depth knowledge of an expert regroups all the expertise possessed by the expert in his/her specialty domains. This knowledge is mainly composed of tacit elements and experiences; directly linked to the

expert to whom it belongs. We thus think that collaborative knowledge is a set of knowledge that can be applied to a particular collaborative design situation. The collaborative knowledge can be seen as being the support of a partial and superficial exchange of knowledge among various experts and software tools involved in a project. This exchange authorizes the collaboration among these various participants coming from different professional horizons, each having a different past, by sharing models or common references for perceiving a global vision of the problem.

In the proposed framework, we realize the design not only as an integration of multidisciplinary design tools and experts, but also as a complex automatic design process which can significantly reduce the overall time for solving a multidisciplinary optimization problem. It is a Web-based multi-agent system for collaborative design. We think that combining both technologies is the best approach to follow. The system comprises autonomous agents, which represent the several actors of a development process, and integrate different tools (database, CAD and communication tools, among others) in an open environment. The agents provide us with the autonomous, reactive and "intelligent" aspect of the model, whereas the Web-based approach provides us with large-scale collaboration, as we can use the Internet infrastructure to connect all the involved actors. From a global perspective, the architecture proposed here can be seen as a computational and human resource network.

As it has been shown on the state-of-the-art survey, neither the agent- nor the Web-based approaches alone could provide us with a satisfactory environment for collaborative design. Consequently, the use of both has enhanced our proposal, giving it more robustness. The proposed architecture is domain independent because of its independent module-based structure, which can be adapted to different domains. All the modules are completely independent from domain and from one another. An ontology-based approach has been implemented to be used within this architecture. This approach uses ontology-based representation and an inference engine to merge the designers' propositions into a final ontology instance: the final product design (Dutra et al., 2008). The main goal of this ontology-based approach is to deal with conflicts in collaborative design.

We have applied the multi-model approach to this architecture. We consider it to be very useful, as its main idea is to gather all the people involved in the design of a product, i.e. clients, engineers, manufacturers, suppliers, vendors, distribution people, assembly departments, and marketing people, among others. Such a platform provides a collaborative environment where all expertise is present and is taken into account to design a product. We consider that joining together all the concerned people is a good approach to diminish design problems, since in this way we can gather and correlate all the information about the interconnections among the engineering data and the interactions between the participants. In the traditional design project, different specialists usually meet in order to exchange points of view. However, this approach is not enough, as just putting the people involved in the project together to talk is not sufficient to completely solve the existing problems. Besides, it is not always possible to gather everybody, especially in large-scale projects.

Essentially, this architecture intends to be:

- Generic: It is not planned to fit into a specific domain, but instead, it aims to be adaptable to a large number of design environments. Also, experts from different areas are able to collaborate through it, as its final goal is to have a common product design.

- Synchronous: Collaboration is done synchronically. That means that each designer creates his/her own proposed model and in the meantime, everybody else is able to access these models and propose changes to them. Information is exchanged during all design phases, even in the first ones, like conceptual design. This continuous feedback intends to avoid future design misunderstandings and inconsistencies.
- Service Oriented: We propose a service-oriented architecture for collaborative design. We intend to take advantage of SOA – of which one of the promises is the ease of integration of different, loosely coupled services across the Internet and intranets – and all its enhancements for interoperability.
- Agent Based: We believe that collaboration is facilitated by combining the agent- and Web-based technologies. We consider that a collaborative system should provide intelligent services to help users to solve design problems. Besides, a software agent is capable of solving well defined tasks autonomously and pro-actively in cooperation with other agents by order of personal (human) or non-personal principals.
- Ontology Based: Ontologies are considered a key technology to enable semantic interoperability and integration of data and processes. We think that the ontology-based representation is also essential to achieve genericness in our architecture. The representation of product models as ontology instances lets us process them into an inference engine, in order to mitigate and avoid possible eventual conflicts (Dutra et al., 2008).

5.2.1 Generic

As applications are migrating to heterogeneous and distributed computer environments to give support to the design and manufacturing processes, we think that genericness is a very relevant aspect to be considered when starting a collaborative design project. The modular structure of this architecture provides it with domain independence and portability. Adapting it to a given domain means only modifying data; the architecture's functional aspects (vocabulary, knowledge, etc.) remain the same.

Genericness, interdisciplinary collaboration and diverse forms of information are particular characteristics of collaborative design. A successful implementation of collaborative design would need an efficient communication strategy for a multidisciplinary group of people to share and exchange ideas and comments; an integration strategy to link heterogeneous software tools in product design, analysis, simulation, and manufacturing optimization to realize obstacle-free engineering information exchange and sharing; and, an interoperability strategy to manipulate downstream manufacturing applications as services to enable designers to evaluate manufacturability or assemblability as early as possible.

We believe that, as product development becomes more knowledge intensive and collaborative, research on knowledge retrieval, capture, accessibility, and reusability becomes increasingly important. Heterogeneous tools and multiple designers are frequently involved in collaborative product development, and designers often use their own terms and definitions to represent a product design. Thus, to efficiently share design information among multiple designers, the designer's intentions should be persistently captured and the semantics of the designer's terms and intents should be interpreted in a consistent manner. For this purpose, a

standardized data format is a prerequisite. Furthermore, the appropriate design knowledge should be provided during all design processes in order to make proper design decisions.

The use of agents is also a good approach towards achieving heterogeneity, as they encapsulate software units for dynamic and heterogeneous environments. A MAS environment should consist of a geographically distributed network to support a large number of autonomous and heterogeneous agents. Moreover, the combination of agent- and Web-based approaches can not only provide feasible system interoperation, but also increase the efficiency of the distributed collaboration. Agent-based technologies provide the mechanism for cooperative agreements and address the requirements of a dynamic and heterogeneous environment.

Finally, virtual distributed teams that work concurrently to achieve a suitable trade-off between different requirements in multidisciplinary scenarios should consolidate the management of information within different and heterogeneous teams, to provide a consistent picture of the current state of the design processes, artifacts, and models.

5.2.2 Synchronous

We mean by synchronization the continuous update of the common design models. Synchronization is a very useful approach to deal with conflicts. We think that the system should help the designers to complete their tasks efficiently and effectively by providing them with appropriate design functionalities and, if possible, synchronously.

In general, we can say that that traditional collaborative design typically relies on a sequential approach (cf. Figure 5.10). In a design project composed of four actors (e.g. one client, two designers, and one manufacturer), the traditional approach used would consist of a sequential set of steps, repeated as many times as necessary to avoid inconsistencies.

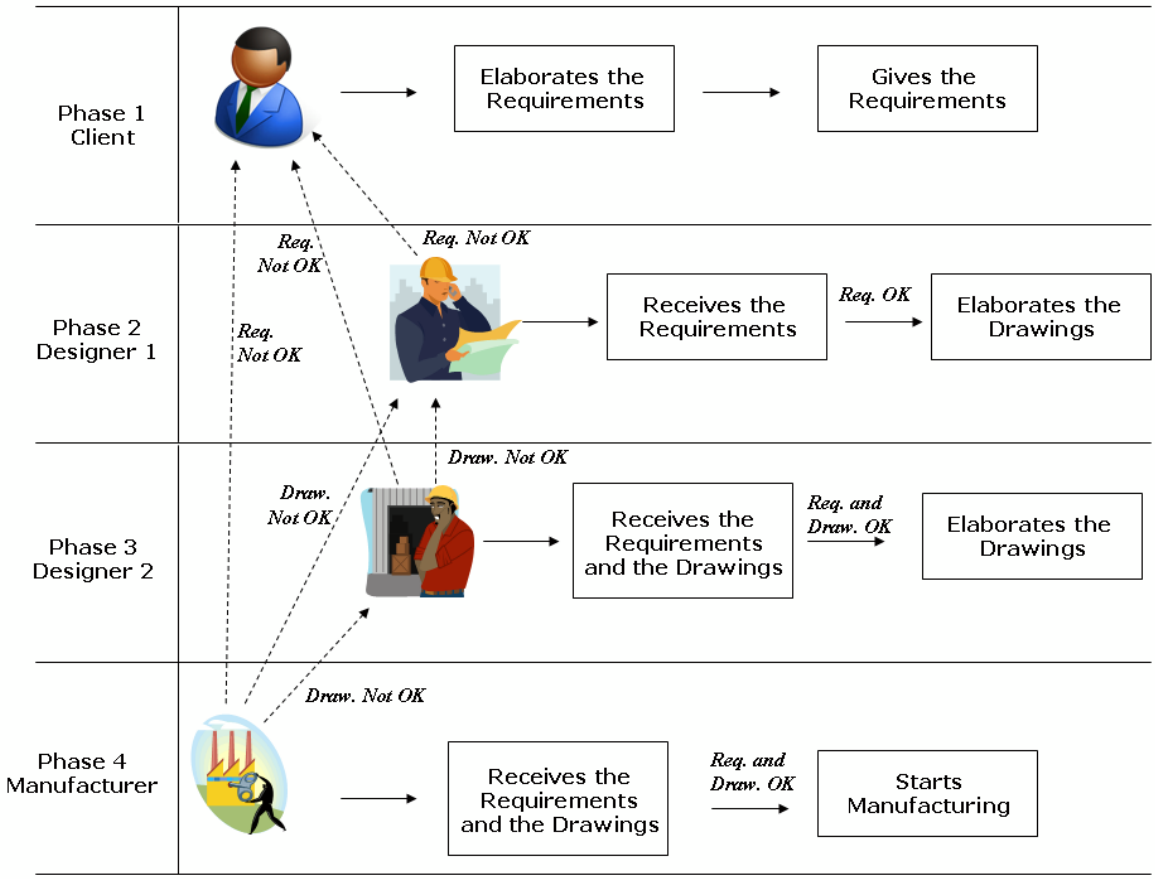

Figure 5.10 – The Sequential Design Approach

Until starting the manufacture of the product, it could be necessary to re-do the intermediate steps several times. Each disagreement or non-acceptance would immediately lead back to an earlier stage of the process. Hence it is almost unnecessary to stress that this scenario would increase production time and cost.

In the synchronical approach (Figure 5.11), on the other hand, all the actors take part in all design phases, from the beginning until the final design is done. Through the access to other designers' preliminary versions during the whole process, each designer is able to refine his/her own model in advance, before achieving a final version of the product design. This constant feedback leads to time and cost reduction, as well as letting the designers attenuate conflicting situations, by the early detection of them.

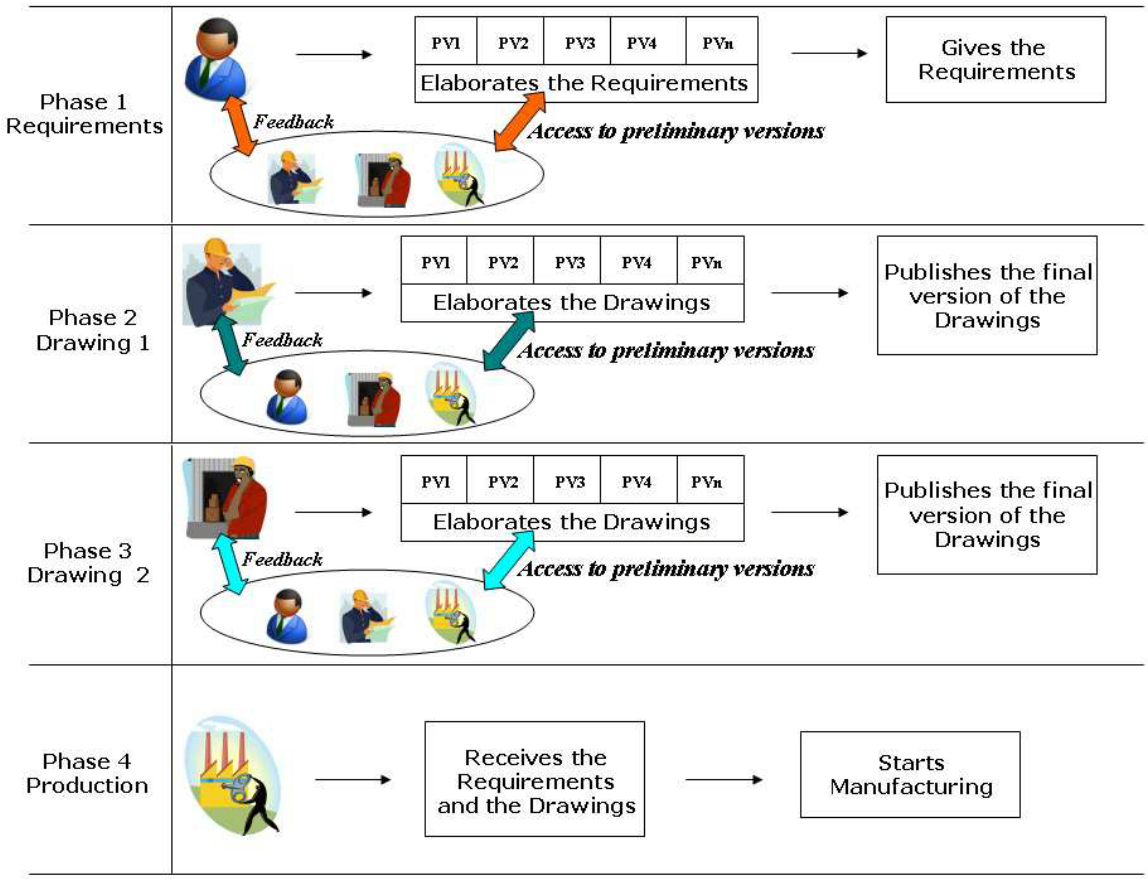

Figure 5.11 – The Synchronical Design Approach

When the manufacturer receives the product's final model, he/she is quite sure to have a consistent model, as it has been reviewed by everybody involved in the collaborative project, including the manufacturer himself/herself. In the synchronical approach, the designers exchange their models through publishing them in the shared workspaces.

5.2.3 Service Oriented

The Internet has become an integral part of business activities of most corporations today. Web-based collaborative design systems are very popular due to the current ubiquity of the Web browser as a client. This kind of system is convenient for product design sharing and review, design discussion, customer survey, among others. We believe that the ability to update and maintain Web-based systems without distributing and installing software on potentially thousands of client computers is one of the reasons for such popularity.

Aiming to provide actual and effective support for collaborative design, we define a set of services (cf. Section 5.5) – provided by different agents – as well as an interaction model to deploy different types of connections (loosely coupled) among these services or, more precisely, between the agents which provide these services. These connections have a mostly collaborative nature, i.e. they enable two or more agents to collaborate through mutual service invoked in order to assure the functionality required as support to the distributed collaborative work. For example, an agent or an agent group that provides the constraint checking service

can collaborate with one or more agents to assure the maintenance and management of the system log, so that the specific data concerning a given constraint – including its evolution in time – can be retrieved.

Service connections can also be concurrent, e.g. when two agents have the same service request (let us consider the situation where a proposition is evaluated according to different criteria, or the one in which data must be replicated to different locations). This service set is implemented through the Internet, and is composed of description, research, and service invocation interfaces that can be directly called by human experts or by system agents.

We propose a Web- and agent-based architecture for collaborative design. We believe that Web technology itself cannot satisfy the interaction of activities, such as geometric and semantic product modeling, design representation, user-interaction, and design browsing and retrieval. Therefore, we consider a hybrid approach (Web- and agent-based) the best choice, considering all the characteristics present in the proposed architecture.

5.2.4 Agent Based

In collaborative design systems, intelligent software agents have long been used to enable cooperative design among designers and to integrate legacy software tools. Agent technology is also competent in the sense of enabling the automation of complex tasks and developing reliable and intelligent systems, which has been demonstrated by various projects.

Multi-agent systems and ontologies represent a potential solution for implementing technological support for information and knowledge management in distributed engineering design. These two emerging technologies are envisioned to form the next distributed computational environment, capable of managing inherently complex and distributed systems.

Web technology alone is not the unique solution to support collaborative design systems, even if it makes communication physically viable through a common network. From a historical point of view, the agent-based technology has been used to develop collaborative design systems even before Web-based technology.

5.2.5 Ontology Based

Exchanging knowledge is a key task for performing an efficient collaborative design process. Nevertheless, this knowledge must be formalized and capitalized to be easily understandable and reusable. We consider the use of ontologies to be an excellent approach to achieve such a scenario.

One of the rare examples of consensus in the knowledge management domain is that knowledge is now perceived as an organizational and production asset, a valuable patrimony to be managed and thus there is a need for tools and methods assisting this management. Ontological engineering is currently being used by a range of functional domains to support the capture and sharing of information and knowledge.

Most of the modern CSCW (Computer Supported Cooperative Work) tools focus on communication features (messaging) and coordination (workflow) or videoconference tools, etc., but few of them are interested in collaboration among actors. Regarding this feature, besides shared documentary bases, market tools propose forums to facilitate the exchange among actors. Unfortunately, such places do not formalize inter-skill relationships and do not guide the actors towards the setting up of a common knowledge project.

The use of ontologies provides the definition of concepts in design and manufacture, and is context adaptable. Moreover, ontologies play an important role as they provide semantic interoperability and integration of data. In this context, ontologies would be required for the increasing development of semantic applications, especially in the corporate Semantic Web, which comprises the application of semantic technologies in an enterprise environment.

In Chapter 6, we are going to discuss ontology-based representation and how it can be used to deal with conflicts in collaborative design, presenting a use case. Chapter 7 depicts the conflict resolution methods used in our architecture (Dutra et al., 2008b).

5.3 System Modules

Our system is composed of three modules: (i) the management module; (ii) the blackboard; and (iii) the ontology module.

5.3.1 Management Module

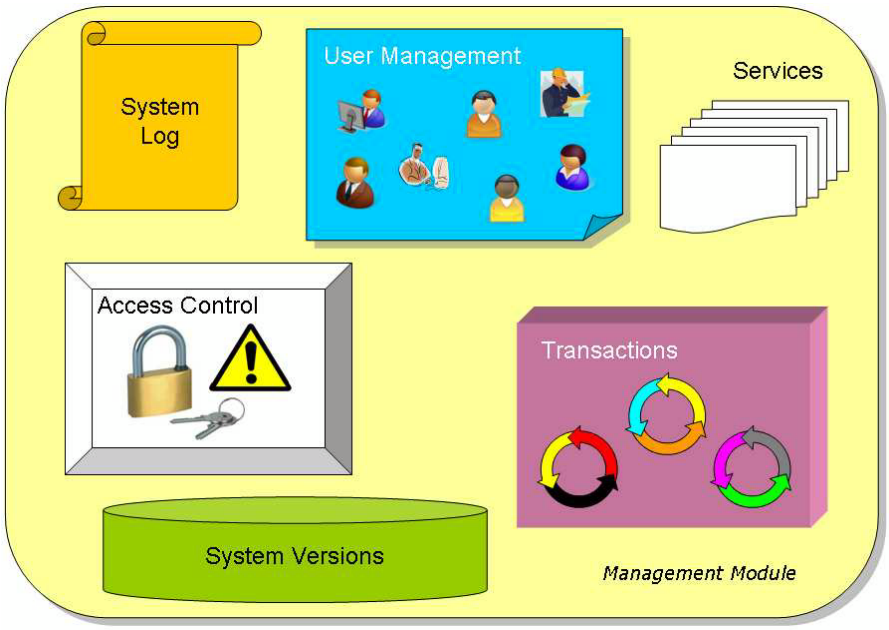

Figure 5.12 – Management Module

This module is responsible for all the operations related to the system management, namely the management of projects, agencies, and agents – including the definition of the project managers and the agency coordinators (Figure 5.12).

This module is in charge of:

- Log Management: A collaborative work environment that enables collaboration among several knowledge areas is likely to have eventual problems during the product design process. If we want the system evolution to be consistent, it is important to keep track of all operations that have been done, in order to provide a possible trace back. This stored information may permit the identification of failures, error causes, and conflicts, among others.
- Version Management: In the same way as the system log, the version management is used to undertake a trace back inside the system. The difference here is that only obsolete models (those which have been already replaced by newer ones) are stored.
- Service Management: A catalog with the services made available by the system agents. An agent that decides to create and make a service available, put its reference into this catalog – accessible to everybody in the same way as yellow pages directories are.
- Transaction Management: In a collaborative environment, sometimes the processing of some interactive transactions may depend on the result of one or more transactions. In this case, transactions should be redefined at the moment of their execution. This task provides the agents with the possibility of intervening during transaction execution, in order to introduce modifications to it. These modifications are done incrementally, i.e., without suspending the transaction in progress.
- User Management: Human resource management is a crucial task during product development. It deals with how to organize and distribute human resources in the best way so that all employees in the organization can be able to contribute their abilities as much as possible. This submodule consists fundamentally of a range of tasks designed to ensure that the appropriate number of the appropriate personnel are in the right place at the right time. In essence, it involves assessing current levels and utilization of staff and skills, relating the internal elements to the market demand for the organization's products, and providing alternatives to match human resources with anticipated demand. At a lower level, it provides the services that are required to manage the system users, such as definition of login, password, role, personal information, and access control.
- Access Control: It is important that the application data for each of the collaborative activities be well protected, so that the partnership can be implemented smoothly. Data protection provides protection from improper access, protection from interference, integrity of data, operational integrity of data, semantic integrity of data, user authentication, management and protection of sensitive data, and multi-level protection. A control policy defines the authorization rules on how subjects and objects can be grouped to share access modes. Besides, a control procedure checks queries against the stated authorization rules to determine the access modes.

5.3.2 Blackboard

In order to provide reliable and relevant software solutions for collaborative product design, this architecture has a global repository for integrated product models, processes and organization (Dutra et al., 2009). The development of computer support for collaborative design should focus on the ways of improving the interaction between collaborating designers in order to establish a suitable environment for sharing design information and knowledge. Dealing with the integration of the design activity in its context is a key element to meet these requirements.

Agencies are organized aroung a blackboard-type system. Each agency publishes its models to the others through a shared blackboard. For example, after the design elaboration of the function model, the design agency notifies other agencies by publishing this model into the blackboard.

In our architecture, the blackboard is the quintessential collaboration space. It comprises two subspaces: Solution Space and Collaboration Space. The blackboard is present in both the project's and agencies' workspaces. Within each agency – as in the project level – the backboard is composed of these two subspaces.

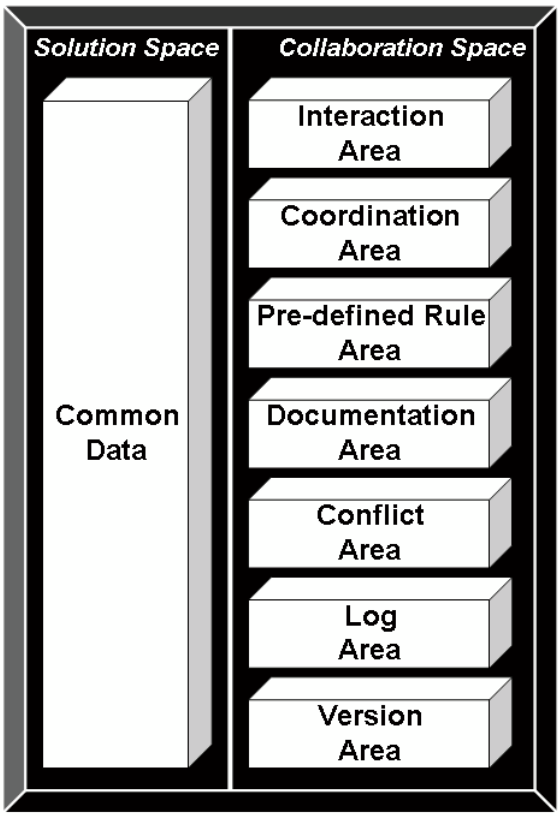

Figure 5.13 – Blackboard Structure

The collaboration space is where collaboration effectively takes place. It comprises seven sub-areas (Figure 5.13):

- Interaction Area: This is the agents' communication area. In there, they are able to notify one another, to leave messages, to make propositions, to make suggestions, and to argue. It is used to clarify global questions/problems related to the design process. A bulletin board allows team participants to post, read or respond to messages used mainly for discussions within a team. All bulletin board information is stored in the database. The database also stores message information such as message subject, posting dates, poster's information, and the content of messages. According to their privilege, team participants can read, post, reply, modify, and delete messages and also investigate the user information.
- Coordination Area: This is the workspace of the project manager/agency coordinator. This area contains information concerning the different strategies to be used in the collaborative project – related to the experts' coordination and collaboration – such as dependency relationship between handled objects and the agents that work with them (i.e. hierarchy, composition and function-dependence relationships). The information stored in this area is fundamental to assure the support for the collaborative design process. Once the system is started up, this area is initialized to serve as parallel support to the design process.
- Pre-defined Rule Area: Stores the rules that will be used for the constraint checker submodule, during the process of publishing ontology instances.
- Documentation Area: Stores the system shared documents. Documents can be put into and removed from this area by the agents. They are organized by index and by author, to ease their access.
- Conflict Area: This area is activated whenever a conflict is detected. It stores the data and the procedures used during the conflict resolution process (Dutra et al., 2008b). In the next chapter, we are going to detail this area.
- Log Area: Stores all the historical information that is relevant to system usability and maintenance, to be later used by the management module.
- Version Area: Keeps old versions of the proposed models, for consultation/restoration.

The solution space contains the common data, the merged ontology instances (produced after the collaboration process), and the predefined ontologies (for the generic approach) to be used as "standard models" by the agents. The purpose of the solution space is to provide a centralized access service for the storage and retrieval of design process data during the execution of the system. It serves the system not only by storing and retrieving engineering data but also by automating data exchange among different design activities. The solution space is responsible for keeping and retrieving all relevant information of the application resources that have been registered with the application resource manager.

The solution space is accessible to all agents. During the collaborative activity, the agents put models/data into or get models/data from this space, as well as the initial and intermediate results of the reasoning activity.

5.3.3 Ontology Module

Each expert designs his model according to his expertise and knowledge skills. A mechanical engineer, for instance, will naturally be concerned about the mechanical structure of the product and its components. Meanwhile, a thermal engineer will be more concerned about heat and temperature control for a specific part of the same product. Here in the ontology module is where the different proposed models are merged together (Figure 5.14). It is divided in two parts: a Constraint Checking Module, and an OWL Reasoner (Dutra et al., 2007b).

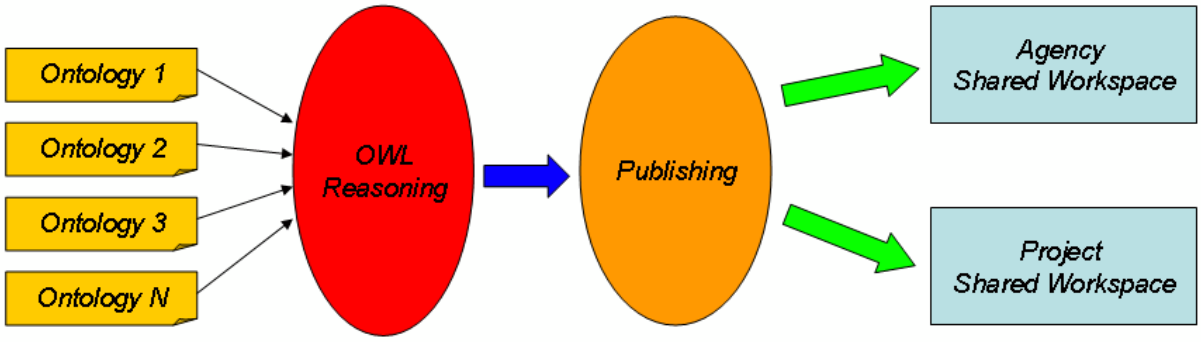

Figure 5.14 – Ontology Reasoning

Constraint checking is used as a conflict attenuator. It uses predefined rules/statements to ensure that only consistent data will be published. Constraints are used to guarantee the consistency of the data handled by the system (cf. Chapter 7). They are also used to represent the system's requirements, which are represented as groups of variables in spaces of feasible values. Such spaces improve efficiency by avoiding artificial conflicts, improving design flexibility, enhancing change management and assisting conflict. Constraint checking is an automated task, done to verify the consistency of a given model. Defined constraints may be relaxed during the negotiation process – if it is necessary – to facilitate the search for a solution.

The OWL reasoner checks the consistency of the given ontology instances/models, expressed in OWL. That is, it verifies whether there are any logical contradictions in two or more ontology axioms (Dutra et al., 2008). Furthermore, it infers whether a particular ontology concept is a subconcept of another, or whether a particular individual in a given ontology belongs to a specific class. The ontology module is activated whenever an ontology publication is done in both collaboration levels.

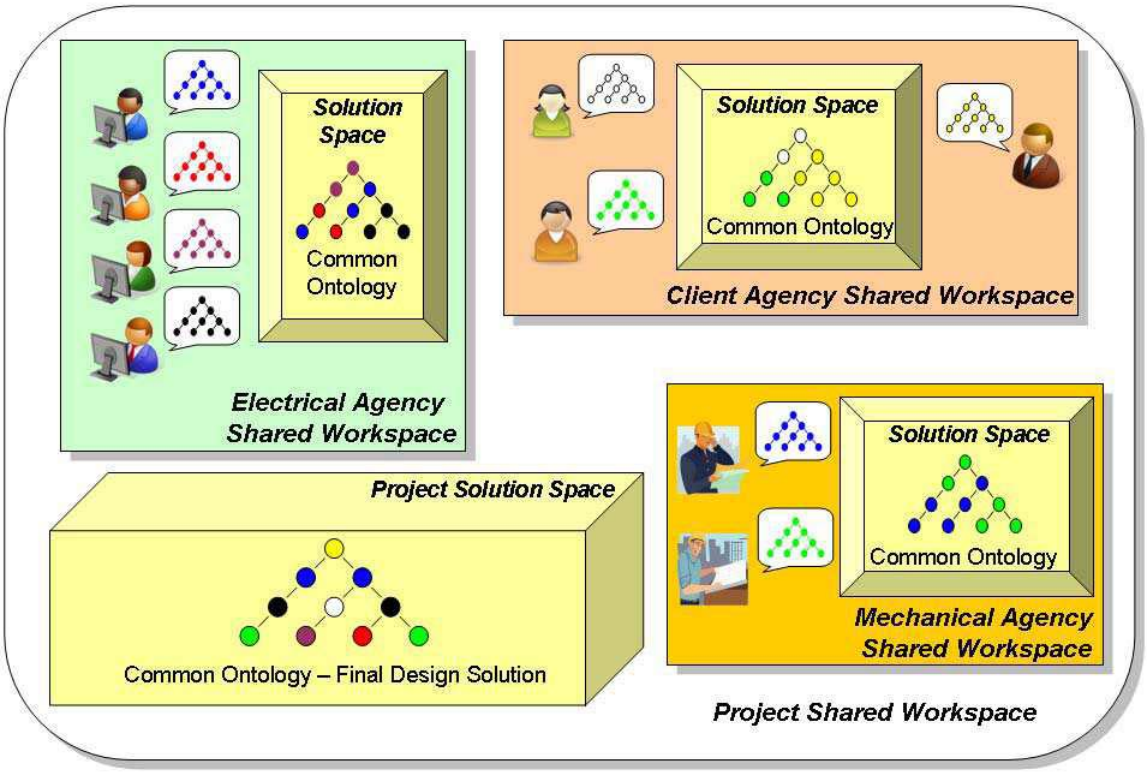

Figure 5.15 – Ontology Integration

To illustrate this scenario, let us consider a design project of a small electrical piece, with clients, electrical and mechanical engineers as participants. We have three agencies taking part in it: Client Agency, Electrical Agency, and Mechanical Agency. Private models are done by the agents and merged together into the agency shared workspace. These resultant models are then merged together into the project shared workspace. The resulting ontology instance placed in the Project Solution Space will be the final design solution of the collaborative project (Figure 5.15).

The following sections depict the system roles of our architecture.

5.4 System Roles

A collaborative system needs two kinds of capabilities and facilities: distribution and collaboration. These two terms emphasize the different aspects of a system: physically, the former separates collaborative systems as geographically dispersed and expands them to support remote design activities, and functionally, the latter associates and co-ordinates individual systems to fulfill a global design target and objective. Collaboration is driven by the design and development of effective collaboration mechanisms to facilitate human–human/human–computer relationships.

Members of multidisciplinary design teams composed of experts in each discipline might work at different design phases or on different design versions and need to exchange data for

the purpose of collaborative design. Furthermore, various engineering computer tools used in design are often geographically distributed and implemented on different, possibly heterogeneous platforms such as UNIX, Macintosh, and Windows. Temporary teams of experts are characterized by constant changes in their composition, as team members are selected according to the specific requirements of every design project. The criteria for this selection are usually the kinds of expertise of the individual designers that are expected to add up to form the desired kind of team expertise. However, as the individual experts have gained experience from previous work in similar teams, they have a potential to build up team expertise with less and less effort. The key for this potential is their ability to learn by forming generalized knowledge of different team environments and individual experts. Large parts of these models can be derived from generalized experiences.

The system roles are a user classification regarding the different tasks the system users are in charge of.

5.4.1 System Administrator

The System Administrator is responsible for the Management Module (cf. Section 5.3.1). Figure 5.16 shows us the UML use case diagram for this special user:

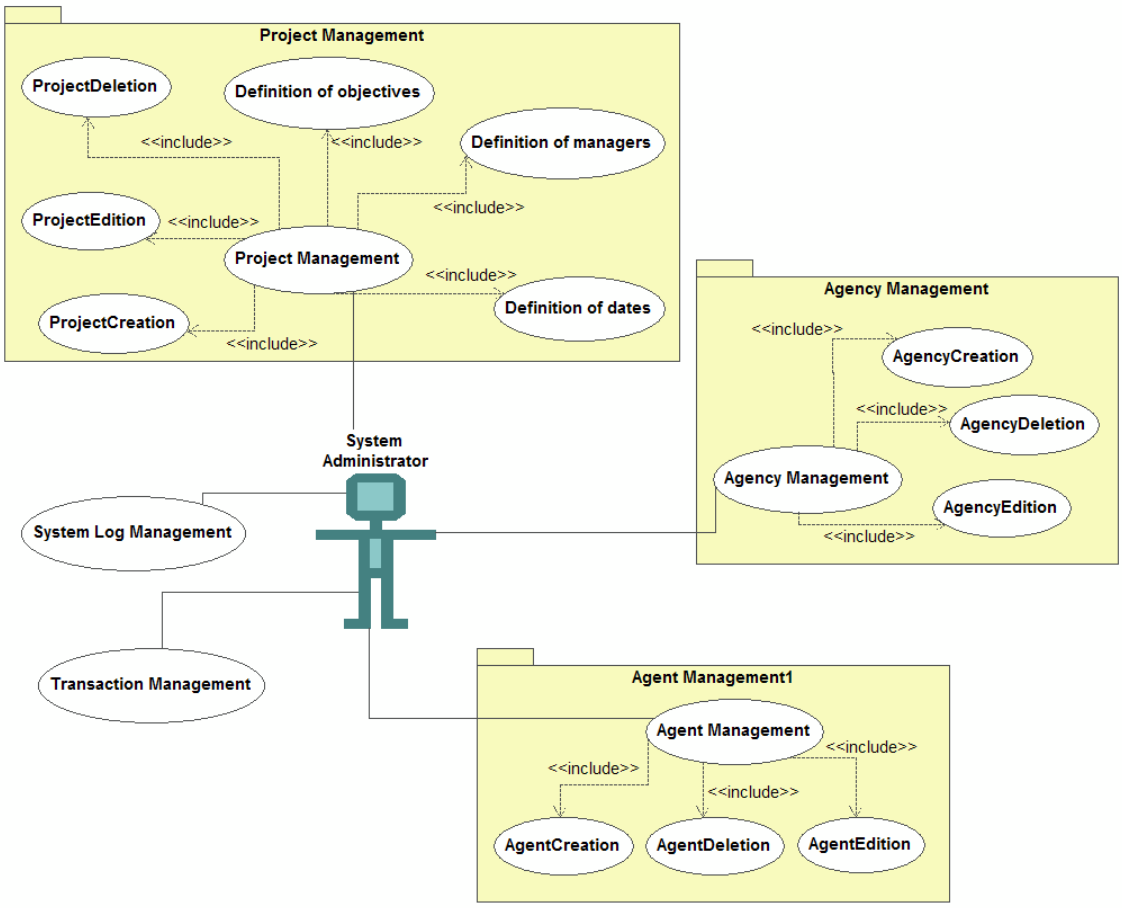

Figure 5.16 – System Administrator Use Case Diagram

The system administrator is responsible for:

- Project Management:
  - Creation/edition/deletion;
  - Definition of project's objectives;
  - Definition of starting and ending dates;
  - Designation of project managers;
- Agency Management:
  - Creation/edition/deletion;
- Agent Management:
  - Creation/edition/deletion;
  - Definition of the agent's personal information;
  - Definition of access rights;
  - Definition of initial login and password;
  - Definition of role (simple user, agency coordinator, project manager);
- System Log Management;
- Transaction Management.

5.4.2 Simple User

The simple user (Figure 5.17) is the system user with no further attributions. Essentially, he/she is a participant of the project, in charge of providing his/her own contribution to the final design model. The simple user has access to a private workspace, as well as to all shared workspaces (*intra-* and *inter-agency*), to the system log and to the blackboards.

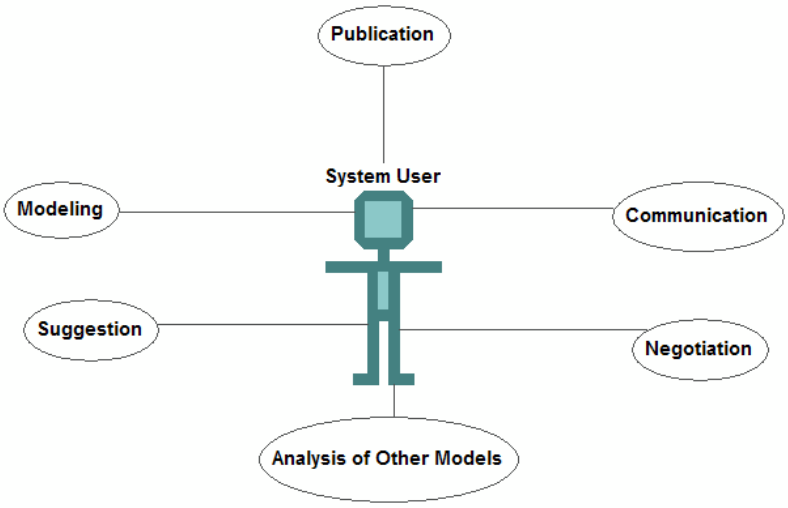

Figure 5.17 – Simple User Use Case Diagram

5.4.3 Project Manager

To increase design performance and consequently to satisfy customer requirements and the expectations of enterprises, the decision maker has to adapt the designers' context of work to the environment of the design process. So the actors' context of work will be improved, and since the decision maker will be able to create effective working groups according to assigned design objectives, the phase of human resource allocation will be more efficient. In our architecture, this decision maker is the project manager.

The management of the design environments comprises a continuous phase of adjustment and evolution of the environment, according to the design situation evolution. Collaboration is a major feature in teamwork-based projects, which are frequently implemented by high performance project teams. Effective project teaming thus has become essential in project management. In project management, project teaming refers to managing a project team with the assignment of project tasks and roles, and the appropriate composition of the development or workplace team that performs ad-hoc project tasks. The importance of team composition is easy to understand, but as it is sufficiently complex to be accomplished, projects often differ considerably, resulting in complex and varied human resource requirements.

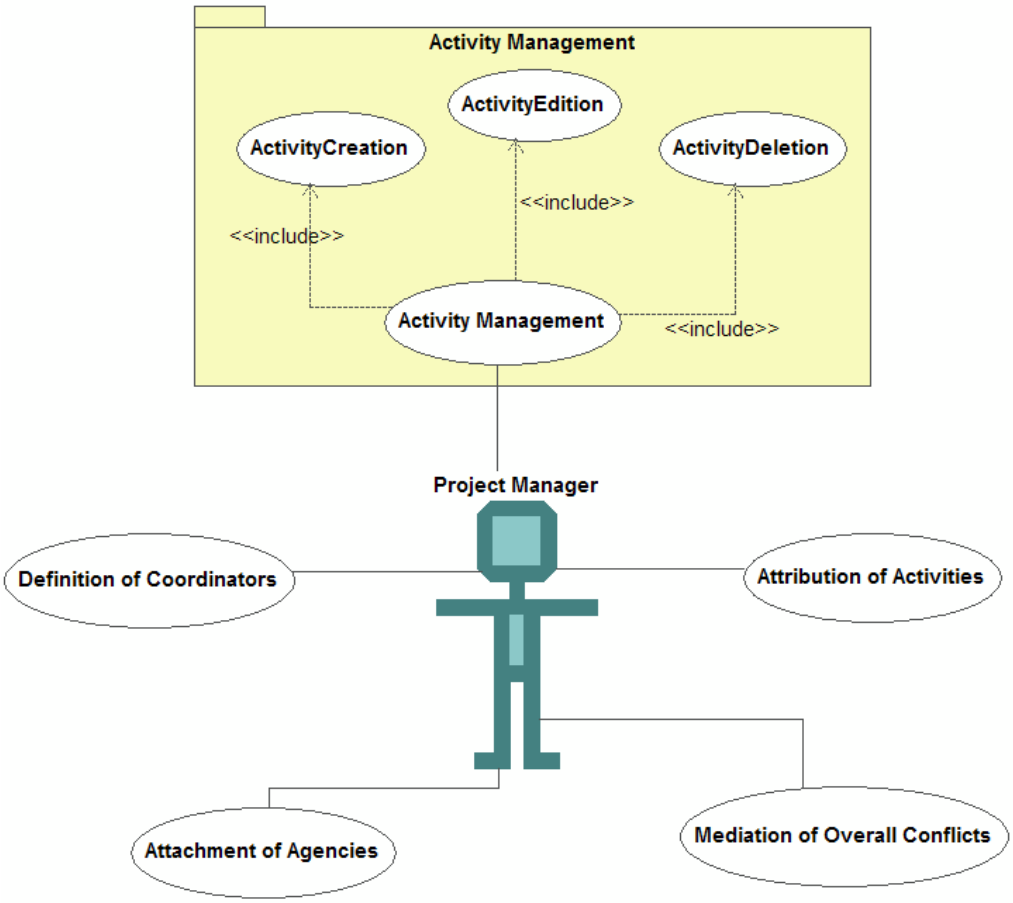

Figure 5.18 – Project Manager Use Case Diagram

In our architecture, the project manager is a user designated to coordinate a design project. He/she inherits all simple users' features, but extends them with the following ones (Figure 5.18):

- Project Management:
  - Creation/edition/deletion of activities;
  - Attribution of activities to agencies;
  - Attachment of agencies to projects;
  - Attachment of agents to agencies;
  - Definition of agency coordinators;
  - Mediation of overall (*inter-agency*) conflicts.

5.4.4 Agency Coordinator

Complex design tasks often necessitate the confluence of various special knowledge and skills. Typically, this heterogeneous set of expertise is embodied in a multitude of designers who come together in unique combinations to form temporary, project-specific teams. The mere collection of different experts, however, is not sufficient to establish a good team. In order to achieve good performance as a team, the individual team members must be able to interact using the results of individual expertise in a way suitable for the current task. This ability can be enhanced using various technologies and infrastructures; however its major part is generally formed through continuous processes of learning each other's roles, responsibilities, priorities, practices, etc. In this context, we believe that a coordinator is fundamental to a good agency operation.

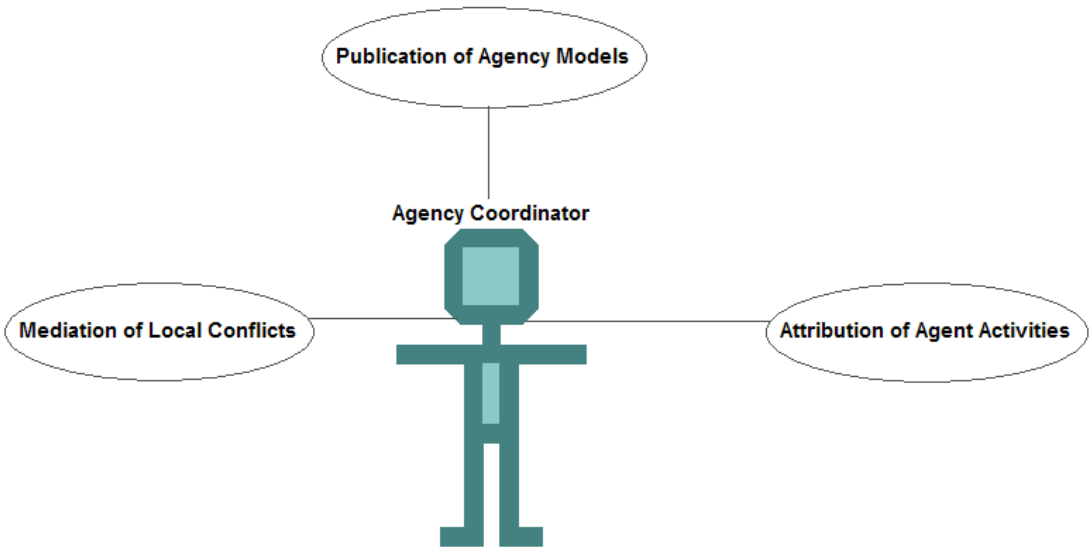

Figure 5.19 – Agency Coordinator Use Case Diagram

A coordinator (Figure 5.19) is an agent chosen to be the manager of the internal activities that take place in the agency, and a broker for external communication with other agencies (Dutra

et al., 2009). It is up to the coordinator to decide the moment to publish the model (or ontology instance) agreed upon by the cluster participants. Publishing an ontology instance means that it will be available for other agents/agencies to access it and work on it. Furthermore, the coordinator acts also as a moderator in the moment of a conflicting situation.

The agency coordinator as well inherits all simple users' features. His/her specific tasks are:

- Attribution of activities to agents;
- Mediation of local (*intra-agency*) conflicts; and
- Publication of agency models into the project shared workspace.

5.4.5 Ontology Expert

An ontology expert is a user designated to be the one who creates/defines the project ontologies. That is, the ontologies that will serve as a base for the agents' ontology instances in the generic approach. Typically, this expert should be someone wise enough to understand the whole spectrum of knowledge used by the project. Eventually, for larger and more complex projects, this user can be more than one person. Figure 5.20 shows the ontology expert use case diagram.

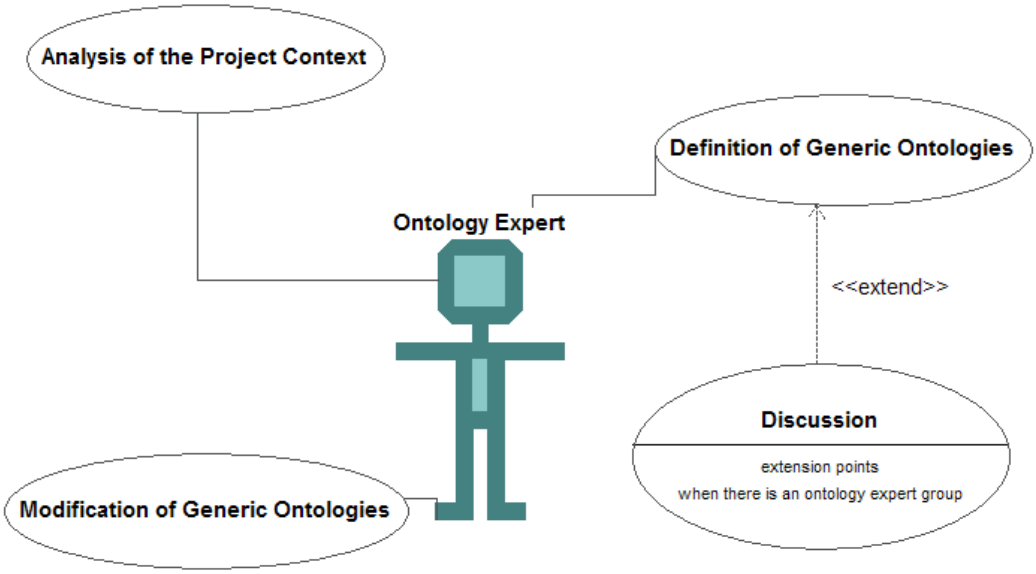

Figure 5.20 – Ontology Expert Use Case Diagram

5.5 Collaborative Services

An application server has been chosen as a support to our collaborative environment. Consequently, all the system functionality is provided through the use of web services. We have classified the required services to be implemented into three distinct categories: system services, generic collaborative services, and specific collaborative services. System services

encapsulate functionalities from the Management Module. These services are called by the system administrator and by the project/agency coordinators to undertake operations related to the system administration tasks, such as creation of a new agent, edition of project descriptions, inclusion of a new agency's activity, etc.

Generic collaborative services are used to access the Ontology Module and are called by all collaborative transactions. Among the generic services, we can highlight the following ones:

- Services tailored to deal with ontologies, such as creation, edition, appending, deletion, publication, importation, among others.
- Services implemented to support blackboard's collaboration space, e.g., services for sending messages of notification, questions, suggestions, arguments and general interaction.
- Services to support coordination messages.
- Constraint checker services: used to implement rule-based support and constraint checking.
- Log services: used to manage the system log, in order to store old conflicting situations and their solutions.

These services act also as a support for the analogy-based approach – during the conflict resolution process –, as they are used for scanning the system log in order to find a matching conflicting situation. Specific collaborative services are tailored to fulfill particular ontology-based situations. For example, consider a design project where a generic ontology is given. This ontology is supposed to be used by all the designers to compose their models. Considering that working with this predefined ontology demands particular actions to be taken, specific collaborative services should be implemented to provide them. We have tailored our platform according to the composed SOA/Web Services paradigm. We propose a set of loosely coupled services, which are specified in SOAP and WSDL. The combination of both of these XML-based formats ensures great interoperability among heterogeneous systems.

In our architecture, Web Services are implemented in Java, providing a cross-platform communication medium among the multiple tools involved in the ontology manipulation. Moreover, this strategy permits us to keep the same programming language as the system interfaces are developed with JavaServer Technology (JSP). JSP is designed to create dynamic web pages. Once the services are deployed through JSP, any programming language can be used to access them, as long as they follow the defined service specifications. Users access these services through JSP interfaces. Web Services, like HTML or PHP Web pages, need a Web server to run. We have chosen Apache Tomcat as the Web server. Tomcat is a free JSP/Java Servlet cross-platform server that allows the deployment of Java WS through an Axis2 WS engine.

Table 5.1 shows a WSDL specification file used in our architecture. It specifies a service called *Publisher* and comprises the *publishModel* method, which is used by the agents to publish their particular models into the agency shared workspaces.

WSDL Specification

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:ns1="http://org.apache.axis2/xsd" xmlns:wsaw="http://www.w3.org/2006/05/addressing/wsdl"
xmlns:ax25="http://io.java/xsd"
xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
xmlns:xsd="http://scoop"
xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/"
targetNamespace="http://scoop">
    <wsdl:types>
        <xs:schema attributeFormDefault="qualified" elementFormDefault="qualified"
targetNamespace="http://io.java/xsd">
            <xs:complexType name="InputStream">
                <xs:sequence/>
            </xs:complexType>
            <xs:complexType name="ObjectInputStream">
                <xs:complexContent>
                    <xs:extension base="ax25:InputStream">
                        <xs:sequence/>
                    </xs:extension>
                </xs:complexContent>
            </xs:complexType>
        </xs:schema>
        <xs:schema xmlns:ax26="http://io.java/xsd" attributeFormDefault="qualified"
elementFormDefault="qualified" targetNamespace="http://scoop">
            <xs:import namespace="http://io.java/xsd"/>
            <xs:element name="publishModel">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element minOccurs="0" name="stream" nillable="true"
type="ax26:ObjectInputStream"/>
                        <xs:element minOccurs="0" name="fileName" nillable="true" type="xs:string"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
            <xs:element name="publishModelResponse">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element minOccurs="0" name="return" nillable="true" type="xs:string"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
        </xs:schema>
    </wsdl:types>
    <wsdl:message name="publishModelRequest">
        <wsdl:part name="parameters" element="xsd:publishModel"/>
    </wsdl:message>
    <wsdl:message name="publishModelResponse">
        <wsdl:part name="parameters" element="xsd:publishModelResponse"/>
    </wsdl:message>
    <wsdl:portType name="PublisherPortType">
        <wsdl:operation name="publishModel">
            <wsdl:input message="xsd:publishModelRequest" wsaw:Action="urn:publishModel"/>
            <wsdl:output message="xsd:publishModelResponse" wsaw:Action="urn:publishModelResponse"/>
        </wsdl:operation>
    </wsdl:portType>
    <wsdl:binding name="PublisherSoap11Binding" type="xsd:PublisherPortType">
        <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
        <wsdl:operation name="publishModel">
            <soap:operation soapAction="urn:publishModel" style="document"/>
            <wsdl:input>
                <soap:body use="literal"/>
            </wsdl:input>
            <wsdl:output>
                <soap:body use="literal"/>
            </wsdl:output>
        </wsdl:operation>
    </wsdl:binding>
    <wsdl:binding name="PublisherSoap12Binding" type="xsd:PublisherPortType">
        <soap12:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
        <wsdl:operation name="publishModel">
            <soap12:operation soapAction="urn:publishModel" style="document"/>
            <wsdl:input>
                <soap12:body use="literal"/>
            </wsdl:input>
            <wsdl:output>
                <soap12:body use="literal"/>
            </wsdl:output>
        </wsdl:operation>
    </wsdl:binding>
    <wsdl:binding name="PublisherHttpBinding" type="xsd:PublisherPortType">
        <http:binding verb="POST"/>
        <wsdl:operation name="publishModel">
            <http:operation location="Publisher/publishModel"/>
            <wsdl:input>
```

```
                <mime:content type="text/xml" part="publishModel"/>
            </wsdl:input>
            <wsdl:output><mime:content type="text/xml" part="publishModel"/>
            </wsdl:output>
        </wsdl:operation>
    </wsdl:binding>
    <wsdl:service name="Publisher">
        <wsdl:port name="PublisherHttpSoap11Endpoint" binding="xsd:PublisherSoap11Binding">
            <soap:address location="http://localhost:8080/axis2/services/Publisher"/>
        </wsdl:port>
        <wsdl:port name="PublisherHttpSoap12Endpoint" binding="xsd:PublisherSoap12Binding">
            <soap12:address location="http://localhost:8080/axis2/services/Publisher"/>
        </wsdl:port>
        <wsdl:port name="PublisherHttpEndpoint" binding="xsd:PublisherHttpBinding">
            <http:address location="http://localhost:8080/axis2/services/Publisher"/>
        </wsdl:port>
    </wsdl:service>
</wsdl:definitions>
```

Table 5.1 – WSDL Specification File

5.6 Summary

Providing a framework to support effective collaborative design is not an easy task to accomplish. There are a lot of variables to be taken into account, such as communication infrastructure, coordination, accessibility, available tools, and knowledge representation, among others. In this chapter, we have presented a generic and synchronous Web-, agent- and ontology-based architecture for collaborative design. The use of the heterogeneous and synchronous approaches allows the designers to interact at the same time that their models are being produced, while letting them keep working with the knowledge representation models that they are familiar with.

We consider that knowledge management in design has become an important area of research for collaborative systems. The existing challenge is to capture and re-use the existing designs and representation models previously generated, by using and combining several techniques from the knowledge representation domain, such as reasoning and semantic resource utilization.

In our collaborative architecture, we chose to use ontologies to model these different kinds of knowledge and expertise, because we consider that they are a very efficient approach to represent knowledge in collaborative environments. Besides, the ontology reasoning process facilitates coping with conflicts in early-stage design, as it permits the detection of inconsistencies. The lastest trends in this research domain – along with the several ongoing projects that work with ontology merging and aligning – have encouraged us to keep going in this direction.

In our proposal, the whole conflict resolution process relies on the ontology-based representation of information. First, we formalize the existing representation models (e.g. EXPRESS language (STEP Part 11, 2002), relational database, etc.) in order to achieve a common format (OWL). Subsequently, we harmonize the legacy ontologies – already expressed in this common format – in order to generate a “generic ontology” to be used in the design process. Hence, two different scenarios should be considered. The first one is the scenario where a generic ontology is given, i.e. defined before the starting of the collaboration process. In this context, designers should take this generic ontology and use it to build their personal models. The second scenario is the one where the design modeling starts with the use of different ontologies. Here, each designer has his/her particular model for content

representation (e.g. Function-Behavior-Structure framework (Gero et al., 2004), STEP-based models, etc.). In that case, the collaborative platform is responsible for harmonizing these legacy models, and merging them in order to achieve a common model. This common model is then used as the generic ontology for the sequence of the design (including the reasoning process).

In the next chapter, we present how the knowledge is represented in our architecture and how the formalization of formats and the harmonization of models for content representation can help mitigate the conflicting situations.

Chapter 6

6 An Ontology-Based Methodology to Deal with Conflicts

Ontologies have been realized as the key technology to shaping and exploiting information for the effective management of knowledge and for the evolution of the Semantic Web and its applications. In such a distributed setting, ontologies establish a common vocabulary for community members to interlink, combine, and communicate knowledge shaped through practice and interaction, binding the knowledge processes of creating, importing, capturing, retrieving, and using knowledge.

Nevertheless, taking into account collaborative environments and conflicts that eventually pop up, we consider that there is a lack of solutions that – using ontologies to represent knowledge – can efficiently solve the conflict problem. In this chapter, we present our methodology for knowledge representation, which facilitates the handling and resolution of conflicts. The remainder of it is organized as follows. Section 6.1 presents our proposal for knowledge formalization, as well as the EXPRESS to OWL translator, a tool developed to enhance the performance of the formalization process. In Sections 6.2 and 6.3, we present the two methods used for ontology integration, the generic ontology and the harmonization methods. In Section 6.4, we present an example of the reasoning process by using the modeling of a small mechanical piece. Finally, in Section 6.5, we have the chapter's summary.

6.1 Formalization Process

The complexity of cooperative project work can be reduced if project data and information are free of ambiguity and redundancy and presented in a format and structure that enables easy use by the project members. The explicit description and defined data structures facilitate the integration of various sources (data and information) and therewith the interoperability within the project. The necessary level of domain description can be achieved through classes which are organized as taxonomies with assigned individuals (instances) and defined relations between them in one model. These models are the ontologies, which can be used to develop a knowledge base for one particular domain under consensus of the involved partners as a means to collect and structure the existing knowledge as well as the generation of new knowledge through reasoning.

While concrete artifact models, such as detailed geometry information in CAD documents, capture the artifact in full detail, these representations are often too fine-grained to draw useful inferences. Instead, a more abstract representation is desired that captures only those

properties that are relevant for a given analysis task. For example, to assess whether two results obtained from finite-element simulations of different parts of a vehicle body may be assembled into a larger model, the detailed geometry information may not be relevant, but the material properties and version number of the simulation software are critical. Similarly, essential requirements and assumptions underlying individual analysis sub-processes can be aggregated into abstract representations.

Through the increasing informational complexity of consumer products, the communication needs of customers regarding the products at the point of sale rise intensely. Products possess a lot of different information, such as product descriptions, service and support features or user generated or professional reviews, and are linked to compatible or alterative products. Design diagrams are an explicit representation of the artifact's geometry, and it is reasonable to expect that categorization be based on 2D criterion that incorporates geometric properties which are:

(i) generic, that they have applicability over a wide spectrum of application domains, characterize as many physical dimensions of the diagram as possible including orientation, distance and topology;

(ii) have both local and global support, i.e., they should be computable on shape primitives and spatial relations;

(iii) provide descriptions capable of higher-level semantic mapping, are invariant over ranges of viewpoint and scale; and

(iv) are robust, and readily detectable by procedures that are computationally stable.

Products are physical goods that are described in a standardized and non-standardized way. A product description constitutes conceptual descriptions of knowledge about products; it changes frequently and increases fast. A digital product description within the scenario of a communication between physical products and customers in physical environments has to fulfill specific requirements, such as:

- Logical semantic structure. The digital product description needs a logical semantic structure for the automatic processing and extension of data.
- Product descriptions. The opportunity of combining product descriptions constitutes the precondition for enabling the communication between products and the dynamic linking of product bundles.
- Mapping of current standards. To avoid the development of an isolated application, the product description has to be able to map external product standards.
- Distributed repositories. Digital descriptions of products may not form a closed system; they are stored in Web-based distributed repositories and provide diverse kinds of information.
- Natural Language Processing. The more natural a communication with a product the higher is the acceptance of services and information.
- Dynamic and automatic extension. The opportunity to extend a digital product description dynamically and automatically allows the integration of contextual information about the physical environment or reasoned conclusions into the semantic description in real-time.

Proper representation of important properties of design artifacts is vital for reasoning about the design process, prerequisites, and results. While task-specific aspects and concrete execution scenarios can be captured in process ontologies, process representations must be complemented with (abstractions of) artifact models to enable comparison of different analysis tasks and their resulting models. In this work, we focus on representation models that are computationally representable and reasonable.

Furthermore, proposals of standardization (like the one presented in Section 6.1.1) can be useful to integrate different standards of representation. We can thus envisage a scenario in which different kinds of representation formats can be used to express user requirements in a collaborative design system (Figure 6.1).

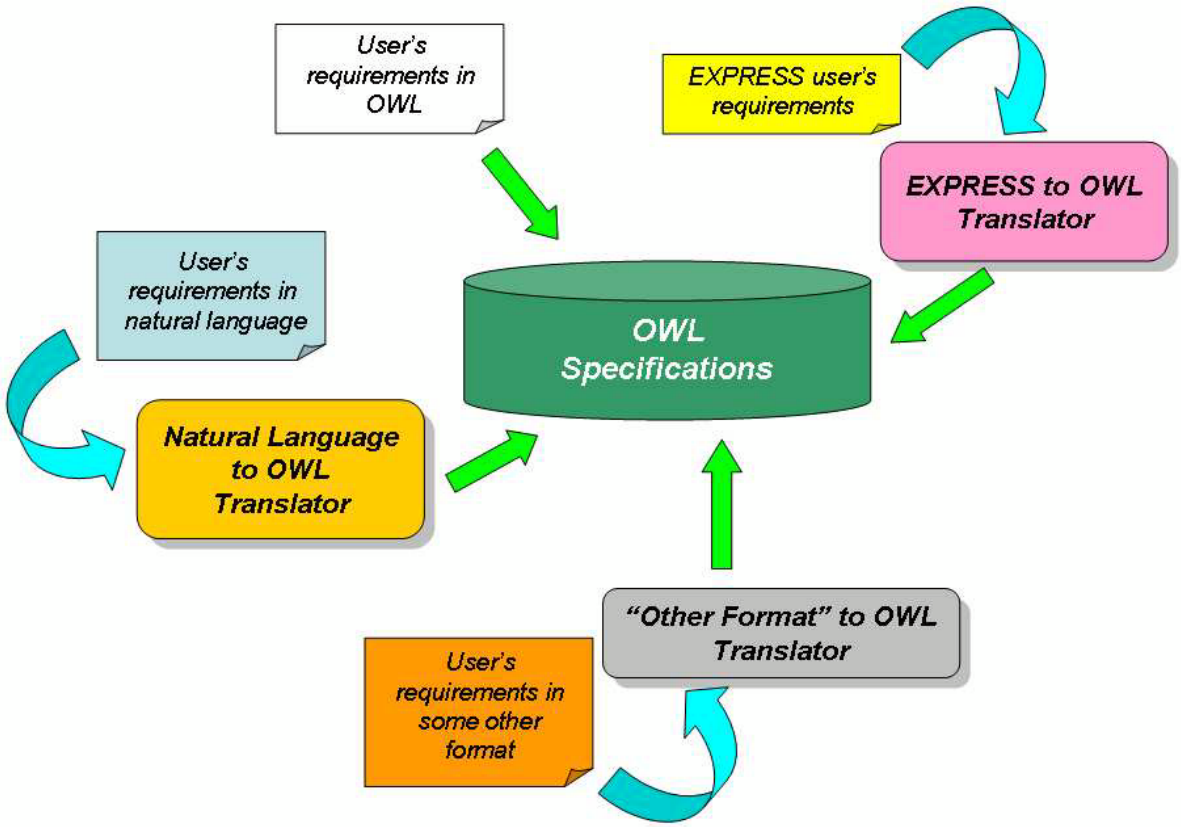

Figure 6.1– Formalization Process

Our methodology works with graph-based representation of knowledge. Graph-based representations can provide a visual and distinct view of the ontology for the users. A conceptual graph is a structure of concepts and conceptual relations where every arc links some conceptual relation *r* to some concept *c*. They are bipartite, which means that every arc of the conceptual graph links a concept to a conceptual relation; there are no arcs that link concepts to concepts or relations to relations. The graph-based representation is sometimes structured with hierarchical relations, which are the core components of all ontologies and are usually organized into hierarchies of concepts. In the concept hierarchy, concepts that are more general or more abstract have a higher level. The concepts that have a higher level are

called super concepts, while the concepts that have a lower level are called sub concepts, and are denoted by predication.

Currently, schema matching systems represent models as directed labeled graphs to support the matching of models from different meta-models. However, the way how a model is encoded as a graph is crucial for the match result as structural similarities are also important in schema matching. As models from different meta-models are represented differently in graphs (different labels, different structures), the matching between such models may produce poor results.

With so many different modeling and implementation standards being used nowadays, STEP is one of the most distinguished regarding product data. To promote the reusability of its industrial standards, ISO has adopted the modular approach of STEP to enable more efficient development, standardization, implementation and deployment. The modular standards may raise the problem of becoming quite hard to understand due to vague definitions not associated with any particular environment (Dutra et al., 2007). Yet, other problems arise when the chosen product model is described using one particular technology (e.g. EXPRESS) and is required to be integrated with end-user systems that use totally different technologies with different degrees of expressiveness, such as XML or OWL.

The integration of the EXPRESS language with the Web Ontology Language can be the means to get around the aforementioned STEP problems, since OWL provides a valuable link with the emerging field of the Semantic Web, which is gaining high relevance in the global market, and has XML syntax for easy data exchange using Web-based systems. The Semantic Web extends the current Web infrastructure, giving the information a well defined meaning that enables software agents and people to work in cooperation by sharing knowledge. Thus, if STEP standards are transformed to OWL, they can, in the future, be easily complemented with links to semantic information, which contextualizes the scope of the defined concepts regarding the environment where they are applied.

Moreover, representing EXPRESS modules as ontologies enables the use of OWL reasoning, a very powerful way to check inconsistency and incoherence of information. This can lead us to a scenario where human users can, in an easier way, exchange and verify EXPRESS represented data. Such a scenario can enhance the use of the EXPRESS language, promoting its adoption by a large number of platform-independent and language-independent users. With this in mind, we have developed a tool to transform EXPRESS models to OWL classes. EXPRESS (STEP Part 11, 2002) is STEP's modeling language.

The STEP standard, despite being very powerful regarding the representation and the exchange of product data, is not very popular among the application developer community. Therefore, and because of the massive adoption and deployment of other standard technologies, such as XML and UML, we believe that the path to follow is to harmonize these standard technologies, combining the cemented knowledge gathered by STEP with the popularity of other standards. This harmonization among complementary technologies has the potential to foster the exchange and sharing of digital data. However, these standards and the STEP reference models were developed using dissimilar methodologies and described using different languages. Indeed, there was no global plan for them to be interoperable with each other. To subsequently achieve interoperability, it is necessary to develop additional methodologies to support their integration.

The EXP2OWL Translator (STEP to OWL transformation) was developed in the scope of the STEP-based Harmonization Famework (cf. Section 3.4.3). This tool was first described in Dutra et al. (2007). In the following sections, we are going to see its main features.

6.1.1 A Proposed Mapping for EXPRESS and OWL

We propose a mapping for the following EXPRESS statements (Figure 6.2).

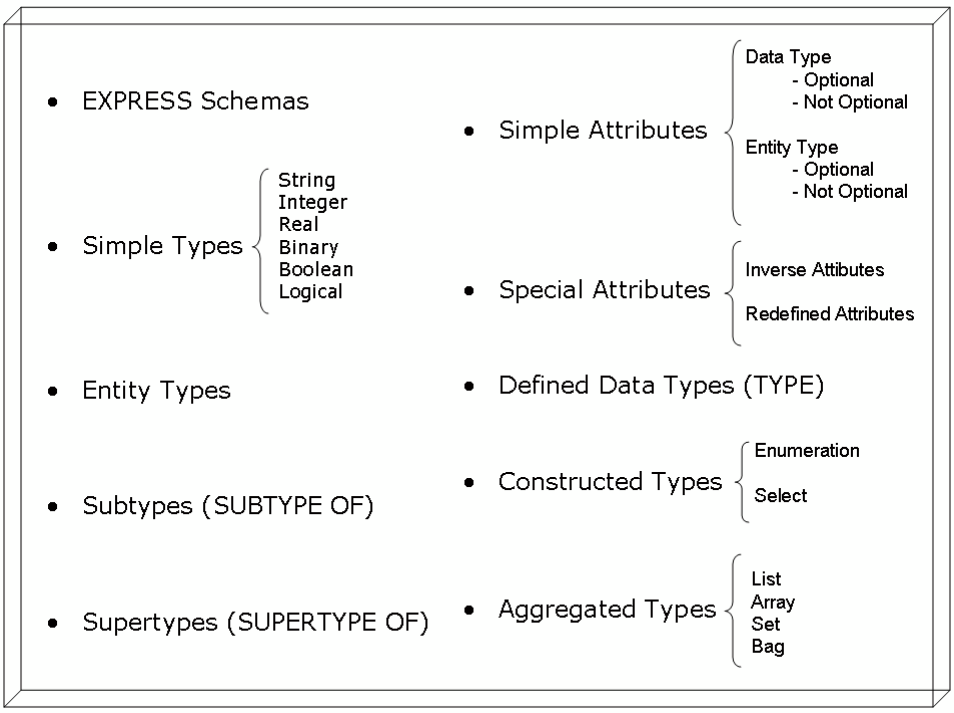

Figure 6.2 – EXPRESS Statements Mappable to OWL

6.1.2 EXPRESS Schemas and Simple Types

Each OWL file represents an ontology. An OWL file header extends the RDF file header, by aggregating URIs to the OWL vocabulary and to the ontology being described. In our approach, EXPRESS schemas were translated into OWL ontologies, by creating separated files to represent each one of them. So, a typical OWL file representing an EXPRESS schema named Fruit_schema – which uses definitions from Fruit_description schema – would be like this:

```
<rdf:RDF
 xmlns="http://www.uninova.pt/ontology/Fruit_schema#"
 xmlns:owl="http://www.w3.org/2002/07/owl#"
 xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
 xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
 xml:base="http://www.uninova.pt/ontology/Fruit_schema">
    <owl:Ontology rdf:about="">
    <owl:versionInfo>1.0</owl:versionInfo>
    <rdfs:comment>Fruit_schema</rdfs:comment>
    <owl:imports rdf:resource="Fruit_description.owl"/>
    </owl:Ontology>
</rdf:RDF>
```

The conversion of EXPRESS simple types (String, Integer, Real, Binary, Boolean, Logical) was direct, as they have equivalents in OWL (actually, XSD types). For example, String type was mapped into *xsd:string type* (Table 6.1).

EXPRESS	OWL
String	`xsd:string`
Integer	`xsd:integer`
Real	`xsd:float`
Binary	`xsd:hexBinary`
Boolean	`xsd:boolean`
Logical	`xsd:boolean`

Table 6.1 – Simple Types

The defined data types were mapped as shown in Table 6.2.

EXPRESS
TYPE Label = STRING;
OWL

`<owl:Class rdf:about="#Label">`
`  <owl:sameAs  rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>`
`</owl:Class>`

Table 6.2 – Defined Data Type

6.1.3 Entity Types, Attributes, and Inheritance Relationships

EXPRESS entities are used to define concepts from the real world which have properties that characterize them. In the entity relationship, they would be tables, but in OWL they are classes. By using this principle, we can map directly any entity as well as their subtypes and supertypes (taking advantage of the OWL classes inheritance). The OWL data property is used to represent EXPRESS simple attributes, and the OWL object property represents named attributes. The OWL cardinality is used according to the EXPRESS attribute's optional flag.

Some EXPRESS inherited attributes – from a supertype entity – can be renamed or retyped according to the user's needs. These redefined attributes are mapped using the OWL classes' specialization. Regarding EXPRESS inverse attributes, which are pointers to the relating entity, the OWL inverse property is available for the same purpose. Table 6.3 shows an example of EXPRESS entities mapped into OWL.

EXPRESS	
ENTITY Fruit; description : OPTIONAL STRING; END_ENTITY;	ENTITY Tree SUBTYPE OF (Thing); root : Root; END_ENTITY;

OWL

```
<owl:Class rdf:about="#Fruit">
 <rdfs:subClassOf>
   <owl:Restriction>
    <owl:minCardinality rdf:datatype=http://www.w3.org/2001/XMLSchema#int"> 0
    </owl:minCardinality>
     <owl:onProperty>
      <owl:DatatypeProperty rdf:ID="Fruit_description">
       <rdfs:domain rdf:resource="#Fruit"/>
       <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
      </owl:DatatypeProperty>
     </owl:onProperty>
    </owl:Restriction>
   </rdfs:subClassOf>
</owl:Class>

<owl:Class rdf:about="#Tree">
 <rdfs:subClassOf rdf:resource="#Thing"/>
 <rdfs:subClassOf>
  <owl:Restriction>
   <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"> 1
   </owl:cardinality>
   <owl:onProperty>
    <owl:ObjectProperty rdf:ID="Tree_root">
     <rdfs:range rdf:resource="#Root"/>
    </owl:ObjectProperty>
   </owl:onProperty>
  </owl:Restriction>
 </rdfs:subClassOf>
 </owl:Class>
```

Table 6.3 – Entities

In Table 6.4 we can see the mapping for the EXPRESS Supertype statement.

EXPRESS
ENTITY MarketFruit SUPERTYPE OF (ONEOF (Orange, Lemon) ANDOR ONEOF (Papaya, Banana) ANDOR Apple ANDOR Grape) SUBTYPE OF (Fruit) END_ENTITY;
OWL

```
<owl:Class rdf:about="#MarketFruit">

  <owl:equivalentClass>
  <owl:Class>
       <owl:unionOf rdf:parseType="Collection">
              <owl:Class rdf:about="#Orange"/>
              <owl:Class rdf:about="#Lemon"/>
       </owl:unionOf>
  <owl:Class>
  </owl:equivalentClass>

   <owl:equivalentClass>
   <owl:Class>
         <owl:unionOf rdf:parseType="Collection">
              <owl:Class rdf:about="#Papaya"/>
              <owl:Class rdf:about="#Banana"/>
       </owl:unionOf>
   </owl:Class>
   </owl:equivalentClass>

   <owl:equivalentClass>
   <owl:Class rdf:about="#Apple"/>
   </owl:equivalentClass>

   <owl:equivalentClass>
   <owl:Class rdf:about="#Grape"/>
   </owl:equivalentClass>

   <rdfs:subClassOf rdf:resource="#Fruit"/>

</owl:Class>
```

Table 6.4 – Supertypes

While Table 6.5 shows an example of mapping for EXPRESS redeclared attributes, Table 6.6 shows the mapping for EXPRESS inverse attributes.

EXPRESS	
ENTITY Fruit flavor: Flavor; END_ENTITY;	ENTITY SweetFruit SUBTYPE OF (Fruit); SELF\Fruit.flavor: SweetFlavor; END_ENTITY;

OWL

```
<owl:Class rdf:about="#Fruit">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:cardinality
rdf:datatype="http://www.w3.org/2001/XMLSchema#int">1</owl:cardinality>
      <owl:onProperty>
       <owl:ObjectProperty rdf:ID="Fruit_flavor">
       <rdfs:range rdf:resource="#Flavor"/>
       </owl:ObjectProperty>
      </owl:onProperty>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>

<owl:Class rdf:about="#SweetFruit">

  <rdfs:subClassOf rdf:resource="#Fruit"/>

  <rdfs:subClassOf>
    <owl:Restriction>
       <owl:onProperty>
         <owl:ObjectProperty rdf:about="#Fruit_flavor"/>
       </owl:onProperty>
       <owl:allValuesFrom rdf:resource="#SweetFlavor" rdf:type=
"http://www.w3.org/2002/07/owl#Class"/>
    </owl:Restriction>
  </rdfs:subClassOf>

</owl:Class>
```

Table 6.5 – Redeclared Attributes

EXPRESS
ENTITY Tree root: Root; INVERSE partOf: RootFOR root; END_ENTITY;

OWL

```
<owl:Class rdf:about="#Tree">

<rdfs:subClassOf>
  <owl:Restriction>
    <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int">1</owl:cardinality>
    <owl:onProperty>
       <owl:ObjectProperty rdf:ID="Tree_root">
       <rdfs:range rdf:resource="#Root"/>
       <owl:inverseOf rdf:resource="#Root_partOf"/>
       </owl:ObjectProperty>
    </owl:onProperty>
  </owl:Restriction>
</rdfs:subClassOf>

</owl:Class>
```

Table 6.6 – Inverse Attributes

6.1.4 Constructed Types

There are two kinds of constructed data types in EXPRESS: enumeration data types and select data types. Enumeration is a concept common to many other languages, and defines a set of names to be used in a domain. Regarding the select, it is a concept very characteristic of EXPRESS to define a data type that enables a choice among several named data types (STEP Part 11, 2002).

Enumeration (Table 6.7) and Select (Table 6.8) types were mapped through the use of the OWL clauses *owl:oneOf* and *owl:unionOf*, respectively. In the Select case, it was also necessary to assure the disjointness of each resulting class.

EXPRESS
TYPE red_fruits = ENUMERATION OF (Strawberry, Apple, Raspberry, Cherry); END_TYPE;
OWL

`<owl:Class rdf:about="#red_fruits">`
 	
`  <owl:oneOf rdf:parseType="Collection">`
`  <owl:Thing rdf:about="#Strawberry"/>`
`  <owl:Thing rdf:about="#Apple"/>`
`  <owl:Thing rdf:about="#Raspberry"/>`
`  <owl:Thing rdf:about="#Cherry"/>`
`  </owl:oneOf>`
 	
`</owl:Class>`

Table 6.7 – Enumeration

EXPRESS
TYPE Citrus_Fruit = SELECT (Orange, Lemon, Grapefruit); END_TYPE;
OWL

```
<owl:Class rdf:about="#Orange">
  <owl:disjointWith rdf:resource="#Lemon"/>
  <owl:disjointWith rdf:resource="#Grapefruit"/>
</owl:Class>

<owl:Class rdf:about="#Lemon">
  <owl:disjointWith df:resource="#Orange"/>
  <owl:disjointWith df:resource="#Grapefruit"/>
</owl:Class>

<owl:Class rdf:about="#Grapefruit">
  <owl:disjointWith rdf:resource="#Orange"/>
  <owl:disjointWith rdf:resource="#Lemon"/>
</owl:Class>

<owl:Class rdf:ID="Citrus_Fruit">
 <owl:equivalentClass>
  <owl:Class>
   <owl:unionOf rdf:parseType="Collection">
      <owl:Class rdf:about="#Orange"/>
      <owl:Class rdf:about="#Lemon"/>
      <owl:Class rdf:about="#Grapefruit"/>
   </owl:unionOf>
  </owl:Class>
 </owl:equivalentClass>
</owl:Class>
```

Table 6.8 – Select

6.1.5 Aggregated Types

In order to map the EXPRESS aggregated types, it was necessary to define an intermediate class, as there was no equivalent OWL structure to represent such types. Thus, EXPRESS List, Array, Set, and Bag were first represented in an OWL metaclass; the final class was set as a subclass of this metaclass. With a new class we could easily manage properties and restrictions of each type, without changing their initial definitions (Figure 6.3). Table 6.9 shows the mapping for the Set statement.

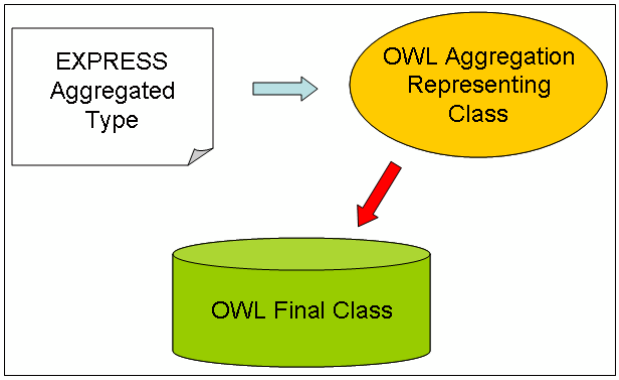

Figure 6.3 – Approach for Aggregated Types

EXPRESS

```
TYPE Orchard = SET [1:?] OF Tree;
END_TYPE;
```

OWL

```
<owl:Class rdf:ID="OWL_Set_Tree">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty>
        <owl:ObjectProperty rdf:ID="OWL_Set_belongTo_Tree"/>
      </owl:onProperty>
      <owl:allValuesFrom rdf:resource="#Tree"/>
    </owl:Restriction>
  </rdfs:subClassOf>

  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#OWL_Set_belongTo_Tree"/>
      <owl:minCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"> 1
      </owl:minCardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>

<owl:Class rdf:ID="Orchard">
  <owl:sameAs rdf:resource="#OWL_Set_Tree"/>
</owl:Class>
```

Table 6.9 – Set

Tables 6.10, 6.11 and 6.12 show mapping examples for the other aggregated types: Array, List, and Bag.

EXPRESS

```
ENTITY FruitMarket;

SummerFruits: ARRAY [6] OF Fruit;

END_ENTITY;
```

OWL

```
<owl:Class rdf:about="#OWL_Array_Size_6">
        <rdfs:subClassOf rdf:resource="#OWL_List"/>
        <rdfs:subClassOf>
        <owl:Restriction>
      <owl:maxCardinality
rdf:datatype="http://www.w3.org/2001/XMLSchema#int">6</owl:maxCardinality>
      <owl:onProperty rdf:resource="#OWL_List_item"/>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>

<owl:Class rdf:about="#FruitMarket">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:cardinality
rdf:datatype="http://www.w3.org/2001/XMLSchema#int">1</owl:cardinality>
      <owl:onProperty>
        <owl:ObjectProperty rdf:ID="FruitMarket_SummerFruits">
        <rdfs:range rdf:resource="#OWL_Array_Size_6"/>
        </owl:ObjectProperty>
      </owl:onProperty>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

Table 6.10 – Array

EXPRESS

```
ENTITY FruitMarket;

AvailableSummerFruits:
LIST OF Fruit;

END_ENTITY;
```

OWL

```
<owl:Class rdf:about="#OWL_List_Element">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:cardinality
rdf:datatype="http://www.w3.org/2001/XMLSchema#int">1</owl:cardinality>
      <owl:onProperty>
        <owl:DatatypeProperty rdf:ID="OWL_List_Element_index">
               <rdfs:domain rdf:resource="#OWL_List_Element"/>
               <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#int"/>
               </owl:DatatypeProperty>
      </owl:onProperty>
       </owl:Restriction>
  </rdfs:subClassOf>


  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:minCardinality
rdf:datatype="http://www.w3.org/2001/XMLSchema#int">0</owl:minCardinality>
      <owl:onProperty>
       <owl:ObjectProperty rdf:ID="OWL_List_Element_content">
       <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#Thing"/>
       </owl:ObjectProperty>
      </owl:onProperty>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>


<owl:Class rdf:about="#OWL_List">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:minCardinality
rdf:datatype="http://www.w3.org/2001/XMLSchema#int">0</owl:minCardinality>
      <owl:onProperty>
       <owl:ObjectProperty rdf:ID="OWL_List_item">
       <rdfs:range rdf:resource="#OWL_List_Element"/>
       </owl:ObjectProperty>
      </owl:onProperty>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>

<owl:Class rdf:about="#FruitMarket">

  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:cardinality
rdf:datatype="http://www.w3.org/2001/XMLSchema#int">1</owl:cardinality>
      <owl:onProperty>
       <owl:ObjectProperty rdf:ID="FruitMarket_AvailableSummerFruits">
       <rdfs:range rdf:resource="#OWL_List"/>
       </owl:ObjectProperty>
      </owl:onProperty>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

Table 6.11 – List

EXPRESS

```
ENTITY FruitMarket;

FruitSaladRecipe: BAG [1,?] OF Fruit;

END_ENTITY;
```

OWL

```
<owl:Class rdf:ID="BagOfFruits">
  <rdfs:subClassOf rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#List"/>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#first"/>
        <owl:allValuesFrom rdf:resource="#Fruit"/>
      </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>

<owl:Class rdf:ID="OWL_FruitBag"/>

<owl:ObjectProperty rdf:ID="OWL_FruitBag_items">
    <rdfs:domain rdf:resource="#OWL_FruitBag"/>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
    <rdfs:range rdf:resource="#BagOfFruits"/>
</owl:ObjectProperty>

<owl:Class rdf:about="#FruitMarket">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:cardinality df:datatype="http://www.w3.org/2001/XMLSchema#int">1</owl:cardinality>
      <owl:onProperty>
        <owl:ObjectProperty rdf:ID="FruitMarket_FruitSaladRecipe">
          <rdfs:range rdf:resource="#OWL_FruitBag"/>
        </owl:ObjectProperty>
      </owl:onProperty>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

Table 6.12 – Bag

6.1.6 EXPRESS Rules

The EXPRESS language contains a high level of expressiveness and uses constructs that are hard to map to other modeling languages. Among some of these constructs are rules, queries, functions, and constraints to attribute values.

Nevertheless, it was not possible to translate EXPRESS rules into OWL classes. Concerning the EXPRESS uniqueness rule, both languages have different ways to interpret this principle. While in EXPRESS a unique value is attributed to each object and understood by all involved actors, there is no such possibility in OWL, where infinite URIs may be set to represent the same object. Moreover, as OWL is a declarative language, it is not possible to use it to represent function statements. Finally, the mapping of EXPRESS domain rules (WHERE clause) was not possible either.

The morphism shown above is used within our collaborative architecture, in order to enable the input of requirements in EXPRESS format. However, it is important to stress that the different degrees of expressiveness of the referred languages prevent a full binding – e.g.

EXPRESS rules – thus sometimes causing a partial morphism. In those cases, the morphism may result in some loss of information.

6.1.7 EXP2OWL Prototype

Figure 6.4 shows a screenshot of the EXP2OWL prototype. It was built with the Eclipse SDK, running Java 1.6.0.01. The OWL plug-in version 2.2 for Protégé 3.2 was used. This prototype takes EXPRESS files (**.exp*) as input and gives OWL files (**.owl*) as output.

Figures 6.5 and 6.6 show some small pieces of code. The first one builds an enumeration class, which comprises the given elements. The other one builds and retrieves an OWL Array class.

Figure 6.4 – EXP2OWL Prototype

```
protected OWLNamedClass getOWLEnumeration(String modelName, String className, java.util.List items){

    OWLModel owlModel = getOWLModel(modelName);
    OWLNamedClass namedClass = getOWLClass(modelName, className);
    OWLEnumeratedClass enumClass = owlModel.createOWLEnumeratedClass();

    Iterator i = items.iterator();
    while (i.hasNext())
        enumClass.addOneOf(getOWLClass(modelName, (String)i.next()));

    namedClass.setDefinition(enumClass);

    return namedClass;
}
```

Figure 6.5 – Function getOWLEnumeration

```
protected OWLNamedClass getOWL_Array(String modelName, boolean singleType, String typeName, int cardMin, int cardMax){

    String owlArrayClassName;
    String strCardMax = (new Integer(cardMax)).toString();

    if (!strCardMax.equals("0"))
        owlArrayClassName = "OWL_Aray_" + strCardMax + "_" + typeName;
    else
        owlArrayClassName = "OWL_Aray_" + typeName;

    if (owlClasses.get(owlArrayClassName) != null)
        return (OWLNamedClass)owlClasses.get(owlArrayClassName);

    OWLModel owlModel = getOWLModel(modelName);
    OWLNamedClass owlListClass = getOWL_List(modelName, singleType, typeName, null);
    OWLObjectProperty owlListPpty = getOWLObjectProperty(modelName, owlListClass.getName(), typeName, "OWL_List_Element_content", true);

    OWLNamedClass owlArrayClass = owlModel.createOWLNamedSubclass(owlArrayClassName, owlListClass);
    OWLMinCardinality minCardinality = owlModel.createOWLMinCardinality(owlListPpty, cardMin);
    owlArrayClass.addSuperclass(minCardinality);
    if (!strCardMax.equals("0")){

        OWLMaxCardinality maxCardinality = owlModel.createOWLMaxCardinality(owlListPpty, cardMax);
        owlArrayClass.addSuperclass(maxCardinality);
    }

    owlClasses.put(owlArrayClassName, owlArrayClass);

    return owlArrayClass;
}
```

Figure 6.6 – Function getOWL_Array

Appendices A and B present two EXPRESS schemas and their corresponding OWL output, translated into the EXP2OWL prototype.

6.2 Ontology-Generic Method

A number of very general ontologies formalizing notions such as processes and events, time and space, physical objects, and so on, have been developed and some of them have become accepted standards. The explicit goal of these ontologies is to have domain-specific ontologies extend them, thus providing the grounding in common vocabulary for them. The common top-level or reference ontology is usually more general since it needs to encompass the top level for ontologies yet to be developed.

It is certainly helpful to have ontologies that we need to match to refer to the same upper ontology or to conform to the same reference ontology. However, we often do not have this ease and need to create mappings between ontologies that perhaps use the same specification language but do not have any vocabulary beyond the specification language in common. Some techniques have produced good results towards achieving such a scenario. Heuristic-

based approaches to ontology mapping are similar to heuristic-based approaches to matching database schemas and XML structures and use lexical and structural components of definitions to find correspondences. Ontology-based approaches often go further, exploiting semantics of relationships in ontologies, such as, for example, the semantics of the *subclass-of* or *part-of* relationships, attachment of a property to a class, domain and range definitions for properties, and so on.

In order to support the collaborative work within the development projects and to reduce the development time, there is a need for assistance in definition, structuring and generation of the knowledge. Instead of having one or more glossaries as text documents, which are usually not related with distributed data carriers, explicit definition of objects and terms within the manufacturing system and the relations between them is required. This will enable an unambiguous manipulation with the concepts, and will facilitate the consideration of only related data and information within the collaboration activities in design, modeling and simulation of manufacturing systems as a part of the digital factory. The following functionalities emerge for a framework for generation of knowledge, modeling and simulation of manufacturing systems:

- Embedding of domain knowledge within a single knowledge base.
- Inclusion of results from the daily work of project members involved in the design and development of new products and manufacturing systems.
- Generation of new knowledge through inference.
- Integration of generated knowledge into the knowledge base and therewith enrichment of the knowledge base.
- Facilitating simulation and evaluation of a manufacturing system and integration of simulation results back into the knowledge base.
- Extraction of data and information from the knowledge base for the users and project members in a format needed for daily project work (e.g. spreadsheets, process models, documents, drawings, etc.).

6.2.1 STEP-Based Product Model

The adoption of standards like STEP have established standardized ontologies and representations for artifacts, in particular application domains. Since different analysis tasks are likely to address different aspects of a product, a single view is unlikely to be sufficient. Furthermore, a monolithic representation may also hinder reuse, since irrelevant attributes may affect the compatibility test. Hence, abstractions and compatibility criteria between abstract models must be defined with respect to particular analysis goals.

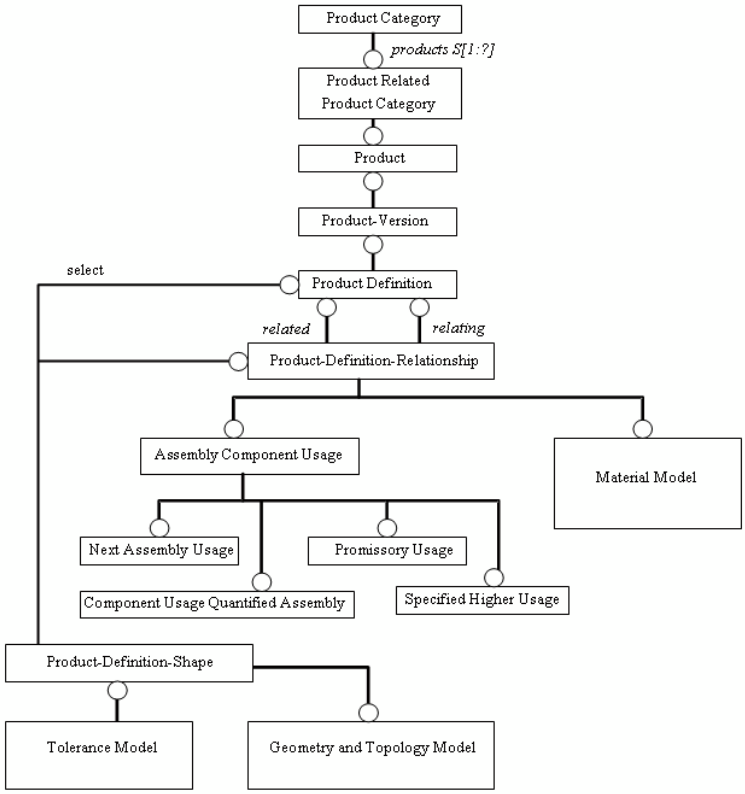

Figure 6.7 – STEP-based Product Ontology

STEP/EXPRESS has been identified as a suitable framework, since it provides an expressive language that can capture not only concrete representations of artifacts through application protocols (AP), but also formalize the aggregation operators to transform concrete representations into abstract views and to define compatibility between abstract views. Figure 6.7 shows an example of a STEP-based product ontology.

The powerful representation in EXPRESS leads to simpler development through abstract specification and meta programming. Meta-models can be used to consolidate different models and compatibility criteria, as they integrate artifact and process models with the domain models of the STEP standard. EXPRESS is a formalism for design artifact representation as well as for specification of concrete and abstract artifact properties and their mappings. We can directly access the vast amount of domain knowledge already formalized in STEP, by also expressing the engineering model in EXPRESS. Furthermore, the models on the meta-level form a frame of reference for all concrete models and thereby provide the means to express the relations between properties on different levels of abstraction.

6.2.2 FBS Ontology

The modeling with the FBS ontology starts from collecting the users' requirements and transforming them into formal specifications. These specifications are subsequently used to describe the product functions, behaviors and structures (Figure 6.8). We have sometimes used the FBS ontology representation in our architecture. In Chapter 8, we are going to present a scenario that uses this kind of representation.

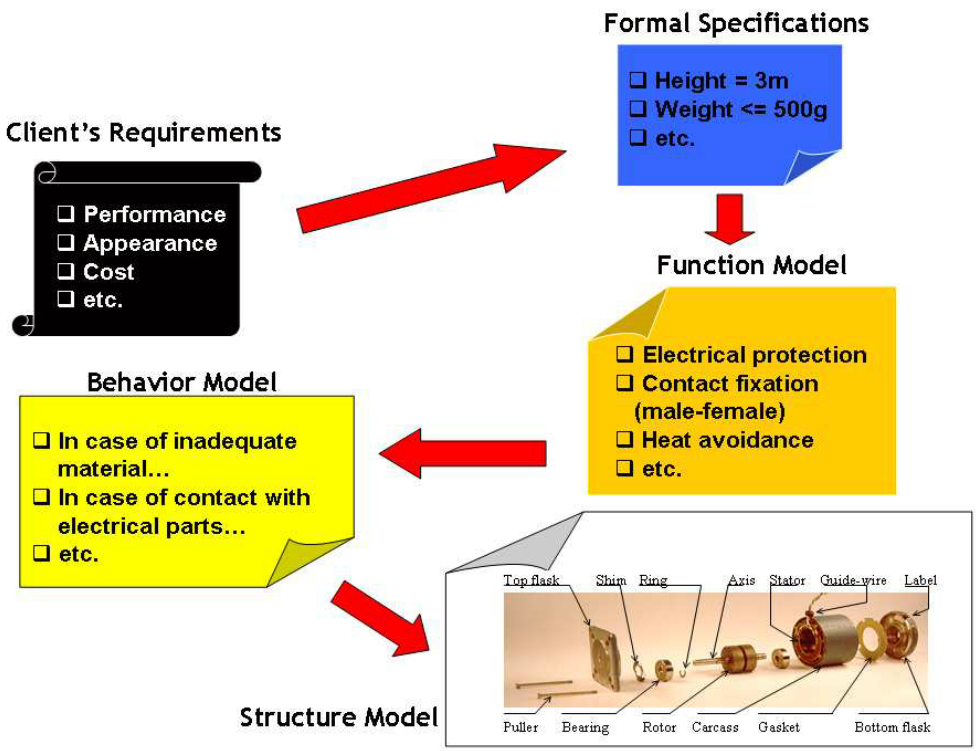

Figure 6.8 – FBS Ontology

6.2.3 Ontology Reasoning

How human observers reason about and compare the similarity of artifacts is a main concern in understanding the perception of design. Psychologists and cognitive scientists have pursued questions of how people classify objects, form concepts, solve problems and make decisions based on perceived similarity. Much of this debate has surrounded the investigation of the variables used for reasoning and the different forms of reasoning.

Due to the nature of the environments that most ontologies operate in (the Semantic Web, for example), it is more likely that we will need more than one ontology to achieve knowledge sharing. It is increasingly unlikely that a single ontology will both adequately capture the domain in question and also be consensual among all interested parties. Strict knowledge engineering practice is difficult to enforce when dealing with outsourced ontologies. Syntactic compliance with Semantic Web standards – like OWL – is not enough to guarantee that our inferences will make sense and that automated reasoning will be possible. It is not uncommon to find subtle differences in meaning between any two ontologies even if they represent the same domain and are encoded in the same formalism. So, a reasoning process should be undertaken to fill this gap (Dutra et al., 2007b).

As a prerequisite to solve the ontology interoperability issue, it is vital to understand how varied ontological concepts can be and in which ways ontology and semantic mismatches take place, which impede achieving seamless interoperability. Semantic mismatches can be interpreted from perspectives such as knowledge elicitation, databases and knowledge representation.

6.2.4 Constructing a Generic Ontology

Domain conceptualization is critical to ontology construction. The goals of domain conceptualization are to identify and define concepts in the application domain and to specify relationships among all concepts. In practice, capturing domain concepts and their relations is a bottleneck for constructing an ontology. The main reason is that knowledge engineers usually obtain domain concepts from dictionaries, books, standards, manuals, expertise, and so on. This work is time and cost consuming. Additionally, if knowledge engineers have different views about the same concept or if inconsistency arises in knowledge sources, ontology inconsistency interfering with the expansion, sharing and application of the ontology becomes inevitable. With the development of machine learning, it is possible to obtain concepts and their relations automatically. Classification and clustering techniques are also applied to capture concepts and their relations efficiently as well as the efficient utilization of existing knowledge (reuse).

The composition of a design team often changes with every new project, i.e. new members join the team while others leave. As a consequence, the expertise of the team is constantly reformed. In most cases, however, experts do not have to start building the team expertise from scratch. They can use their experience gained from being involved in previous teams to quickly integrate themselves and other experts into the new team. Expertise of teams can thus be viewed as emerging from the ability of individual experts to interact with one another using their generalized and specialized knowledge acquired over a series of team interactions. Developers usually cooperate with domain experts to perform knowledge engineering tasks such as atom identification, solution analysis and rule representation.

To illustrate, let us consider the scenario in which a designer uses a relational database model as the generic ontology to design a pen. This model comprises two tables – Product and Composition – structured as follows:

```
Product (String pID, String pName);
Composition (String pIDFather, String pIDSon);
```

The designer then fills both tables with the following records (Tables 6.13 and 6.14):

Product	
pID	**pName**
P1	Pen
P2	Hood
P3	Body
P4	Tube
P5	Refill
P6	Plug
P7	Writing Head
P8	Reservoir
P9	Head Support
P10	Head
P11	Ball

Table 6.13 – "Product" Table

Composition	
pIDFather	**pIDSon**
P1	P2
P1	P3
P3	P4
P3	P5
P3	P6
P5	P7
P7	P9
P7	P10
P7	P11

Table 6.14 – "Composition" Table

The resulting product model, instantiated from the given generic ontology is shown in Figure 6.9. This model also aggregates the graph-based representation, since we have defined a hierarchical structure to represent a product called "Pen." Each tree node is an instance of the generic ontology, as well as a sub-product of the final model. Each node has its own definitions and constraints, as defined by the generic ontology.

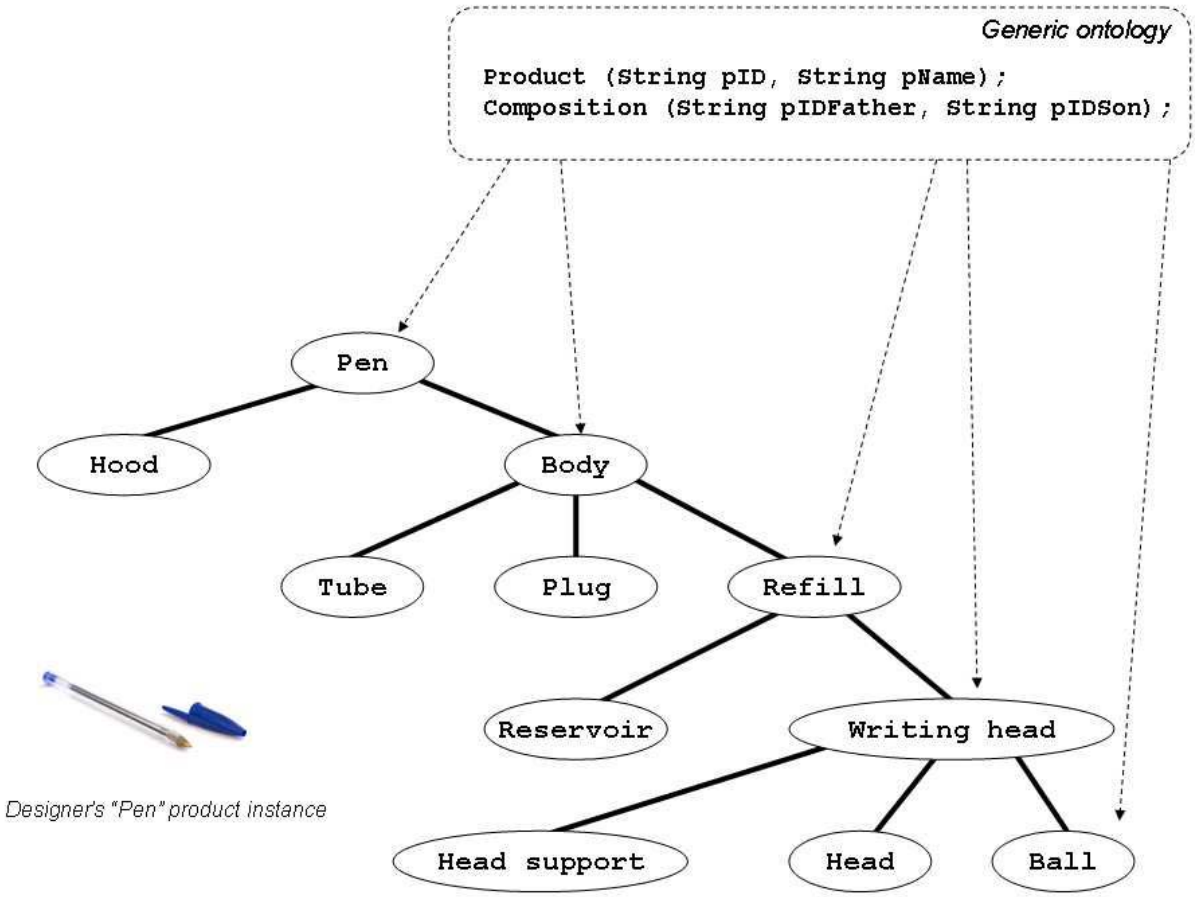

Figure 6.9 – Designer's "Pen" Product Instance

Once a common (or domain) ontology is agreed upon by the designers, they have a common background in which to start sharing knowledge. The example presented above takes a single

ontology to create its sub-product instances, but we can extrapolate it considering that there could be several generic ontologies available to model the product, instead of just one. In that case, the harmonization method should be used.

6.3 Harmonization Method

In order to support different kinds of knowledge representation to be integrated into our architecture, we propose the harmonization method. As shown in Figure 6.10, the final goal of the harmonization is to provide a common model to be used by all the design participants. The harmonization method is a stage undertaken before defining the generic ontology for the project. It requires earlier interaction between the ontology experts and the designers.

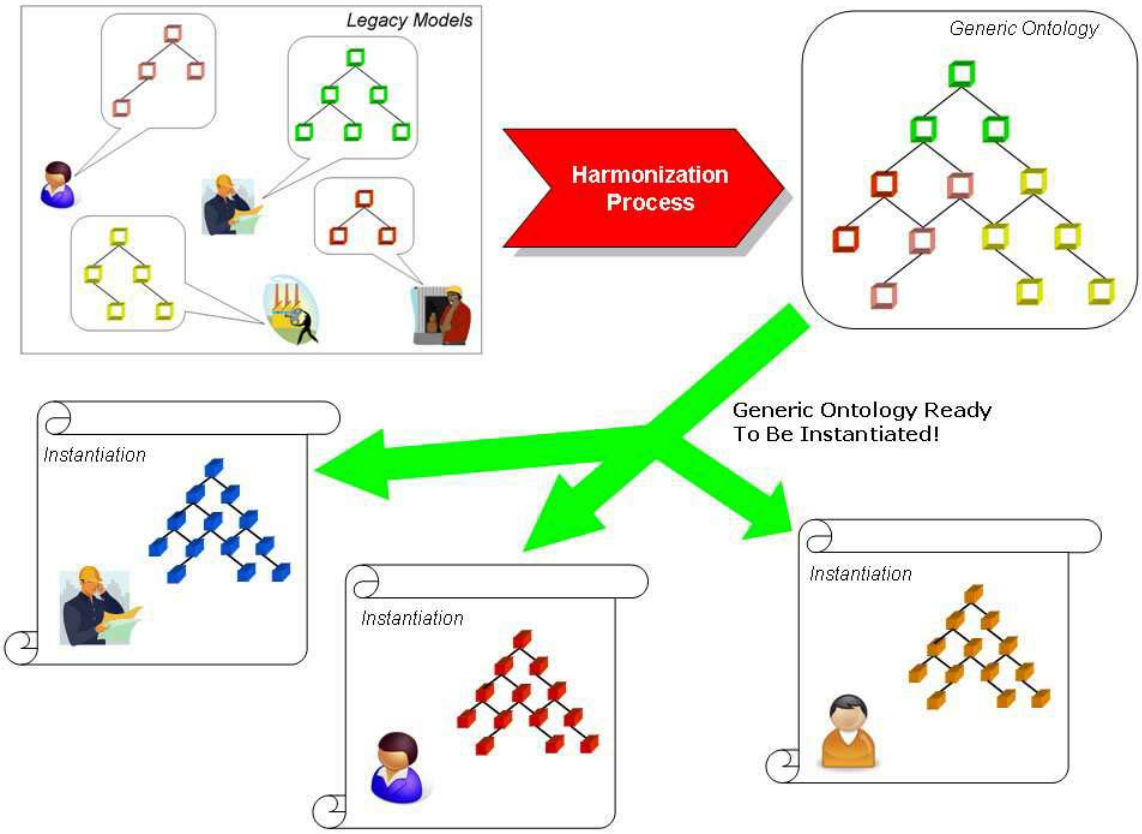

Figure 6.10 – Harmonization Process

Our harmonization method intends to combine the knowledge of different formalisms in order to improve representational adequacy and deductive power. A hybrid specialized ontology system needs to be able to interoperate with the enterprise's proprietary ontologies. There are generally three approaches for combining such distributed heterogeneous ontologies:

- Ontology Inclusion, in which the source ontology is simply included within the target ontology;

- Ontology Merging, using mediators;

- Ontology Mapping, in which a part of the source ontology is related to the target ontology's entities.

We have chosen the ontology merging and mapping as the approach more adequate to combine the engaged ontologies in the system, since the focus is maintaining the private ontologies, but building a new one to be their reference in the domain.

Now, consider the "Pen" example presented in the previous section and assume that another participant is integrated into the design project. This new participant works at the assembly line, hence having manufacturing viewpoints, and is used to working with STEP-based representation models instead of using the relational database model. The ontology used by this manufacturer is the following one, written in EXPRESS:

```
ENTITY Product;
      Id: INTEGER;
      Name: STRING;
      Composed_of: BAG [0:?] OF Product;
      INVERSE
            Final: SET [1:?] OF Product FOR Composed_of;
END_ENTITY;
```

The instantiaton of the manufacturer's ontology will produce the following output (the '#' symbols represent child nodes and the '$' symbols represent leaves):

```
ISO 10303_21;
Begin data;
#1 = Product ("Pen", (#2, #3, #4, #5));
#2 = Product ("Hood", $);
#3 = Product ("Plug", $);
#4 = Product ("Cylindrical Tube", $);
#5 = Product ("Cartridge", (#6, #7));
#6 = Product ("Repository", $);
#7 = Product ("Point", (#8, #9));
#8 = Product ("Ball", $);
#9 = Product ("Cone", $);
End data;
End ISO 10303_21;
```

The graph-based representation of the manufacturer's model is presented in Figure 6.11.

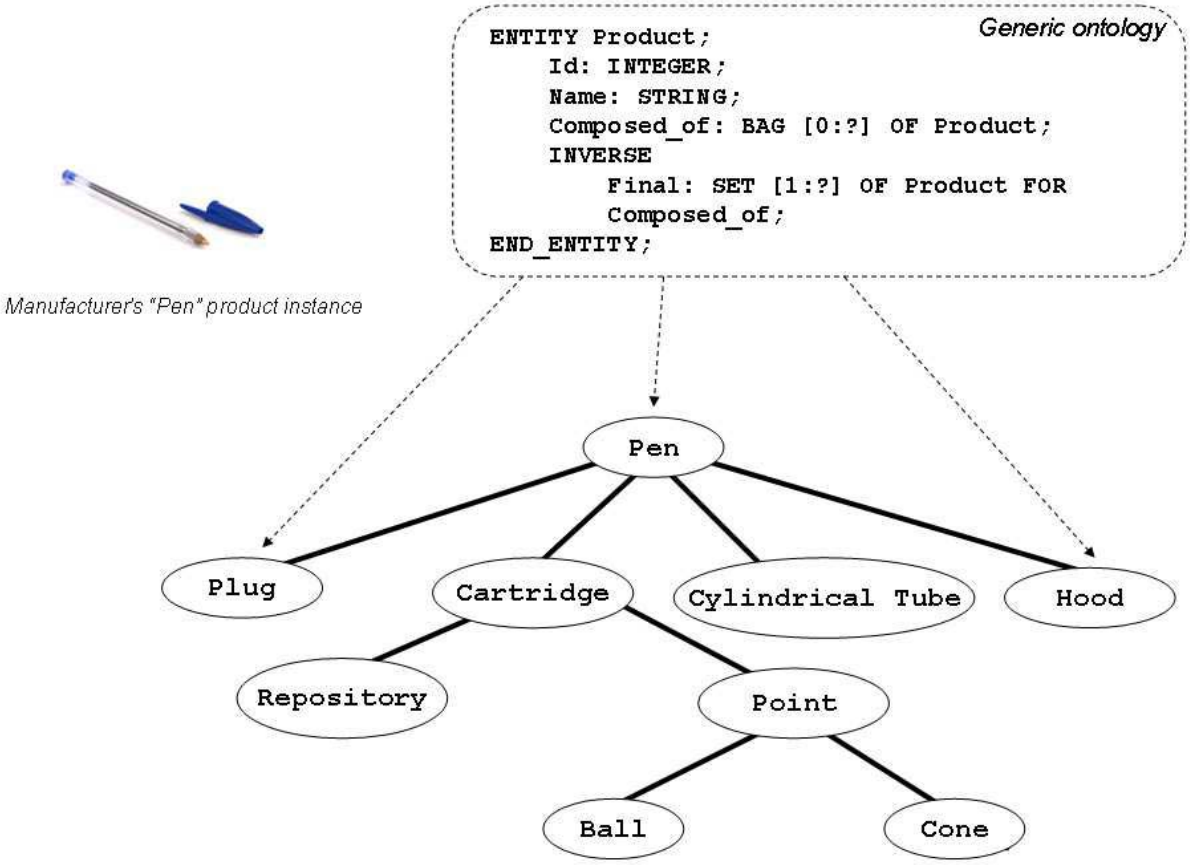

Figure 6.11 – Manufacturer's "Pen" Product Instance

At the end of the formalization process, both the designer's and manufacturer's models are expressed in OWL. Their contents, however, remain represented in different ways, as both experts do not share the same viewpoint on the product. To overcome this issue, a harmonization process is undertaken. An ontology expert, designated by the project manager, starts brainstorming with the designer and the manufacturer in order to achieve a common model, which should represent the merging of the two ontologies without loss of information. They decide then to carry out a lexical-based mapping of ontologies. A correspondence table is built in order to provide the automatic exchange of data/information without semantic loss. Table 6.15 shows the correspondence table agreed upon by the ontology expert, the designer, and the manufacturer.

Table 6.15 also shows the generic ontology, resulting from the merging of the two previous existing ontologies. This generic ontology is the result of the harmonization process, where the participants agreed to standardize the ontologies' terms and structures of composition.

Designer's Viewpoint		Manufacturer's Viewpoint		Generic Ontology	
1	Ball	1	Ball	1	Ball
2 3	Head Head Support	2	Cone	2 3	Head Head Support
4	Writing Head	3	Point	4	Writing Head
5	Reservoir	4	Repository	5	Repository
6	Refill	5	Cartridge	6	Cartridge
7	Plug	6	Plug	7	Plug
8	Tube	7	Cylindrical Tube	8	Tube
9	Body	5 6 7	Cartridge Plug Cylindrical Tube	6 7 8	Cartridge Plug Tube
10	Hood	8	Hood	9	Hood
11	Pen	9	Pen	10	Pen

Table 6.15 – Correspondence Table for the "Pen" Design Project

After the mapping, the resulting OWL-based ontology is enriched with some other attributes, such as geometrical (shape, size, etc.), aesthetical (color), and structural (raw materials) ones, among others. This generic ontology produced during the harmonization process is consequently ready to be instantiated by the participants of the "Pen" project. Figure 6.12 shows an example of the resulting instantiation.

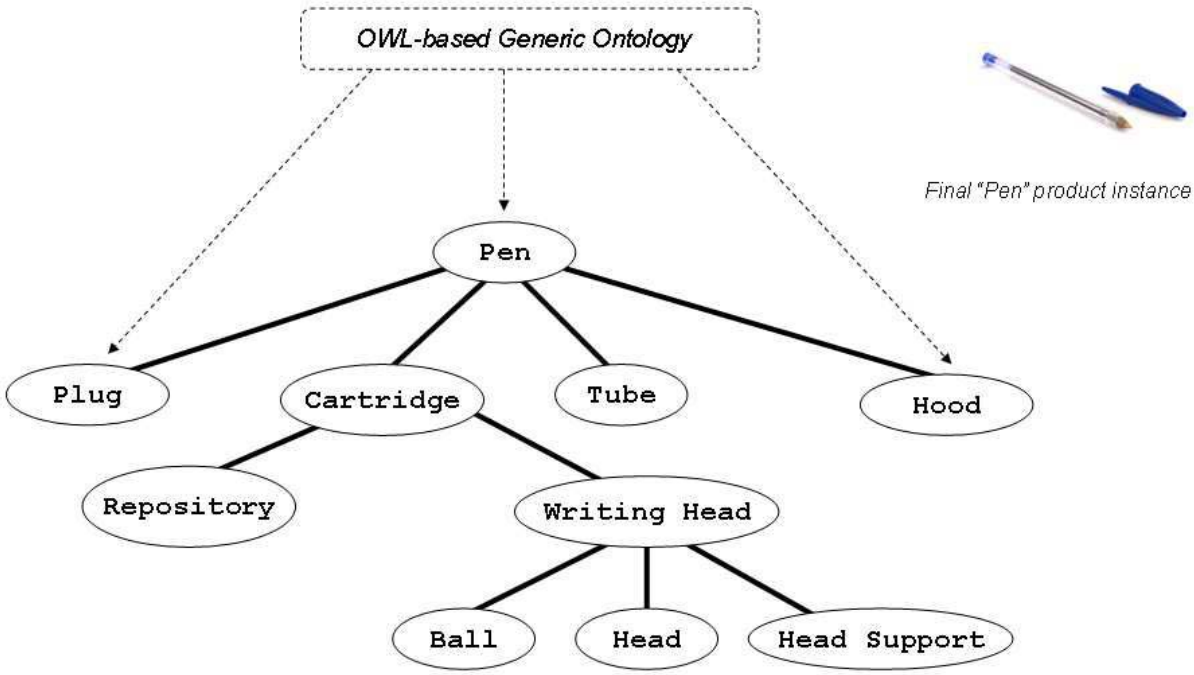

Figure 6.12 – Final "Pen" Product Instance

6.4 Applying an Inference Engine on a Mechanical Piece Model

The formalization and the harmonization processes are very effective strategies to mitigate collaborative conflicts. We can say that both approaches eliminate most of the conflicts that arise in the early stages of design. There are, however, other conflicts that continue to occur, even when all the designers are already working with the same expressing format and the same model for content representation. In that case, an ontology reasoning must be undertaken to detect the remaining inconsistencies.

In this section, we present an example of how this reasoning is done. A simple mechanical piece is taken as an example. Typically, the harmonization of the two ontologies shown below would be enough to avoid inconsistencies. Nevertheless, we consider, for illustrative purposes only, that they have already been harmonized.

This example uses the Protégé ontology-editor tool (Noy et al., 2001) to display the connector representations, and the Java libraries of Pellet (Sirin et al., 2007) and those of the OWL API (Horridge et al., 2007) to implement both the process of consistency verification and the functionality for ontology debugging.

6.4.1 Description Logics-Based Reasoning

A significant advantage of using OWL (DL) to represent knowledge is the capacity of reasoning that it provides. In comparison with the simple SQL requests from classical databases, responding to a request in a Description Logics(DL)-based system entails carrying out a more complex logical reasoning to check the presence of information, because several interpretations should frequently be considered. Hence, whereas the absence of information in a classical database conveys a negative message, in a DL-based system, it is interpreted as purely a lack of knowledge. A reasoner processes a subsumption test to determine the class hierarchy, among other ontology concepts. A reasoner also lets us check the consistency of an ontology at the same time as it is being constructed.

A typical ontology is composed of a taxonomy of classes and a set of inference rules. A rule describes a conclusion that can be drawn from a condition, an assertion that is processed by a reasoner, or an inference that is generated from a pre-defined rule by an inference engine. By using rules, the inference engines are able to deduce new knowledge, which is generated from previously existing knowledge. In this work, we are interested in reasoning DL-based ontologies, such as those represented in OWL DL.

Formally speaking, a concept represents a set of individuals with common characteristics, and a role represents a binary relation between individuals. Whereas a concept corresponds to a generic entity, an individual – or concept instance – corresponds to a particular entity. Concepts and roles are defined by structured descriptions, elaborated from a certain number of description constructors, namely intersection ($\sqcap$), union ($\sqcup$), negation (), and quantifiers ($\forall$, $\exists$). These constructors allow the definition of new complex concepts and roles from simpler ones.

Besides, we use the classical view of concept representation, which states that the features representing a concept are singly *necessary* and jointly *sufficient* to define a concept. Concepts can be primitives or defined. A primitive – or partial – concept ($C \sqsubseteq D$) works as the basis for constructing defined concepts. Its description only contains *necessary* conditions. A *necessary* condition implies that, if an individual is part of a class, it must satisfy this condition. On the other hand, a concept is said to be defined – or complete – ($C \equiv D$) if its description contains at least a set of *necessary* and *sufficient* conditions.

Intuitively, a feature φ is *necessary* for the definition of E *if and only if* (*iff*) the existence of E implies the existence of φ. More formally, let us assume a feature φ. We define a set Φ consisting of all items of the domain that contain the feature φ. Then, φ is necessary for the representation of the concept E *iff* $E^I \subseteq \Phi$. Now let us assume that $\{\Phi_1, \ldots, \Phi_n\}$ represents the set of concepts corresponding to the features $\varphi_1, \ldots, \varphi_n$, respectively. Then if $\{\Phi_1 \cap \ldots \cap \Phi_n\} \subseteq E^I$ we say that $\varphi_1, \ldots, \varphi_n$ are *sufficient* for the definition of E. Thus, we can easily derive the universal set of equations corresponding to the above notions of *sufficient* and *necessary* conditions for the representation of concepts. The existence of these conditions is enough to determine that any individual satisfying them is either an instance or an extension of the defined class. But it is important to remark that reasoners can only classify defined classes.

In DL-based knowledge representation systems, different types of reasoning can be done. In Baader et al. (2003), four types of inference can be distinguished at the therminological level (*TBox*):

- Subsumptiom: A concept C subsumes a concept D, or D is subsumed by C, $D \sqsubseteq C$, if C is more general than D, i.e. the set of individuals represented by C contains the set of individuals represented by D. That is, for each interpretation I, $D^I \subseteq C^I$.
- Equivalence: A concept C is equivalent to a concept D *iff* for each interpretation I, $C^I = D^I$. This can be interpreted by a bi-directional subsumption, i.e. $C^I \subseteq D^I$ and $D^I \subseteq C^I$.
- Satisfiability: A concept C is satisfiable if there is at least one interpretation I, so that $C^I \models \top$. When modeling new concepts, it is particularly useful to check if the new concept is coherent with the knowledge base.
- Disjunction: Two concepts C and D are disjoint *iff* for each interpretation of I, $C^I \cap D^I \models \bot$.

6.4.2 Graph-based Representation of Concepts

Graphs are a powerful and efficient technique for knowledge representation. Labeled graphs are useful for knowledge representation in general. Graphs – as mathematical objects – represent knowledge in a natural way, thereby allowing the construction of effective algorithms. Until recently, the major technique used in computer science for representing the semantic relationships between objects in a data structure was to use a graph technique known as semantic networks. There have been many attempts to formalize and standardize these graphical knowledge representation schemes, but probably none has been as extensive and comprehensive in recent times as conceptual graphs. The major use of conceptual graphs is in

representing the relationships between concepts in a system. Although their origins are different, conceptual graphs can be thought of as a formalization and extension of semantic networks. They are labeled graphs with two types of nodes: concepts, which represent objects, entities or ideas, and relation nodes, which represent relations between the concepts.

6.4.3 Defining Axioms

OWL DL contains a rich set of constructors, which provide the definition of concept axioms, relations, and instances. For instance, axioms of transitivity, inverse relation, and cardinality can be represented as *owl:TransitiveProperty*, *owl:inverseOf*, and *owl:Cardinality*, respectively. OWL also provides constructors for correspondence between different entities, e.g., it is possible to define equivalence axioms between concepts, via the equivalence constructor *owl:equivalentClass*, or between properties, via *owl:equivalentProperty*. In the same way, subsumption relations between concepts or properties can be defined by using *rdfs:subClassOf* or *owl:subPropertyOf*, respectively.

In the DL context, an axiom is considered as a reasoning hypothesis, which provides knowledge into a data model in order to allow machine interpretation for the semantics. This comprises the definition of subsumption or equivalence relations between classes or properties, as well as some property characteristics, such as transitivity. This axiomatization process makes the ontology semantically richer. It also makes it possible for us to call the inference services.

The following descriptions illustrate some examples of ontology axioms:

- Axiom 1: Let us consider that the *hasPrevious* property has been defined to infer the predecessors of an element that makes up part of some list. This property is defined as the inverse property of *hasNext*.

$$hasPrevious \equiv (hasNext)^{-}$$

- Axiom 2: We consider now the existence of the *hasSubPart* property, which is transitive.

$$Tr\ (hasSubPart)$$

- Axiom 3: Here, let us consider that the *Onto1:SoftProduct* concept is defined as a subclass of the *Onto1:Product*. It also has at least one object property *Onto1:hasCharacteristic* with the *Onto1:Soft* concept.

$$Onto1{:}SoftProduct \equiv Onto1{:}Product \sqcap (\exists\ Onto1{:}hasCharacteristics\ Onto1{:}Soft)$$

6.4.4 Modeling a Small Gadget

Conflicts regarding knowledge representation may be solved by applying services from an inference engine. This inference engine is based on the description logics. Our system includes a consistency verification process and a function for ontology debugging that apply inference engine services. The consistency checking process triggers the inference engine, which then checks the ontologies for contradictions. If the inference engine finds unsatisfiable concepts, then a conflicting situation is detected and the concerned parties are notified. Usually these parties are those who have published the conflicting ontology instances into the public workspace. The ontology debug 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.

The collaborative design of a small gadget – an electrical connector – is taken as an example for this scenario. We consider that two different points of view on the same product are going to be published into the public workspace. The first designer thus publishes his ontology instance into the shared workspace.

After the publication, all the other design project members are able to access this published instance. Figure 6.13 shows the main ontological classes and the compositional properties for the published instance, while Figure 6.14 shows the definition of the proposed "Connector" concept in Protégé.

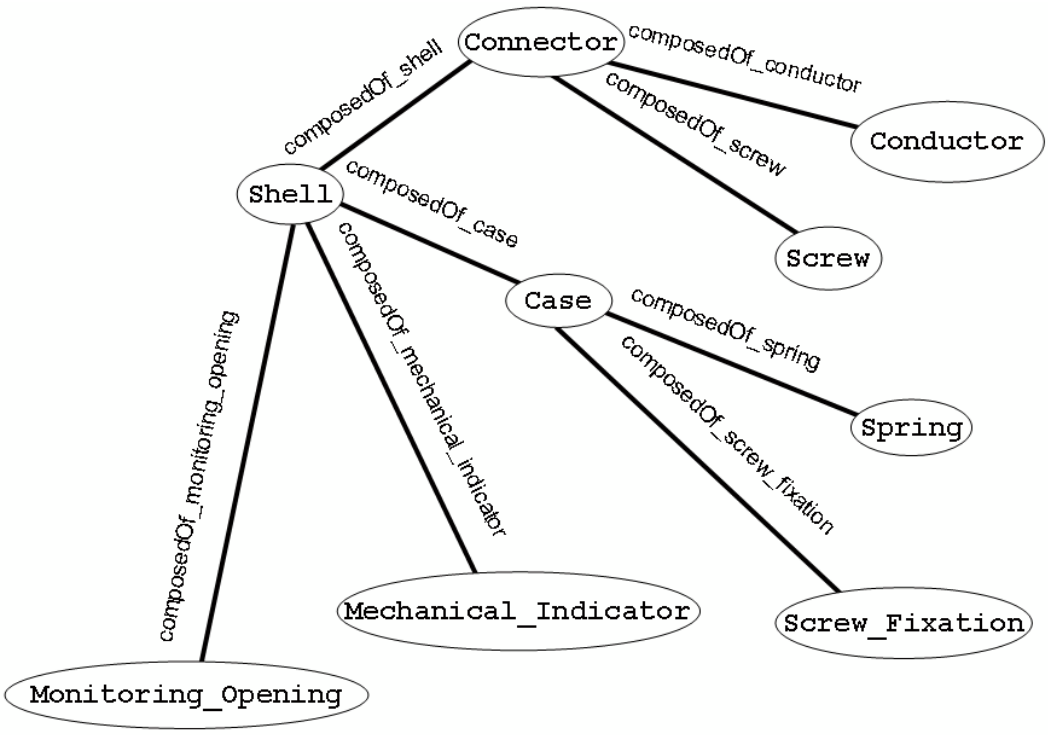

Figure 6.13 – Graphical Representation of Connector 1, Arcs Representing Compositional Properties

According to this ontology instance, the “Connector” concept is composed of the following product 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 6.14 –Representation of the Connector 1 Classes in Protégé

Figure 6.15 shows the object property hierarchy for Connector 1 in Protégé.

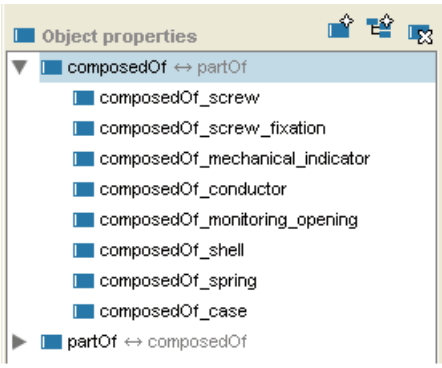

Figure 6.15 – Properties Defined for Connector 1

Further on, another designer publishes his own ontology instance for the same product (cf. Figure 6.16). This time, the “Connector” concept is composed of the concepts “Cable”, “Screw”, and “Body.”

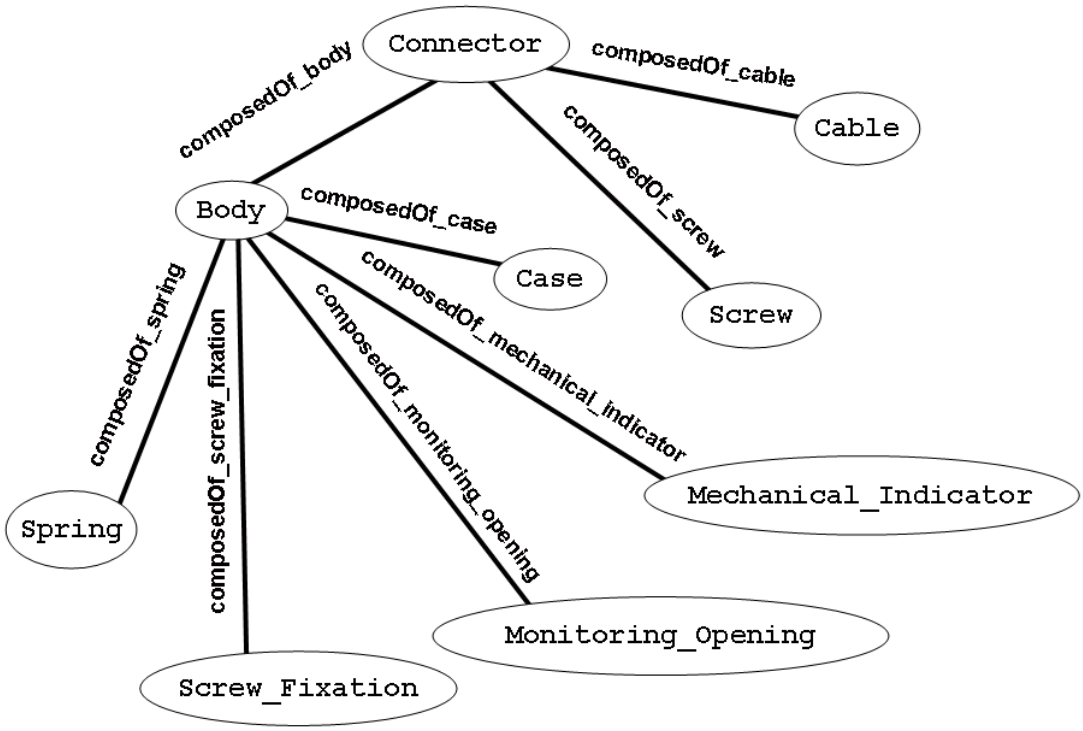

Figure 6.16 – Graphical Representation of Connector 2, Arcs Representing Compositional Properties

Figure 6.17 shows the *necessary* and *sufficient* definition for Connector 2, as shown in the Protégé tool.

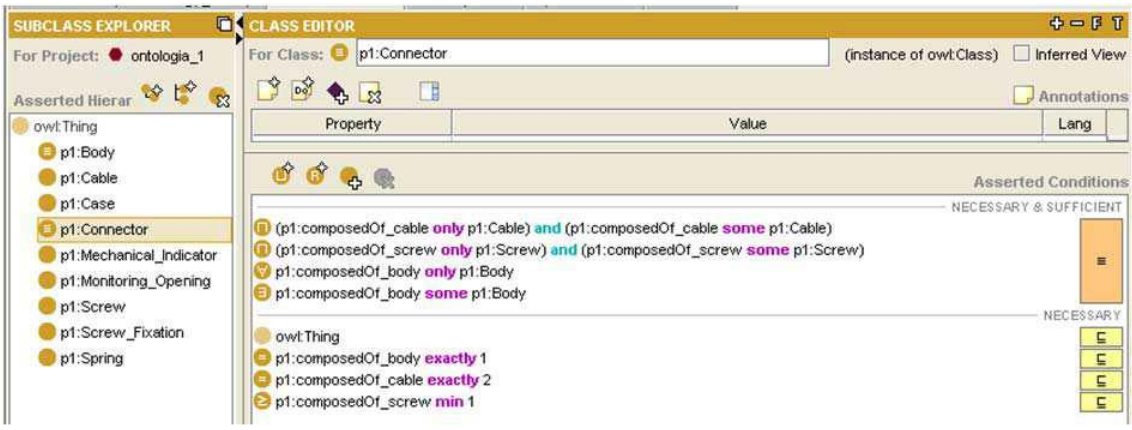

Figure 6.17 – Representation of the Connector 2 Classes in Protégé

Let us consider that the second designer also publishes information stating that the "Cable" concept is *equivalent* to the "Conductor" concept on the ontology model previously published by the first designer. The existence of two ontologies representing the same product design in the shared workspace triggers the process of consistency checking that calls the consistency checking service of the inference engine. In the present case, both ontologies are inconsistent (cf. Figure 6.18).

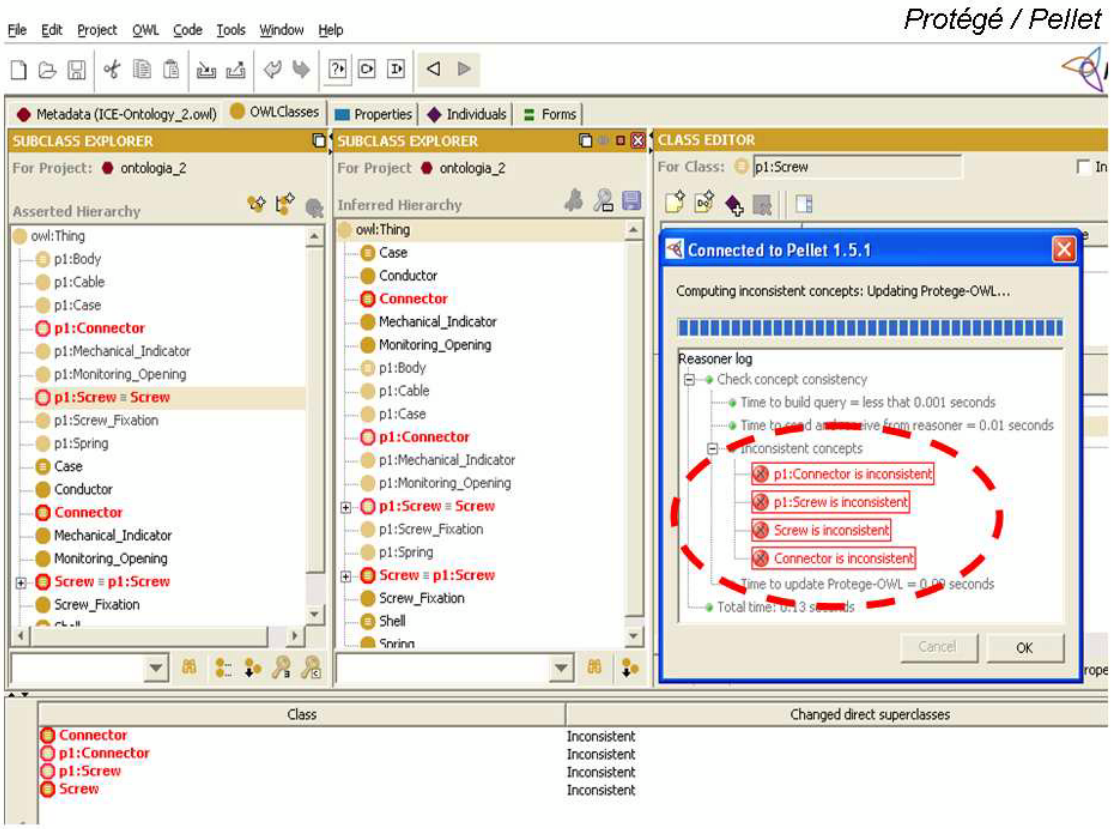

Figure 6.18 – Ontology Inconsistencies Detected with Protégé and Pellet

This has occurred because both concepts are disjoint. Besides, because "Cable" and "Conductor" have been assigned as *equivalent* concepts, incoherence was detected during the integration of the models. The SWOOP editor tool (Kalyanpur et al., 2006) displays the ontology debugging process. In Figure 6.19, we can see how the co-existence of axioms 2 and 3 has produced the incoherence. As the "Conductor" concept is part of the "Connector" concept definition, the previous incoherence brings about the next one, signaled by axiom 1. At this moment, the conflict resolution process should be started.

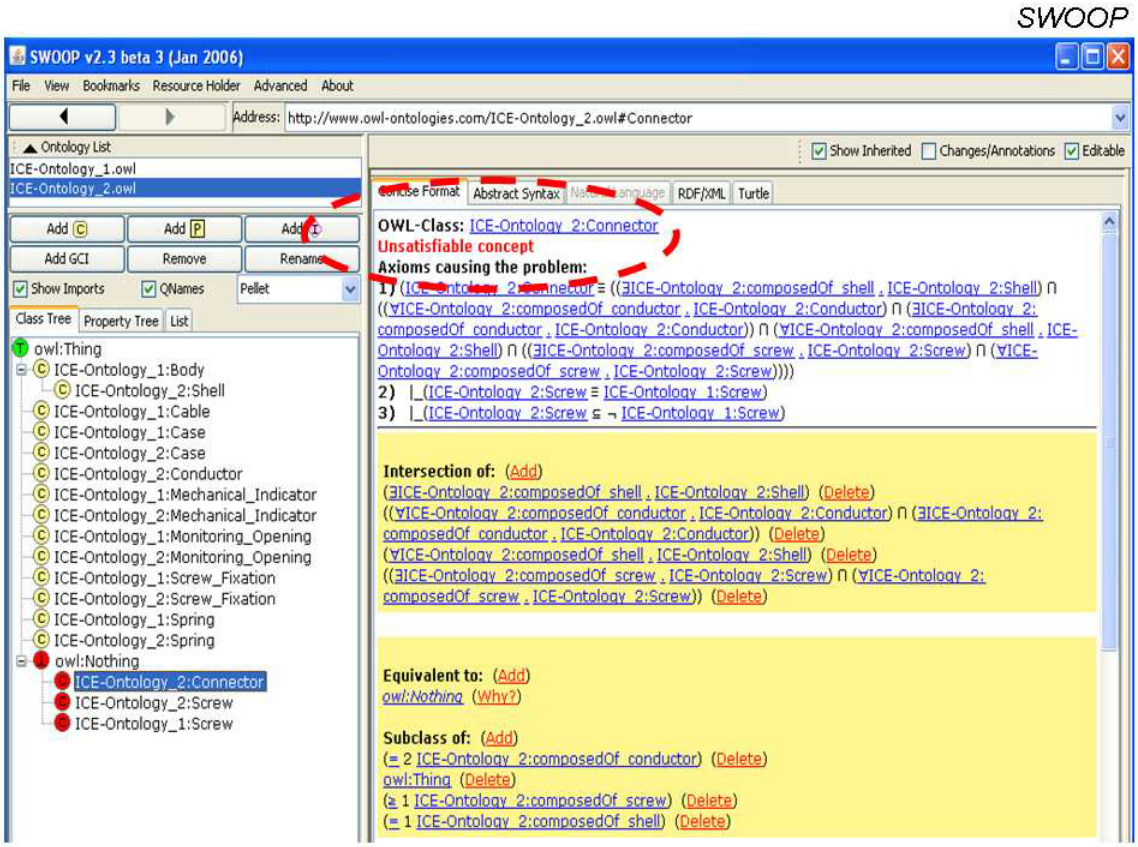

Figure 6.19 – Ontology Axioms that Caused the Inconsistency

Appendices C and D present the OWL-based representations for both Connectors 1 and 2.

6.5 Summary

The use of ontology modeling in collaborative design has proved to be a prominent approach to detect conflicts in early-stage design. Most of the required data and information in the early project phase are informal and based on the experience from already existing products and manufacturing systems. Modeling expertise as ontology models permits us to identify conflicting axioms through the description logics reasoning of ontologies. Besides, representing knowledge in OWL offers a reasonable trade-off between expressibility and decidability, which 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. This kind of tool lets us automate most of the conflict detection process and, hence, to diminish the remaining situations to be treated in an eventual negotiation process among the designers.

In this chapter, we have presented an ontology-based approach to deal with conflicts in collaborative design. This approach relies first on the formalization of knowledge, in a way that it is computational representable and decidable. The use of widespread standards – like XML, OWL, and STEP –, along with the proposal for interoperable exchanging between them, provides us with a very powerful method for dealing with collaborative conflicts.

Moreover, providing the designers with an automated approach for conflict resolution, to be done before the negotiation process, takes another step towards achieving the reduction of time and cost of the collaborative design process, along with the already cited synchronous and generic approaches. The APIs available for ontology reasoning – like the Protégé OWL API and Pellet – let us integrate all the inference engine functionality into the source code of our architecture. In the next chapter, we are going to present our proposed method for conflict resolution.

Chapter 7

7 The Conflict Resolution Method

Although in most design processes, coordination entails clear communication between designers, the real reason for this coordination is not for communication but for resolving dependencies between product data. The design process is usually constraint oriented, and comprises many interdependent parts. Designers must consider not only the functional requirements of a product but other aspects such as geometrical, behavioral, and structural, among others. Each of these has its own set of constraints which may contain conflicting or unsatisfied requirements and designers cannot always oversee the various alternatives and constraints. Then, due to the multi-actor interaction, conflicts emerge from disagreements between designers about incompatible and interdependent proposals. A critical element of collaborative design is conflict management, which can be perceived as the succession of mainly three phases: conflict detection, identification of the conflict resolution team, and conflict resolution.

When the collaborative system detects potential design conflicts among all sub-systems of a product, knowledge rules are capable of transmitting information of the design conflicts to the related members of the collaborative design teams and providing appropriate suggestions for resolving design conflicts and improving collaborative product development. This kind of strategy thus provides the ability to define messages with the sender and the receivers (for disseminating conflict information), together with a title and some text that explains design conflicts and suggests methods of resolution. This function plays a vital role in the product development process by shortening product delivery time and reducing product development costs.

We classify conflicts into three categories:

- Format conflict: ways of expression are not identical, which causes the information to be transferred differently. The data is not unified, planned or managed effectively.
- Content conflict: when the constraints of different knowledge fields cannot fulfill the system's need, or when there is a difference of viewpoints on the same product.
- Resourse conflict: arisen when the human, manufacture, software or hardware resources cannot be scheduled reasonably or provided in time.

Figure 7.1 shows an example of a conflicting situation in a collaborative design project that uses the FBS ontology.

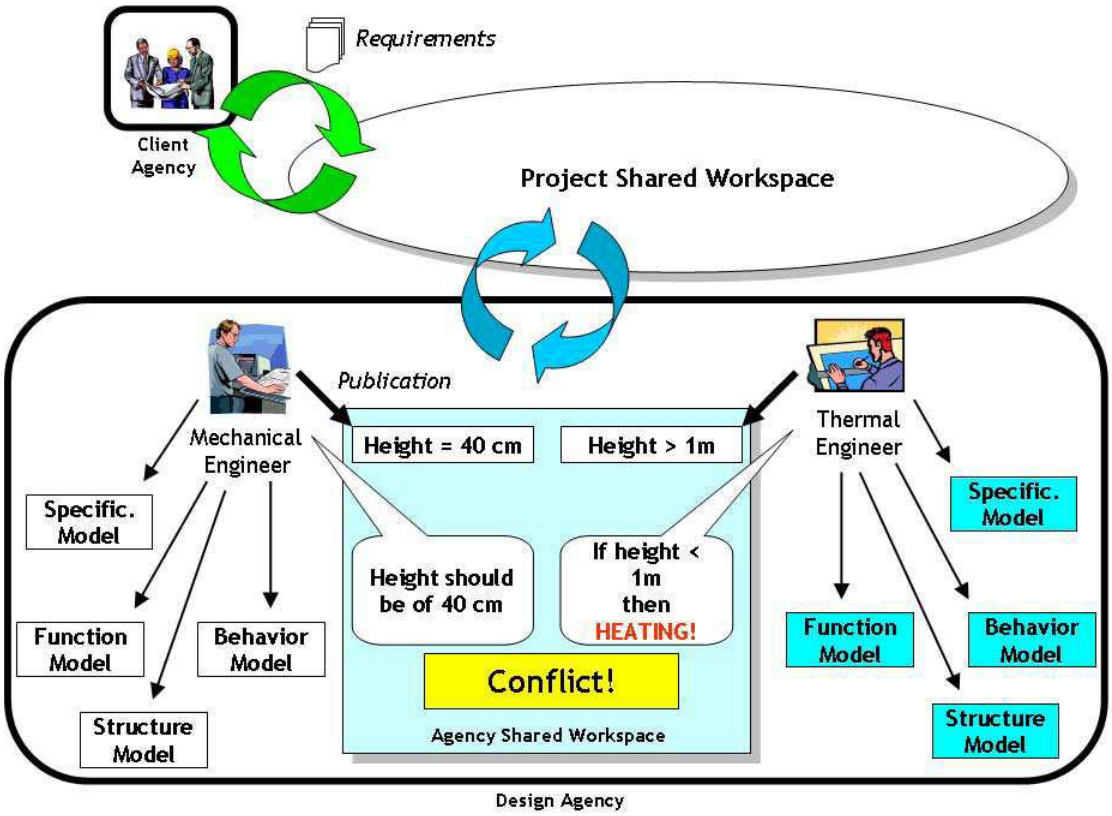

Figure 7.1 – Conflicting Situation

A critical reality in integration is that knowledge obtained from different sources may often be conflicting. It is characteristic of collaborative engineering design that precedence relationships among design activities contain information flow conflicts. Due to multi-actor interaction, conflicts can emerge from disagreements between designers about proposed designs. Hence, a critical element of collaborative design is to manage the detected conflicts and particularly the impacts once they are resolved. Indeed, the conflict resolution comes up with a solution which often implies modifications on the product and the process organization.

The non-translation of the initial requirements into precise and totally determinist specifications creates the freedom space essential to the design activity and enables innovation. However, the existence of ambiguities is also the source of all conflict causes that occur in collaborative design. The localization of these ambiguities into each one of the design space dimensions (product models, resource sharing, data exchange between tools, goals, and sub-goals) determines the conflict causes and the choice of specific mitigation strategies. To enable a generic approach for conflict prevention one should present to each designer his/her degree of freedom to do each task in the design space, concerning goals, available resources and the design of the part of the product to carry out. At the same time, each designer should be warned about the overlapping spaces between tasks and about potential interferences between the consequences of his/her decisions and those of other designers.

The blackboard structure defined in our architecture has a sub-module for conflict handling, the conflict area, as shown in the Figure 7.2.

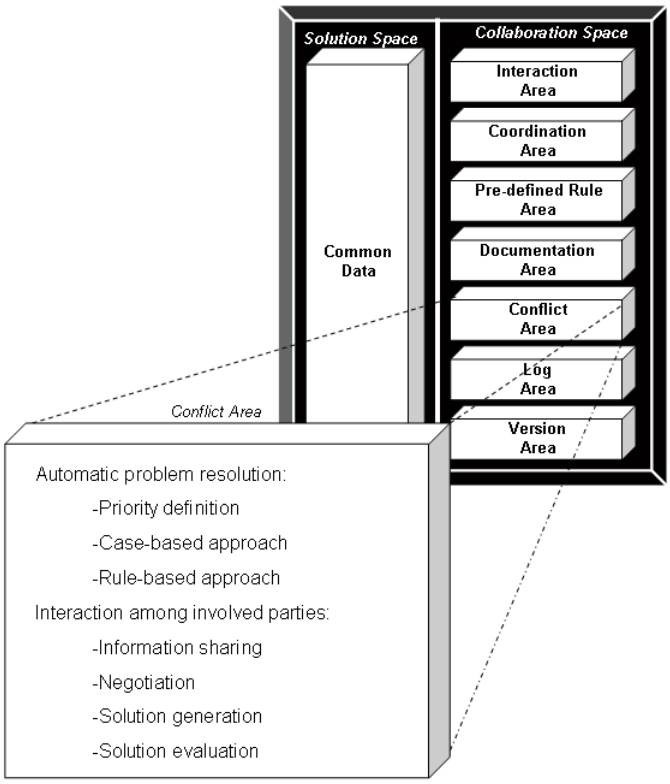

Figure 7.2 – Blackboard's Conflict Area

7.1 Automatic Solving

In a complex system, where different expertise is shared, it is always likely that some conflicting situations appear. It is of course necessary to avoid these situations as much as possible, as they can delay or even block the entire design process. To prevent such situations, it is desirable that the experts/agents coordinate their activities. In spite of this coordination, if a conflict pops up, it is necessary then to call the conflict resolution procedure.

This procedure uses a mechanism for automatic/semi-automatic resolution of conflicts. This automated approach consists of the following stages (Dutra et al., 2008b):

- Priority definition: Priorities are defined by the project leader. He is the one in charge of setting that a specific designer/ontology has priority over another one. In this case, the highest ranked ontology that is consistent is preferred as the common model.
- Case-based reasoning: It is an analogy-based approach. Past cases are analyzed in order to help the resolution process. The conflict is compared to what has already happened before in similar situations. That means that the system consults a knowledge base (KB), which contains solutions for previous conflicting situations. This KB should be loaded in the beginning of the project. Yet, as this approach is not easy because of the great number of involved variables (at the end, we can only say

that rigorously identical situations can be compared), we do not take the achieved solution to be "the one," but rather send it to be evaluated by the concerned designers.

- Rule-based reasoning: If rules have been defined, a deduction process may take place. This step uses the same principles employed by the constraint checker for conflict attenuation. Both approaches are based on rules. Besides, a constraint relaxation process may also be undertaken, if the project manager decides to.

7.1.1 Analogy-Based Resolution

Providing the designers with diagrams containing their propositions and the relations among them allows them to have an overview of the evolution of the design process. Thus, we propose the creation of a conflict repository to store the design history, to make possible the analysis and contextualization of the conflict's causes, in order to prevent them from occurring again. This information can be explored by inference engines, to avoid future conflicts or to suggest alternative solutions.

A conflict repository can serve as a valuable resource for fostering more effective, accumulative and cross-disciplinary research on conflict management, in several important ways. The taxonomic structure of the repository facilitates finding gaps in the conflict management knowledge. The conflict repository structure can enable structured discussions by organizing them around focus topics such as filling in a particular branch of a taxonomy, adding to a trade-off table, or detailing a particular process description.

One can improve conflict mitigation by saving and enabling access to the reasons behind the design decisions and, in some cases, to the record of what decisions were not made and why. Applying the design rationale to conflict mitigation can prevent some conflicts by identifying and presenting to the designers what causes deadlock situations. It can also make easier negotiation processes by confronting and comparing different designers' viewpoints and arguments, and the relations between them. Concerning case-based reasoning applied to conflict resolution, we can say that the design rationale contextualizes the conflicting situations better.

7.1.2 Deduction-Based Resolution

When communication between collaborators includes solution spaces instead of single solutions, collaboration systems may provide additional engineering support. More specifically, a system using solution spaces may assist in avoiding artificial conflicts, detecting real conflicts, maintaining project consistency, making informed decisions and guiding negotiation. Constraints provide a convenient means for representing, exchanging and manipulating information about solution spaces.

As the design system progresses (goes from one state to another), the set of constraints evolves, i.e., new constraints are added whereas some others are removed. To enable the generation of new constraints and the suppression of others (ΔC_i) – simultaneous to the ongoing design process – it is necessary to define exactly which constraints to add, to suppress, or to update, and when. There are three situations that demand constraint generation:

- When a new property or a new object is created. This new property/object may require constraint definition;
- When a problem is formulated, constraints related to the involved objects and their functions may arise;
- When a problem is decomposed, constraints may arise from properties or objects from the resulting sub-problems.

A given stage E_i affects the hierarchy of problems and objects, updating them. After each stage, the system triggers the constraint generator, which takes as input the current state of the design process e_i and the actions from the stage E_i, to generate ΔC_i (updates done on the constraint network, i.e., constraints and properties used to add/remove C_i in order to construct C_{i+1}) as output.

Constraint checking is undertaken at the end of the constraint generation procedure. It takes as input the new constraint network C_{i+1} to gather information about this network's constraints, i.e. the actual status of each constraint (respected, violated, currently undetermined), as well as to identify the inconsistent properties from each violated constraint. At the end of the constraint checking process, the design process advances from state e_i to state e_{i+1} (execution of the transition t_i), and the outputs from the generation and checking procedures (ΔC_i and the status of C_{i+1} constraints) are stored into the system design log. The log records enable us to trace the evolution of the property values from their initial or last stable state (validated by all the actors). This trace can then be used by the designers to understand the causes of the conflict.

Examples of constraints:

```
a) AircraftCargo:weight < 100T
b) AircraftCargo:weight = (Cargo:weight + Reservoir:weight +
   JetPropulsion:weight + Structure:weight)
```

Besides the use of constraints, a rule-based knowledge representation allows knowledge engineers or design experts to express design knowledge intuitively. Knowledge rules can be viewed as a simulation of the cognitive behavior of human experts. The antecedent of a knowledge rule will include several conditions, which can be formula or variable conditions. These conditions are combined by using logical relationships, that is, *AND* and *OR*. When the design parameters satisfy all conditions of the rule, the conclusion can be drawn. As stated in Chapter 3, we believe that SWRL is a suitable way to increase the power of OWL-based ontologies, as the SWRL rules provide procedural knowledge, which compensates for some of the limitations of ontology inference, particularly in identifying semantic relationships between individuals. We have used SWRL Tab in our architecture, the Protégé plug-in to work with SWRL rules.

Example of SWRL rules:

```
a) Implies(Antecedent(Plug(?x1))
           Consequent(partOf_Shell(?x1)))
b) (SmallScrew(?x1) ∧ Shell(?x2) ∧ hasComponents(?x2, ?x1)
    → partOf_Plug(?x1)
```

7.2 Interaction and Negotiation

At the end of the automated approach, if the conflicting situation persists, a negotiation process is started. This time, the concerned designers directly take part in all phases of the task. Each of them – to whom the conflicting situation matters – receives a system notification (Figure 7.3) saying that an ontology merging has failed and that a conflicting situation has come up. This notifying message must "translate" the detected inconsistency to a comprehensible description, since designers are not necessarily ontology experts. Subsequently, the notification recipients must contact their partners in order to resolve the conflicting situation, attempting to produce together a feasible solution for everybody. They have the possibility of checking each other's proposed models by accessing the public workspaces (*intra-* or *inter-agencies*).

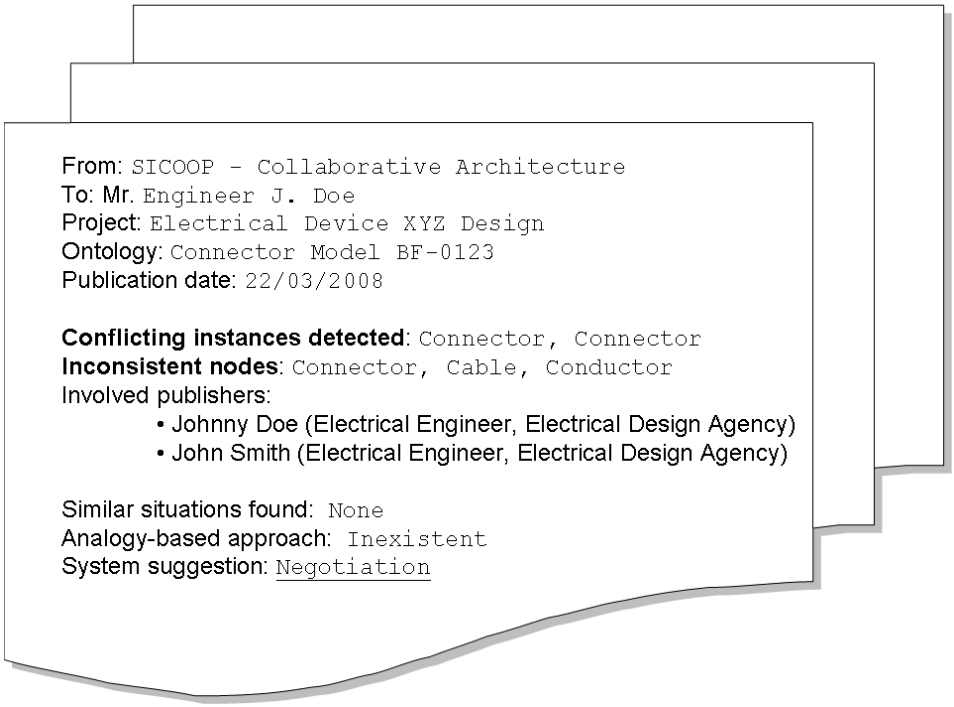

Figure 7.3 – Conflict Notification Example

Conflict resolution, whether performed during the design phase or during run-time, can be costly and, if done without a proper understanding of the usage context, can be ineffective (Dutra et al., 2008b). Conflict resolution cannot be achieved by one single actor since it requires the interplay of different areas of expertise. To avoid iteration in the conflict resolution process, it is highly advisable to do it in a collaborative way that seeks the input of many actors to reach a consensus quickly. These actors refer to those designers producing the product data leading to the source of the conflict. In order to identify these negotiators, a kind of "data dependency network" is extracted and the product data – on which the source of conflict depends – are identified through network backtracking (the source of conflict being the starting point). A set of queries are then applied in order to identify the negotiators forming the conflict resolution team. Once the team is identified, they collaborate during a negotiation phase in order to come up with a solution to the detected conflict.

The negotiation leads to a solution which often implies changing one or more input data of the activity where the conflict has emerged, and thus, generating a cascade of modifications on the already produced data. The impact propagation is therefore highly dependent on the handled data during the design process. It supposes knowing the dependency relationships between the conflict source data and the data previously produced. Thus, it arises that propagating the selected solution impact mainly consists of identifying the dependency relationships between engineering data handled during the design process. Furthermore, the impacted data have to be redefined. This requires a re-execution of the design activities responsible for the elaboration of these product data and also an adjustment to the preliminary design process organization.

Negotiation-based conflict resolution protocols, particularly argumentation protocols, are well suited for autonomous co-operative agents. In argumentation-based negotiation, or simply argumentation, the agents generate and exchange arguments to support their findings or conclusions, evaluate them, and agree to accept the soundest arguments presented or the first consensual alternative proposed. Argumentation-based negotiation methodologies are developed and applied to solve unforeseeable and, therefore, unavoidable conflicts. The supporting conflict module is a distributed belief revision system where argumentation plays the decisive role of revision. The distributed belief revision system detects, isolates, and solves, whenever possible, the identified conflicts. The detection and isolation of conflicts is automatically performed by the distributed consistency mechanism and the resolution of the conflict is achieved via argumentation.

We use two argumentation protocols to solve different types of identified information conflicts – context dependent and context independent conflicts. The autonomous agents are modeled as individual reason maintenance systems where beliefs are always represented with their supporting justifications. A belief's justification constitutes a full argument in favor of the belief and is composed by the set of elementary beliefs (called foundations) that support the belief. So, when an agent shares a belief with others, it sends the belief together with its supporting arguments. The receiver agents, with this additional information, verify if the belief they received is consistent with the ones already represented and, in case of conflict, immediately identify and isolate the conflicting set of beliefs in order to preserve consistency.

Negotiation focuses on understanding what local behaviors are to be expected from (the relatively small number of) self-interested agents attempting to come to agreements in the face of their interdependencies. Complex systems research complements this perspective by attempting to understand the global dynamics that emerge as the collective effect of many such local decisions. These two perspectives, when brought together, we believe, have much to offer to the understanding of the dynamics of collaborative design. Furthermore, due to information latency and disciplinary differences, it is often a difficult task for designers to reach agreements when needed. Despite this, negotiation has been a successful method for facilitating information exchange, mutual understanding, and joint decision-making.

We propose a "collaborative activity" for the negotiation process. Once this activity is created, it is necessary to initialize it by creating a first iteration in which the problem is explained. If the information is missing, this will be requested by the concerned users (who have been previously notified). The management and the assigning of the best human resources by the project manager are essential to meet the milestones and the quality requirements for the product design. The software solution is supposed to help the project manager refine the design, and the actors are selected by providing some performance

indicators. These performance indicators are taken from the past project observations of the involved actors, regarding the type of collaborative work situation they are involved in.

Once the collaboration activity is initialized, the users previously selected can iteract via the interactions formalized in explanations or justifications (notifications). They must interact as much as they can to resolve the conflict. However, if accomplishing such a task is not possible for any reason, whatever it may be, a mediation process will be undertaken by the project manager (Dutra et al., 2008b). During these exchanges, one of the users can take the initiative to create a new solution that can be discussed by the consortium. Stakeholders strive to use popularization knowledge while running these phases of popularization/mediation, in order to be as understandable as possible to maintain a certain synergy within the established consortium.

At the end of a pre-defined duration, and in case there is no convergence between the proposed solutions, a vote request message is sent to various participants. A time delay is granted to validate their choice from the proposed solutions. Each subscribed agent must be identified to be able to vote and then choose a solution among those previously proposed during the various previous iterations and validate his/her choice. When the agent sees that no solution is relevant, it can decide not to vote. The result must take into account the constraints regarding participation, voting, and specific timeframe.

It is instantiated further to a friendly consensus obtained by explicit support of all the actors to a given solution before the deadline of the popularization/mediation phase; further to a consensus stemming from various iterations of vote made by each subscriber, or in case of vote process failure by the project manager. The knowledge capitalized in this iteration permits the project manager to advantage of this experiment and the stakeholders to reuse it in a future similar project. At this moment, since he/she will optimize the contents of the design and the team composition by considering capitalized information about actors, he/she will be able to propose a more efficient design environment.

To summarize, we can say that conflict negotiation is run by several iterations of explanation and argumentation, proposed to refine the conflict causes. After all the actors have agreed upon and checked that the conflict causes were well identified, a first solution concerning the modifications to be made on the product is proposed. This solution must then be capitalized, in a process of common agreement between all the involved parties. Once the conflict solution is achieved and modifications are carried out, a series of validation checks is done on the resulting model. The project leader or the agency coordinator (depending on the collaboration level) then closes the conflict management process and the various subscribed actors are thus informed of this decision.

7.3 Supervision

The mediation – or supervision – phase aims at advocating an alternative solution to resolve a popularized problem and thus avoiding the unfortunate consequences that might be generated by the former solution. It comes from the actor who has just popularized the problem in order to propose the solution, or from another conflict concerned actor who would resolve the popularized problem. Once a solution is formulated, actors concerned by this new solution are notified (the list of actors concerned with a new solution varies from one mediation to another because each new solution may introduce new expertise according to its introduced

technologies). Then, each actor from the subscribing list has, on his/her turn, to argue his/her choices (a stage of popularization is launched). He/she can contribute to the decision-process by proposing an alternative solution and/or criticizing the alternatives proposed by other subscribers.

Sometimes, we can avoid supervision by simply applying "weights" to viewpoints, designers, or situations. These weights will be used to build a priority scale, to be consulted during a conflicting situation.

Figure 7.4 shows the UML Activity diagram for our conflict resolution method.

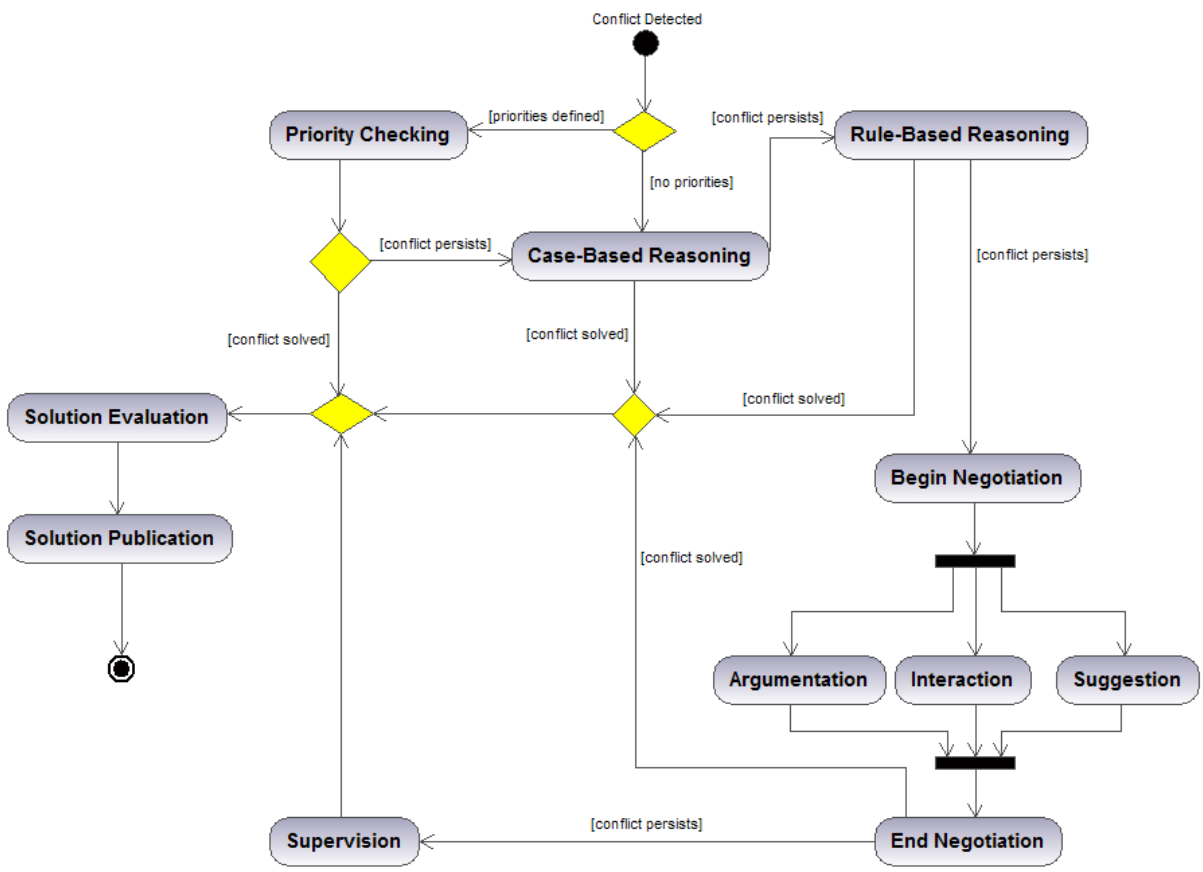

Figure 7.4 – Conflict Resolution Method Activity Diagram

7.4 Summary

Solving collaborative design conflicts is a hard but rewarding task to accomplish. Differences among designers must be taken into consideration, especially when considering a very-large-scale design project. Each one of them has different background, different expertise and different cultural and social viewpoints. Hence, it is never that easy to solve problems when such a set of people is involved.

We think that dividing this task into two sub-approaches is suitable for enhancing the system performance, as the interaction/negotiation phase is only handled in the last case. Automating

the first part of this process – through priority definition, analogy- and deduction-based reasoning – saves time and provides the users with formal criteria to resolve conflicts, namely priorities, rules, and past cases. The negotiation process is facilitated by the fact that designers are already used to interacting, because they have done so during the whole design process. Thus, this stage should be seen as a natural step to them, not as something unusual. Furthermore, the collaboration mechanisms provided by the system serve as a support for the negotiators, as they are able to see the proposed models from one another, and to propose/suggest changes, in order to smooth the integration process.

Our architecture has been thought out to avoid as much as possible the mediation/supervision stage. We consider that the formalization process undertaken at the beginning of the design process is a very important method to mitigate conflicts, as it lets the designers "speak" the same language. Consequently, we believe that this strategy, along with the automatic solving during the conflict resolution process, contribute decisively to make the mediation a rare action to be done.

In the next chapter, we present our prototype, which was built to simulate the whole proposed architecture, and show how it can be used to provide collaboration in the design process.

Chapter 8

8 Prototype

In this chapter, we present a prototype developed to simulate the proposed architecture. This prototype is a standalone application, whose main goal is to simulate the ontology-based knowledge representation and detect conflicting situations during the publication of the model. This chapter comprises two parts. In the first part, we present the example scenario example. In the second one, we present Scoop, a prototype for collaborative design.

8.1 Modeling an Electrical Connector

In order to validate the proposed method for dealing with conflicts, a case study is used to demonstrate the collaborative development of a product design. The model of an electrical connector was chosen as an example scenario for this validation.

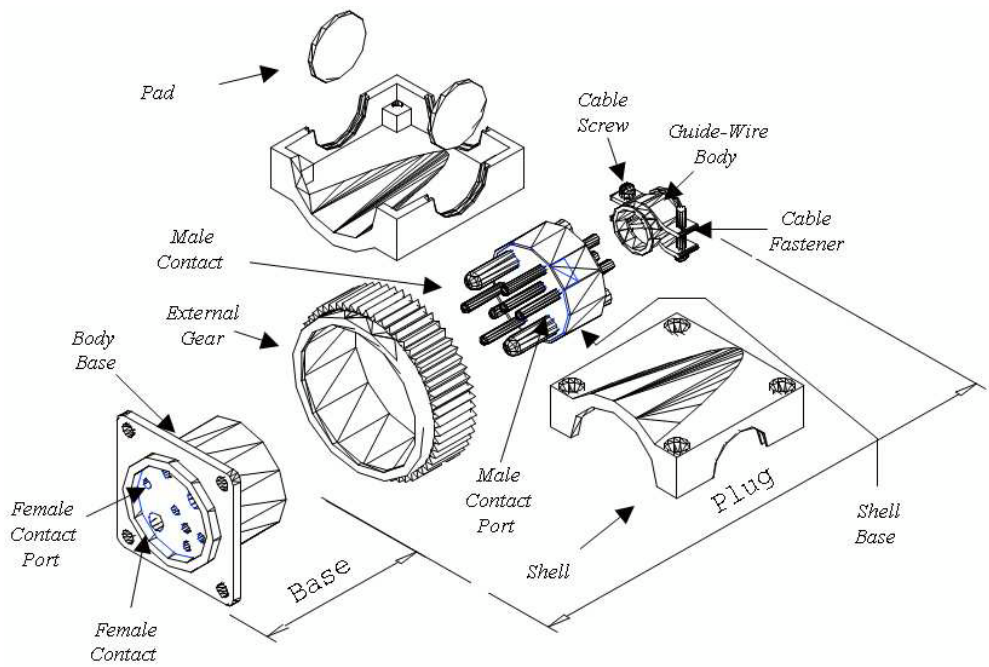

Figure 8.1 – Electrical Connector

Six agents take part in this collaborative project: a project manager, a consumer, an electrical engineer, a thermal engineer, a mechanical engineer, and a manufacturer. We consider that a generic ontology has been defined previously and is ready to be used by the design participants. Figure 8.1 shows the connector to be modeled.

The project manager is responsible for the creation of the project's agencies, namely Client, Manufacture, Mechanical Engineering, Electrical Engineering, and Thermal Engineering Agency. Subsequently, each agent is attributed to an agency, and his/her tasks are then specified. To simplify the scenario, each agency has only one agent. The prototype provides for the designers graph- and ontology-based tools for modeling, which let them structure their models as trees. The generic ontology is based on the FBS model (cf. Chapter 6). With the exception of the consumer and the manufacturer – both responsible for giving their product requirements only – all the other designers must propose a model for requirements, functions, behaviors, and structures. Figure 8.2 shows the product requirements given by the consumer. In Figure 8.3 we see the manufacturer's requirements.

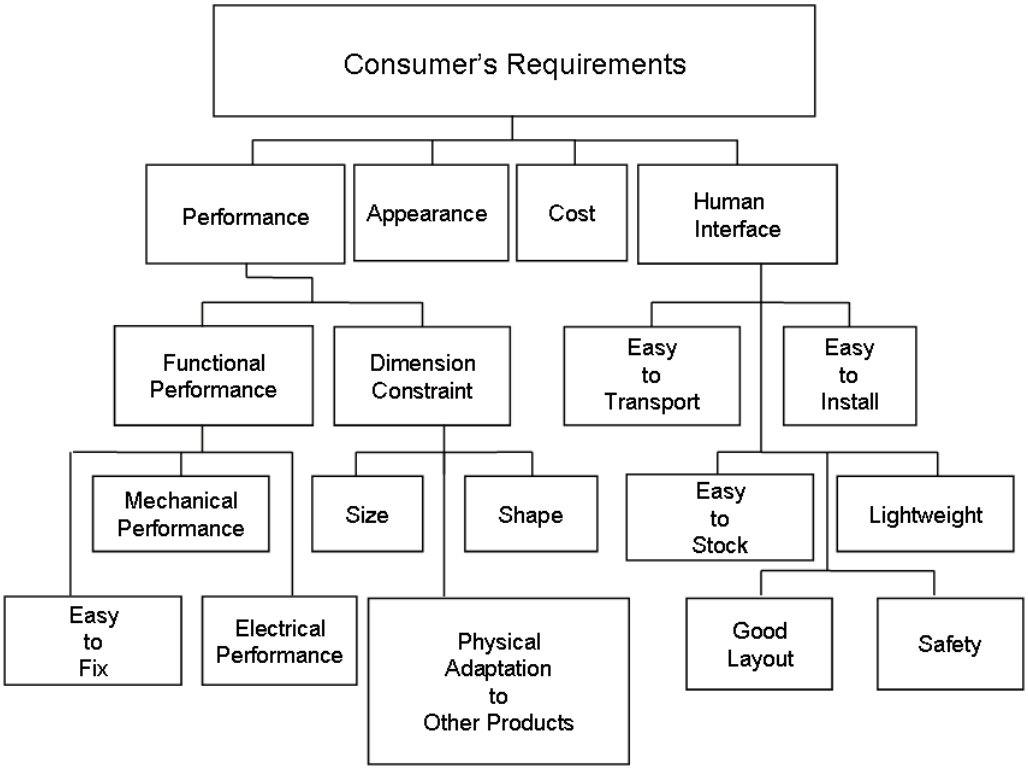

Figure 8.2 – Consumer's Requirements

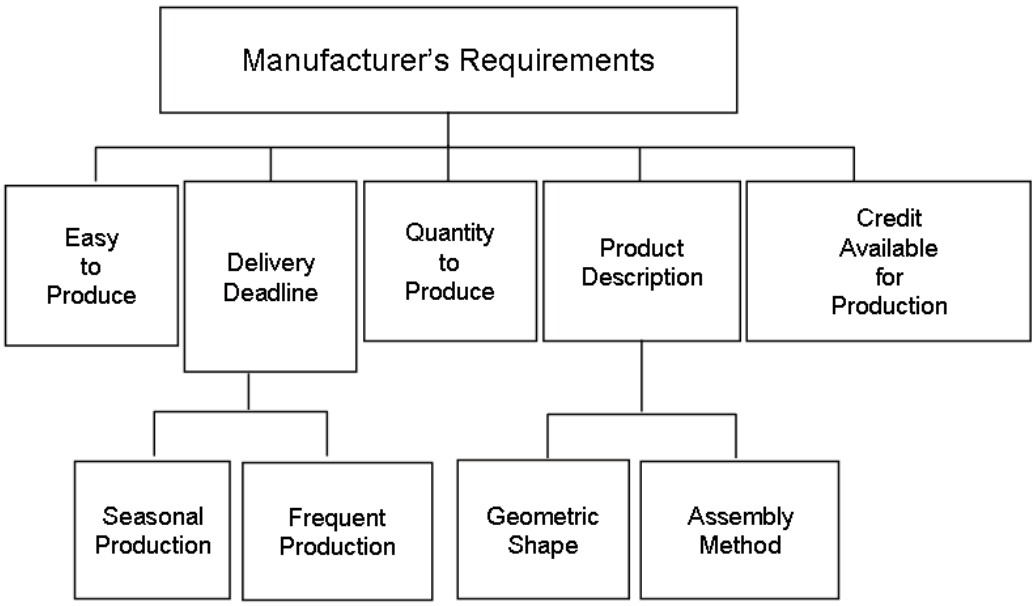

Figure 8.3 – Manufacture's Requirements

They will be eventually integrated with other agents' requirements in order to produce the final model for the requirements of the connector. Figure 8.4 shows the final function model, obtained after the integration of the models proposed by the Mechanical, Thermal and Electrical Agencies.

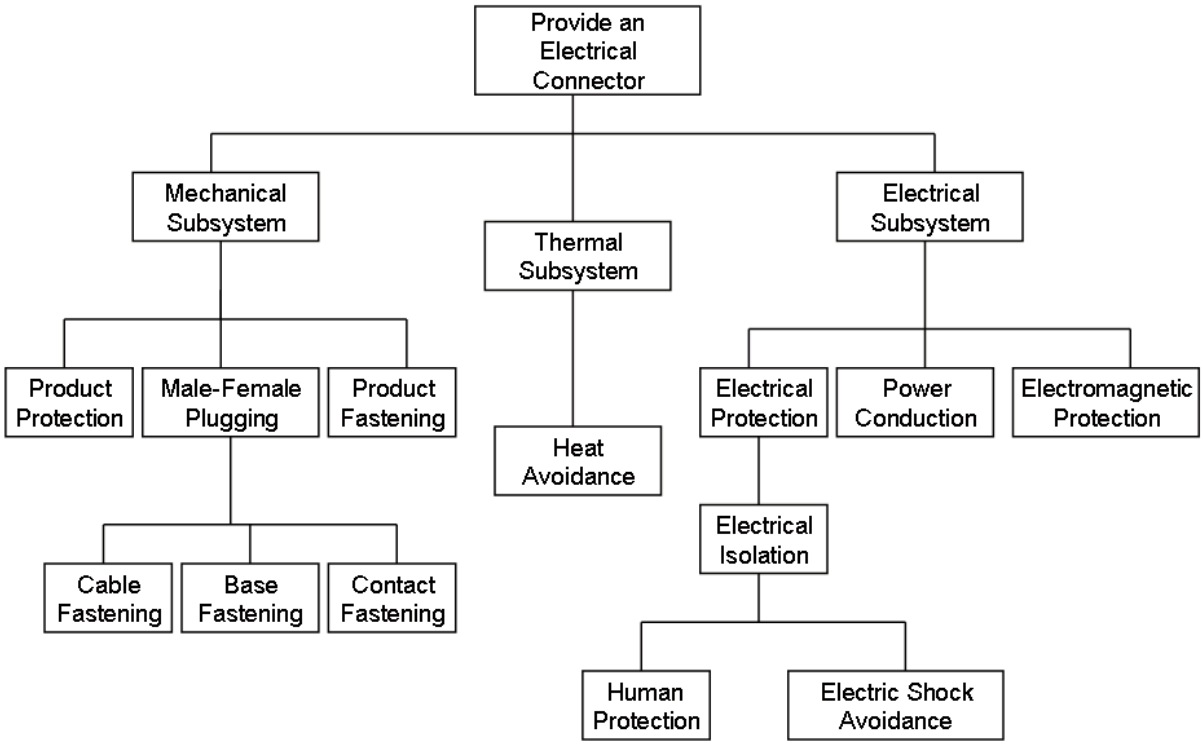

Figure 8.4 – Connector's Function Model

A regular use case of this scenario would consist of the following steps:

1) The project manager creates the project and defines its goals.
2) Later on, the project manager creates all the agencies and agents needed, establishing the required links between them.
3) The agents log into the system.
4) Each agent starts to develop its private model.
5) Once a model is ready, the agent publishes it into the agency shared workspace. If the model is consistent and does not conflict with another model, it can be published into the project shared workspace too.

Step 5 is repeated until the whole design process is completed or until a conflict pops up. In this case, the following steps are taken:

6) The system notifies the involved agents that a conflict has come about.
7) The automatic resolution approach is attempted.
8) If successful, the model is published. Otherwise a negotiation process is initiated.
9) Once the final model is achieved, the other agents are able to access the common data, give their opinion and evaluate the final proposed model.

10) If no further conflict appears, then the last integrated model is the final product model, and it is ready to be sent to the assembly line.

8.2 Scoop – System for Cooperative Work

This collaborative tool is called Scoop – System for Cooperative Work (Figure 8.5).

Figure 8.5 – Scoop Login

Scoop intends to emulate a collaborative environment, providing generic and synchronous design collaboration. Scoop's internal structure is based on ontology representation. Figure 8.6 shows the system administrator's workspace.

Figure 8.6 – System Administrator's Workspace

In Figure 8.7, we can see the project manager's workspace.

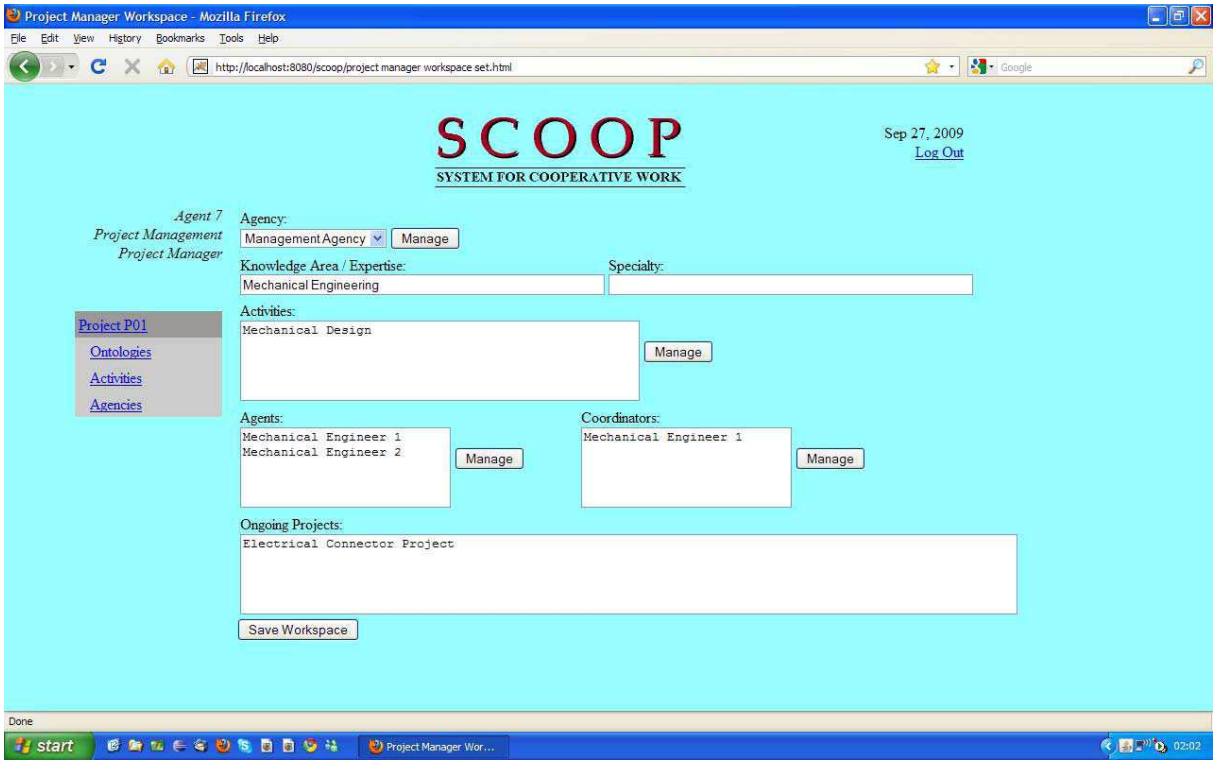

Figure 8.7 – Project Manager's Workspace

In the Figure 8.8, we can see the project manager during the attribution of agencies to the project.

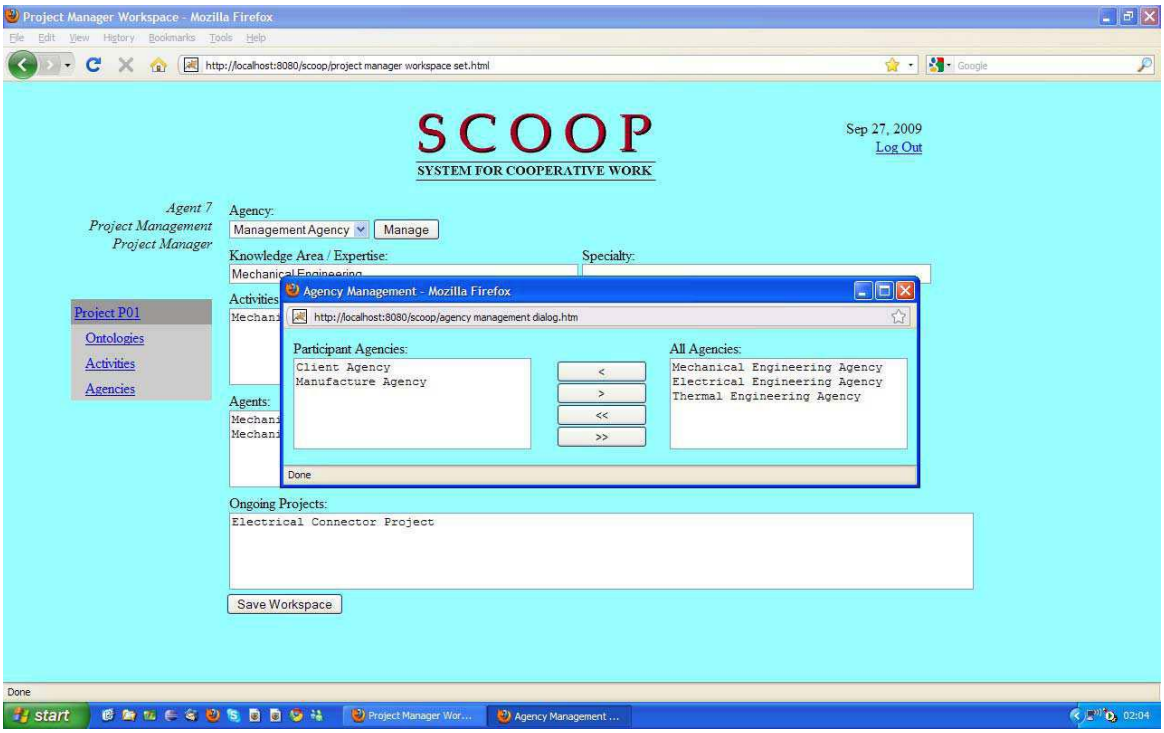

Figure 8.8 –Attribution of Agencies

Figure 8.9 shows the the thermal engineer's private workspace.

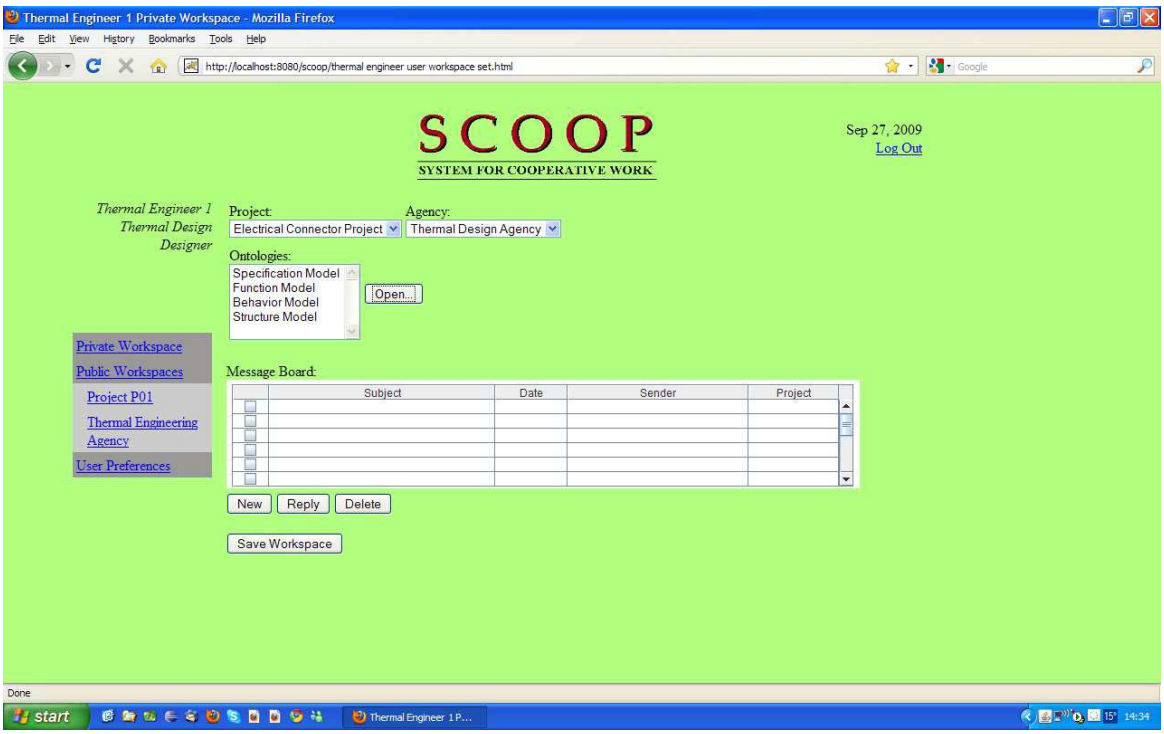

Figure 8.9 – Thermal Engineer's Private Workspace

Figures 8.10 and 8.11 show the specification models proposed by the client and manufacturer agents.

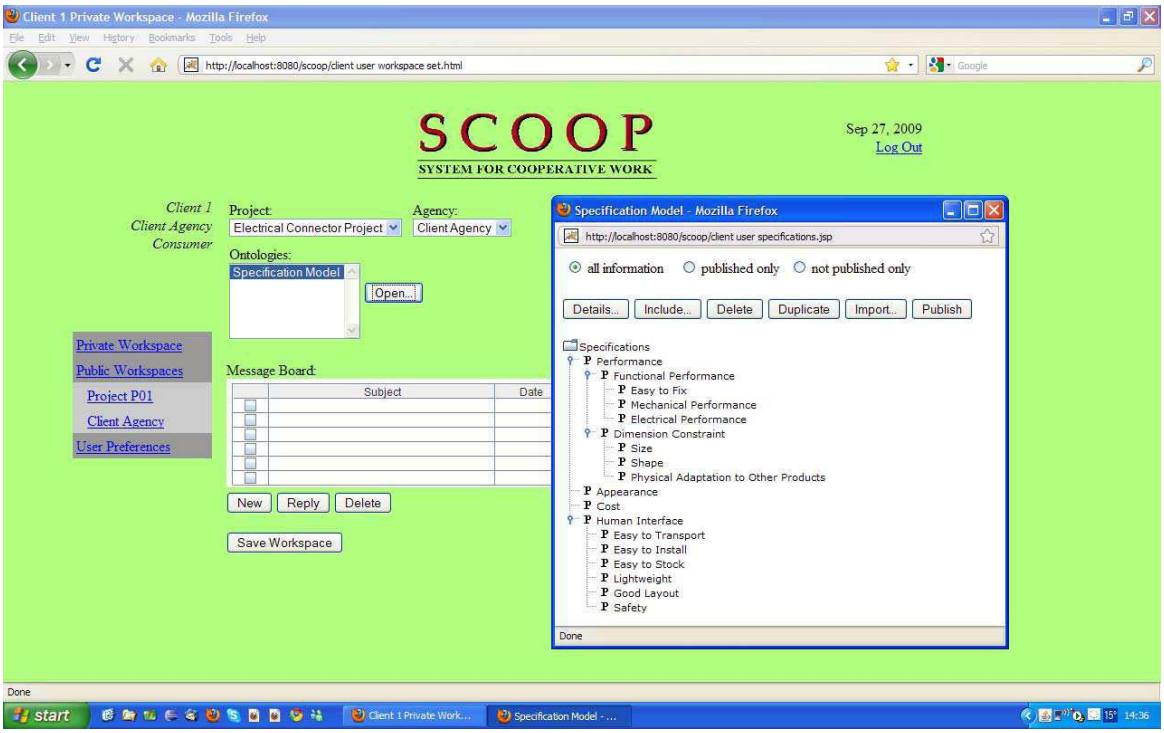

Figure 8.10 – Client Agent's Specification Model

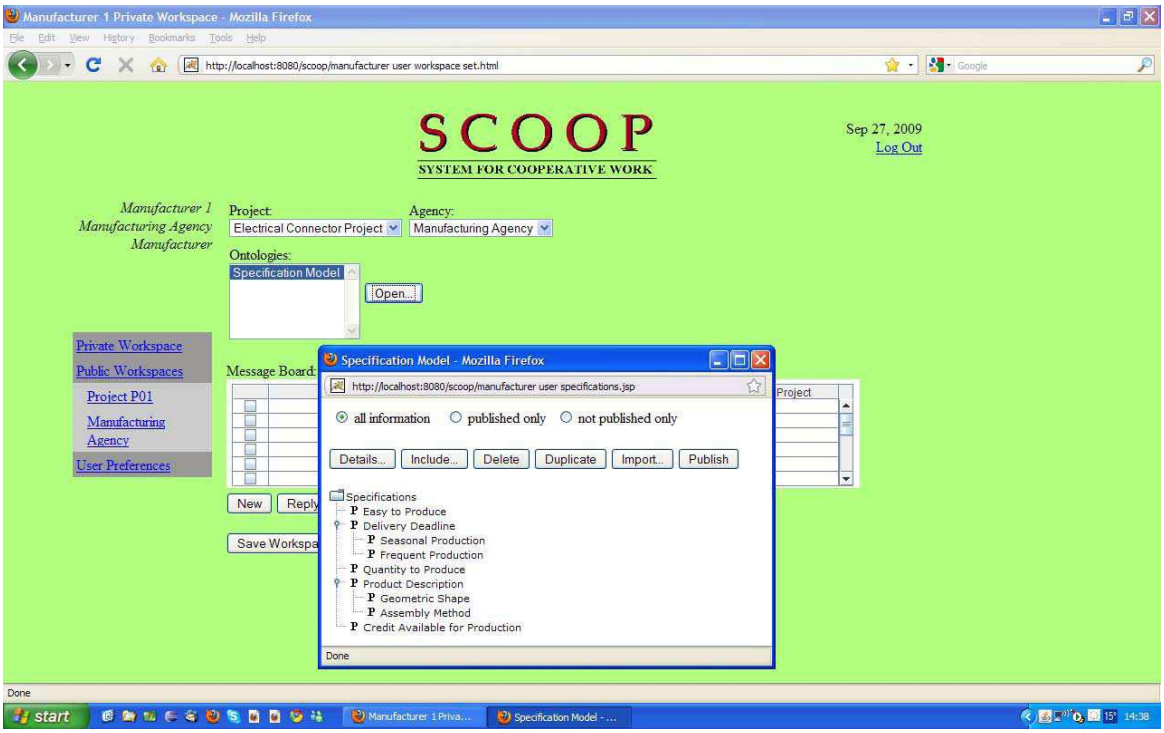

Figure 8.11 – Manufacturer Agent's Specification Model

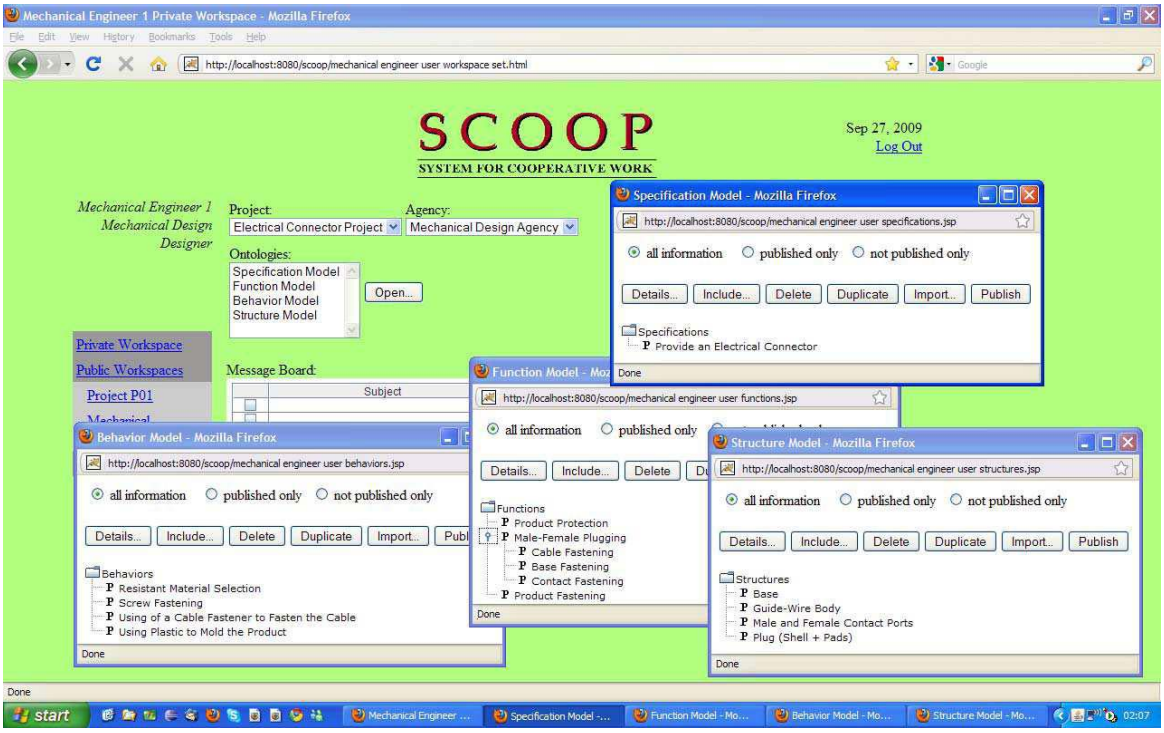

Figure 8.12 – Mechanical Engineer's Proposed Models

In Figure 8.12, we can see the models proposed by the mechanical engineer agent in its private workspace. Figure 8.13 shows the thermal engineer's models.

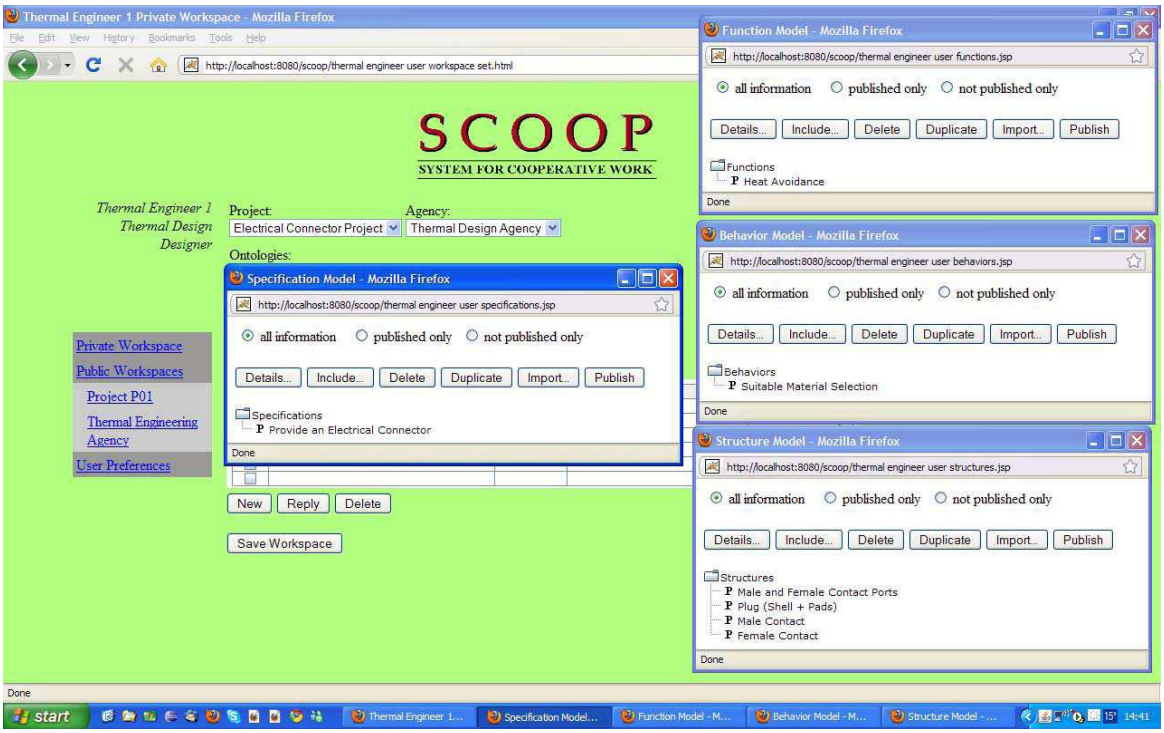

Figure 8.13 – Thermal Engineer's Proposed Models

In Figure 8.14, we can observe the project shared workspace and its composed models.

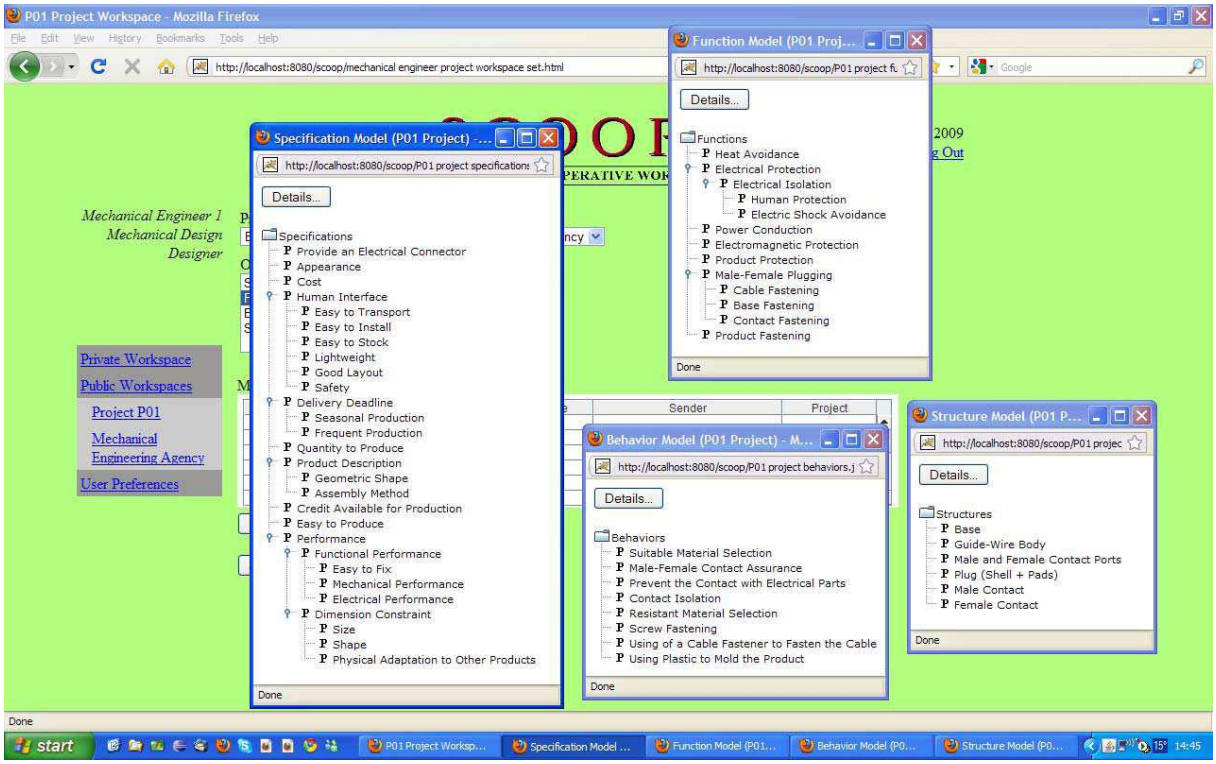

Figure 8.14 – Project P01's Shared Workspace

8.2.1 Background Tools

We have implemented Scoop in Java, using Pellet as the DL reasoner, SWRL as the means to represent rules, and Jess as the rule engine. Ontology consistency is often used to refer to concept satisfiability. We use Pellet for checking the ontology consistency, which can also be enforced by defining rules that capture such inconsistencies (conflicts). When bringing together (integrating) fragmented models, the aim is to obtain one single model in which we can easily identify such inconsistencies.

To support the detection of inconsistencies in our ontology, we have defined a class named Inconsistency; all instances causing a logical inconsistency are given membership to this class. The cases which cause a logical inconsistency are represented by a set of SWRL rules: a rule has an antecedent defining an inconsistent situation and a consequent that marks the individuals causing this inconsistent situation. Marking is done by asserting them to have a problem relation between them. A Problem is a property of the Inconsistency class. We have specified a set of rules assigning inconsistent individuals to the Inconsistency class via the property Problem.

Each design model is represented as an OWL class. We use Jess to run the SWRL rules. The rules are transferred to Jess along with the ontology, and the rule engine evaluates these rules during the publication of the ontology. We then use the Pellet API to check the consistency of the ontology and compute the inferred types (new assertions that are made by firing the rules in Jess). Pellet infers that features having a relational problem are members of the

Inconsistency class. Our approach allows the integration of models with distributed features by means of specifying the points of integration and using rules to check the consistency of the variability model. The combination of the rule engine's ability to run conflict detection rules with the reasoner's ability to infer new types enables the detection of inconsistencies that follow from implicit relations between the models.

The inference engine matches design parameters with knowledge rules. If the knowledge rule is satisfied, then the conclusion of the new knowledge rule will be activated. Jess is, therefore, well suited to develop an integrated knowledge representation and to support a dynamic, collaborative design inference via the Internet.

8.2.2 Conflicting Situation

As shown in Section 8.1, the connector design process is divided into nine steps. Each of these can be further divided into a number of sub-steps. During the mechanical design process, the final shape of the connector that meets the requirements is to be determined. The forging process is to select the optimum shell shape and proper shell filling through the adjustment of the parameters (cf. Figure 8.1).

Let us now consider a situation where a conflict pops up. The heat treatment undertaken by the thermal engineer is supposed to mitigate the residual stresses and distortion eventually generated. If the distortion and the residual stress are acceptable, the final shape will be accepted. If that is not possible, the mechanical design should be repeated to change the final shape.

In Figure 8.15, the UML sequence diagram that represents the publication of an ontology is shown.

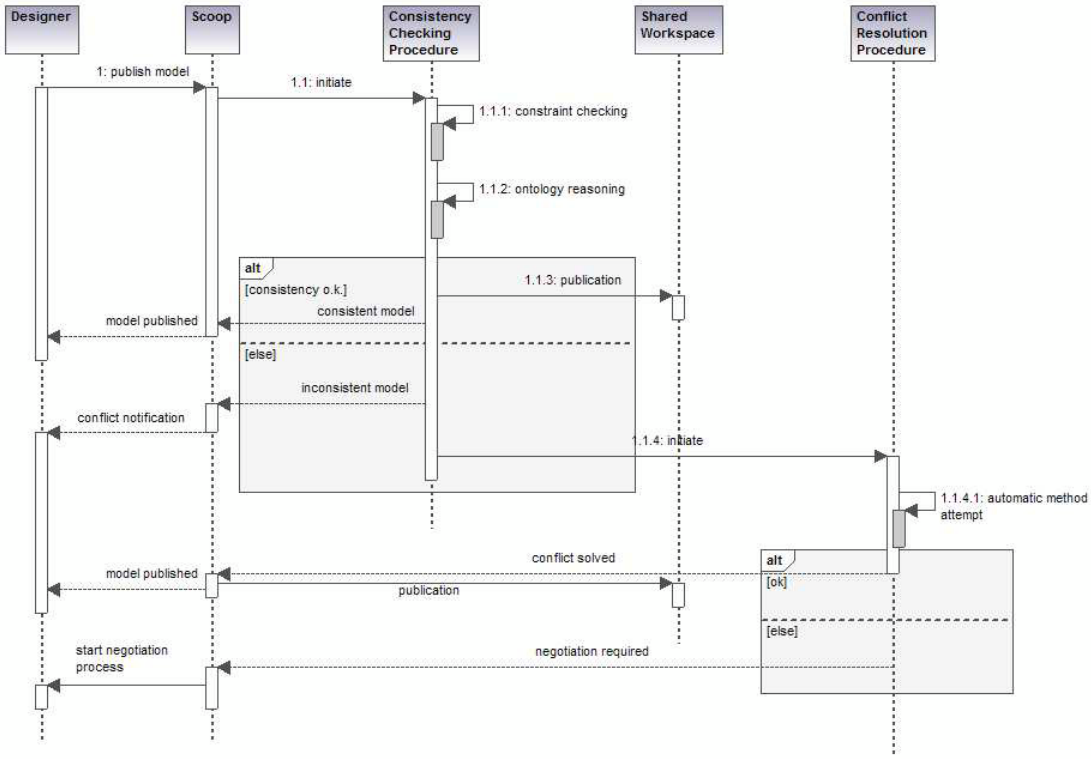

Figure 8.15 – Ontology Publication Sequence Diagram

The thermal engineer proposes the following model for the connector plug (Figure 8.16). He/she then publishes this model into the project shared workspace.

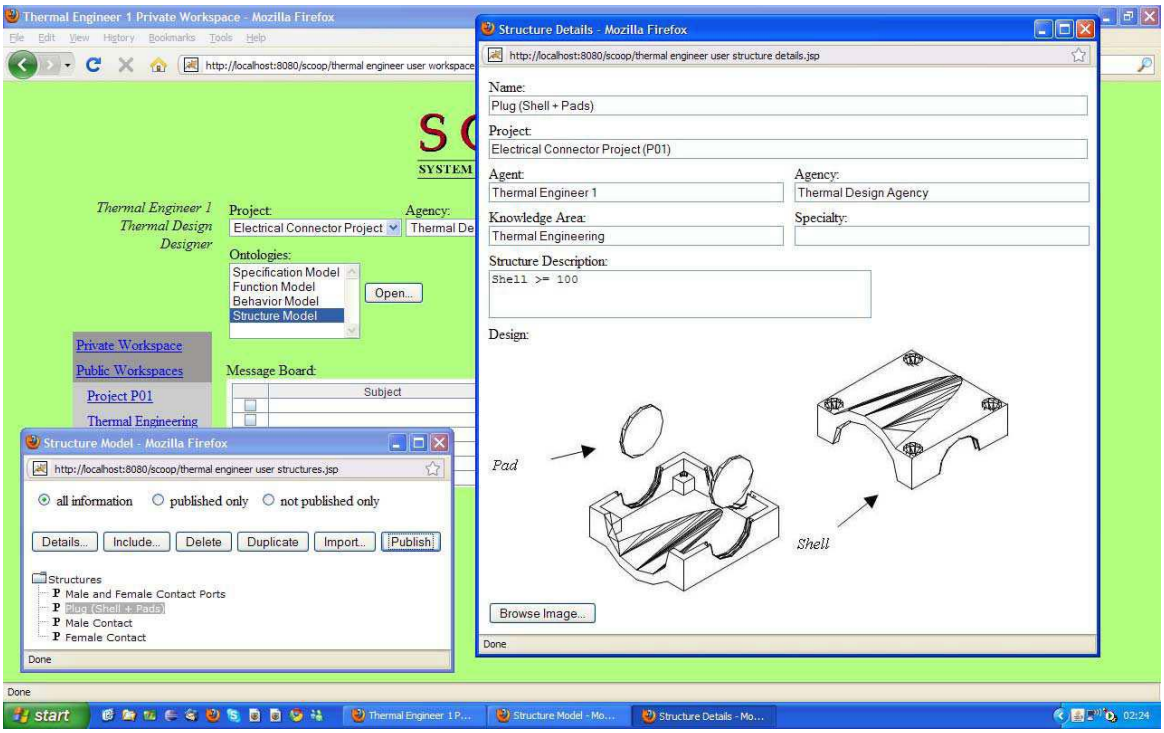

Figure 8.16 – Thermal Engineer's Plug Model

At the same time, the mechanical engineer proposes a slightly different model for the same plug, which defines another specific size for it (Figure 8.17).

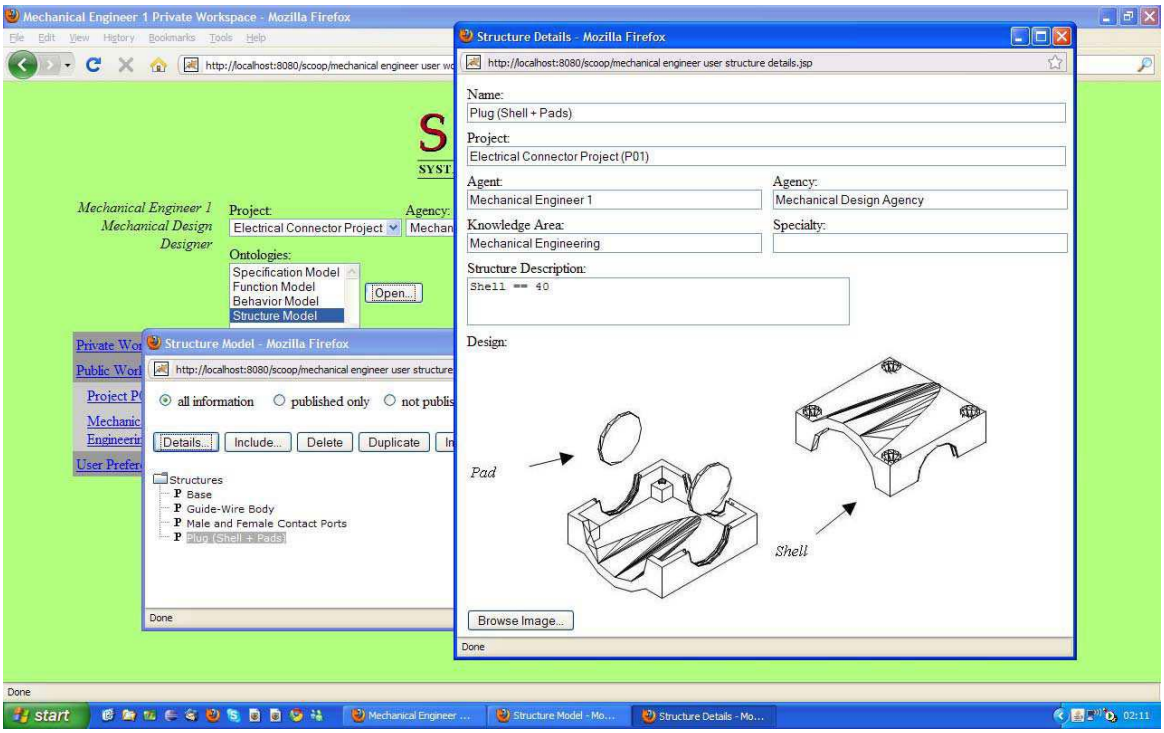

Figure 8.17 – Mechanical Engineer's Plug Model

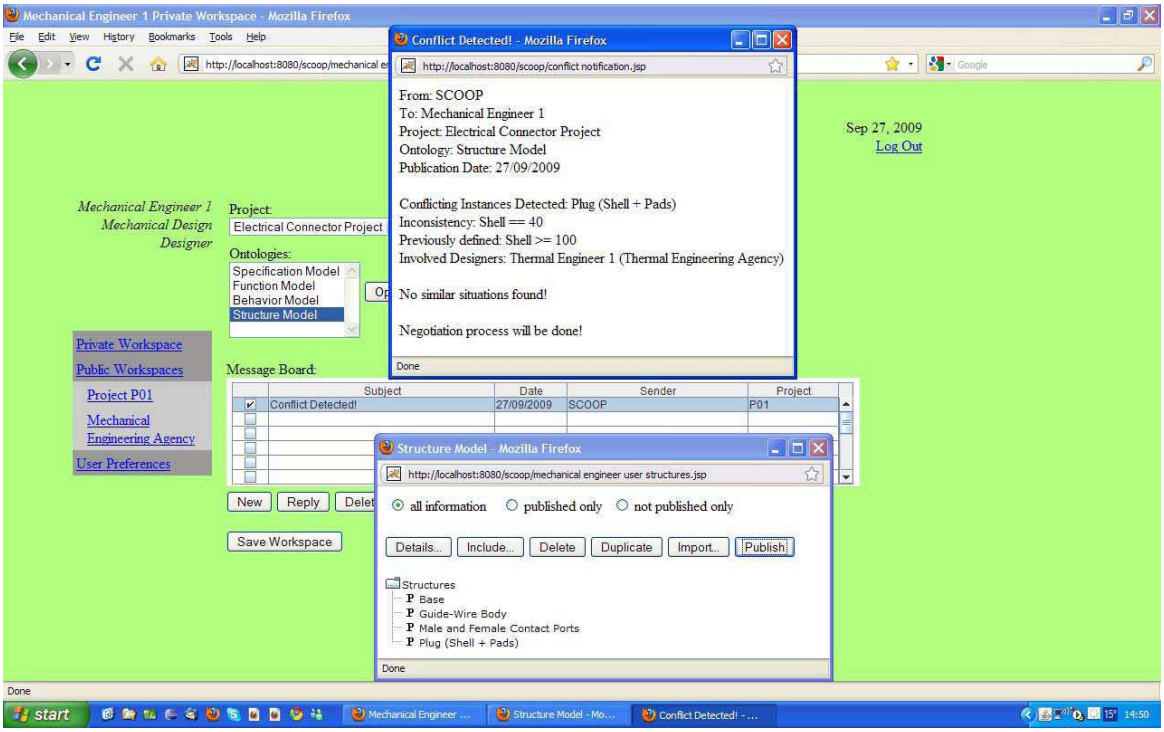

Figure 8.18 – Conflict Detected

When the mechanical engineer decides to publish this model into the project shared workspace, an inconsistency is then detected (Figure 8.18), as the thermal engineer has already specified another size for the same plug.

8.3 Summary

In this chapter, we have presented Scoop, our prototype for collaborative design. We have taken as a scenario the modeling of an electrical connector, which provided us with the possibility of working with people from five areas of expertise: the client, the manufacturer, the mechanical engineer, the electrical engineer, and the thermal engineer. We have created a small design project, in which the project manager structures five agencies (each of them representing a different area of expertise), creates the system agents and the activities to be held, and attributes them to the agencies. This project uses the FBS ontology as the generic model to be instantiated by the agents. The agents create their particular requirement, function, behavior and structure models, and try to integrate them into the public workspace. The requirements given by the client and the manufacturer were shown in order to clarify the connector's specification. We have also presented the integrated function model, in which we can see all the related areas represented.

We have considered tha case where a conflicting situation pops up when the mechanical engineer attempts to publish his/her model for the connector plug. The size specification of some part (shell) of the plug triggers the conflict notification. As our knowledge base has not been previously loaded (considering this is the first project to use it), it is not possible for us to undertake the case-based reasoning. Besides, aiming to simplify the scenario, no design rule or constraint or priority has been set. So, the natural way to proceed is to undertake a negotiation process between the concerned designers. A previous version of Scoop prototype (Figure 8.19), non Web based, has been used to model a scenario in the medical domain. This work was presented in Roummieh et al. (2007).

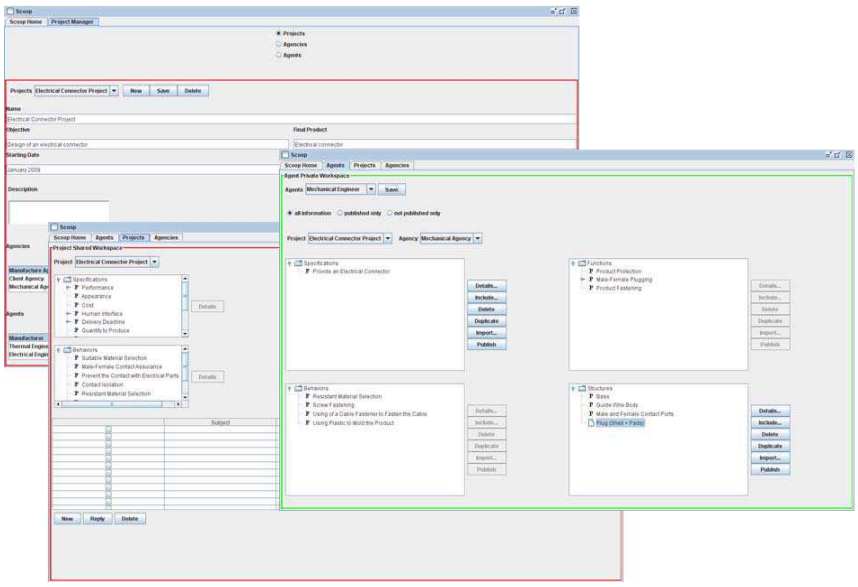

Figure 8.19 – Former Version of Scoop

Chapter 9

9 Conclusions

Current industrial systems use and rely even more on collaborative environments to design new products. There is a trend to share competence within collaborative processes. This sharing entails the exchange of information and data. To accomplish such a task, the collaborative systems should be flexible enough to allow different areas of expertise to collaborate. This flexibility comprises methods for standardizing knowledge representation models, methodologies for integrating these models, and computational-networked environments, which should provide for the designers the tools and infrastructure appropriate for collaborative design.

In this context, we consider that the major contibutions of this work are:

- A collaborative architecture for collaborative design that is:
    - Generic,
    - Synchronous,
    - Service oriented,
    - Agent based, and
    - Ontology based.
- A method to represent knowledge that is based on ontologies and uses a formalization process.
- A method for ontology integration that comprises two sub-approaches:
    - The use of a generic ontology and
    - The harmonization method.
- An ontology-based methodology for handling conflicts that is composed of two sub-approaches:
    - Automatic Solving:
        - Priority Definition,
        - Analogy-Based Resolution (past cases), and
        - Deduction-Based Resolution (rules and constraints).
    - Negotiation Process:

- Interaction,
- Supervision, and
- Result Evaluation.

In Chapter 2, we have evaluated a number of collaborative projects. We have put emphasis on the approach that integrates the Web and agent technologies for collaborative design, because we believe that the use of both technologies provides a more robust and effective strategy, since it combines the reasoning potentials of the multiagent systems with the portability and large-scale infrastructure of the Internet. We have also presented some other approaches for collaborative design, namely service-oriented architecture (SOA), synchronicity, and reuse and optimization, as well as presented some collaborative tools, which currently exist on the market. Most collaborative tools focus on communication (messaging) and coordination features (approval forms, workflow tools, video-conference tools) but few of them deal with collaboration among actors. Regarding this issue, besides shared documentary bases, market tools propose forums to facilitate the exchange of information among actors. Such spaces do not, however, formalize inter-skill relationships and do not guide actors toward setting up a common knowledge project.

In Chapter 3, we have presented a review of current research projects on ontology-based representation. We have discussed the ontology terminology and the Semantic Web paradigm. Moreover, we have talked about interoperability among ontologies and ontology management, as well as presented some ontology integration methods. We have also pointed out the importance of the formalization of the knowledge and the harmonization of ontologies, both of which are further explained and exemplified in Chapter 6.

In Chapter 4, we have presented a review of collaborative design conflicts. We have depicted a taxonomy of conflicts and some strategies for conflict mitigation and constraint management. We have highlighted some techniques that are used in our methodology, such as design rationale and case-based reasoning, rule-based reasoning, constraint definition, priority management, negotiation, supervision, and ontology use. We consider that the best strategy to use for conflict mitigation is the combination of the techniques presented in this chapter.

In Chapter 5, we have presented our generic and synchronous ontology-based architecture for collaborative design. We have proposed generic and synchronous approaches, in which designers interact at the same time that their models are being produced, and where they keep working with familiar ontologies. This architecture is also Web and agent based, hence allowing agents to collaborate and design on top of the Web.

Furthermore, we think that combining SOA with genericness and synchronicity – also focusing on reuse and optimization – is a good path to follow for a collaborative system that intends to be flexible and effective. SOA is a recent architectural paradigm that has received much attention and combines the capacity to invoke remote services with standardized mechanisms for dynamic and universal service discovery and execution. The synchronical approach lets the actors take part in all the phases of the design (from the beginning until the final design is done), and also access other designers' preliminary versions during the whole process, in order to refine their models in advance before achieving a final version of the product design. Reuse means skipping and enhancing the initial phases of design by employing final models that have already been used (and implemented) in other projects as a starting point. To increase the performance of the design, and consequently to satisfy

customers' requirements and expectations, the decision-makers (usually the project managers) have to adapt the designers' work context to the environment of the design process. The work context of the actors is improved and, when the project manager is able to create effective working groups according to the design objectives, the allocation of human resources is more efficient. It is also important to identify factors that create good state of mind and motivation in the group. With the actor placed in a central position, we have to integrate many human aspects to describe the situation of the design properly. The objective is to provide project managers with mechanisms for an efficient capitalization and reuse of pertinent information from previous projects. Each actor will develop his/her own cognitive process to transform his/her scientific and technical knowledge into tangible results on the product.

In Chapter 6, we have presented our methodology for handling conflicts, along with the EXP2OWL prototype and our proposed mapping between EXPRESS and OWL. A great challenge regarding the management of information systems is that much of the knowledge available inside organizations can only be found in a non-structured form. As a consequence, one of the major problems faced by the industrial segment (including capital goods industries) is the low degree of interoperability. This problem is even more serious when considering the whole product lifecycle, where several software applications are involved in the organization of the PLM.

We consider the use of OWL as a very efficient approach to represent knowledge in collaborative environments. Besides, OWL reasoning allows for coping with conflicts in the early stages of the design, as it permits the detection of inconsistencies. The latest trends in this research domain – along with several ongoing projects working with ontology merging and aligning – have encouraged us to keep going in this direction. Our results have proved our expectations in this area. We believe that in order to remain competitive, companies and organizations need to engage in more sophisticated business networks and improve collaboration. Information and communication technologies combined with the use of open standards can be a very powerful tool to improve enterprise competitiveness. In order to better manage the life cycle of services and products, the use of standards allows the development of flexible and integrated environments, in which software applications could be easily integrated independently of the platform being used. However, the adoption of standards is not always straightforward due to many problems that may arise, such as the integration with legacy systems and the unfamiliarity with the required technologies. To overcome these problems, we believe that the translation of the already well-accepted standards – like EXPRESS and OWL – is a good strategy to follow. EXPRESS – STEP modeling language is unfamiliar to most application developers. Although it is a powerful language, it has remained relatively unknown in the world of generic software modeling tools and software engineers. As opposed to other modeling technologies, such as UML, few software systems support the capture, modeling and interpretation of EXPRESS constructs and relationships. Consequently, implementing a tool for translating from EXPRESS to OWL was desirable and suitable for enhancing the formalization process within our architecture.

Regarding the integration of ontologies, our proposal relies on an OWL representation of data/information. Two different scenarios are then considered. The first one is where a "generic" ontology is given, i.e., defined before the beginning of the collaboration process. In this context, designers should take this generic ontology and use it to build their particular models. In the second scenario, we start from different kinds of representation. Here, each designer has his/her particular model for representing content, namely the Function-Behavior-Structure framework (Gero et al., 2004), STEP-based models, among others. In this case, a

harmonization process is undertaken in order to merge these different ontologies into a common one to be used as the generic ontology for the project. The harmonization process, which uses semantic morphisms to merge the ontologies, is a pre-collaboration stage that uses interaction between the ontology experts and the designers. Based on individual agents and their views of one another, we develop a common model that emerges from the interaction of individual experts. The ontology experts and the designers can usually establish the new generic ontology without much effort since they can build upon the generalized expertise/knowledge gained from previously working in similar projects. To build a new ontology to be the reference in the domain, this group uses the ontology merging and mapping approaches, thus combining the existing models in order to create a common one to be used by all the participants of the design project.

In Chapter 7, we have shown our method for conflict resolution. Conflict is the disagreement between two or more agents due to different design approaches or parameter setups. We think that an effective identification of conflict causes facilitates the understanding of conflict behavior and helps designers apply appropriate methods to resolve them. Our proposal for conflict resolution intends to formally express definitions and associate them to terms and viewpoints. It includes a concept comparison operation to highlight definition conflicts and their nature, and other operations (derivation, intersection, union, etc.) to solve the detected conflicts. Human design agents commonly prefer communicating in natural language rather than formal language, which is often considered hard to understand and thus avoided. Through the use of standardized product models, information is shareable and suitable for reuse and feedback, and the more often this information is reused, the more general and adaptable it becomes. Our proposal also enables users to express and record issues, positions, arguments and endorsements that occur during the conflict resolution.

The method for conflict resolution proposed in this work comprises two sub-approaches: the automatic solving and the negotiation process. The automatic solving approach consists of priority definition, case-based reasoning, and rule-based reasoning. The idea here is first to try solving the conflict in a transparent way, so that the involved users do not have to be directly involved in the process. In order to assist this process, priorities can be defined by the project manager at any time during the design process, as well as constraints and (SWRL) rules. In addition, the system log and the knowledge base are used during the case-based reasoning.

The negotiation process involves several iterations for explanation and argumentation of the concerned parties, in order to refine the conflict causes. Once these causes are well identified and the actors have agreed upon them, a solution – concerning the modifications to be made on the product model – is proposed. This solution is then evaluated by all the involved parties. Further steps consist of checking the consistency of the proposed solution, in order to define a resulting model free of problems.

Finally, in Chapter 8, we have presented Scoop – our prototype for collaborative design. We have taken the modeling of an electrical connector as the scenario for running this prototype.

9.1 Perspectives and Future Research

Collaborative design can be used to either increase the product quality/performance for fixed development time or reduce the development time and effort without explicit consideration of product quality/performance issues. Without careful management of the overlap among design activities, the development lead-time and effort may increase. Besides, scheduling and managing emerging activities during the collaborative design process is a difficult exercise that requires much effort and experience. Such emerging activities could occur following the resolution of conflicts. We consider that future work will include assessing the importance of each product data input which could be done by allocating different weights to each of them. This would be very useful to coordinate the data exchange in the case where an activity has several inputs from different activities. The project manager would reschedule the process accordingly.

Issue-based systems give us another strategy to be used. They aim at providing explicit representation to capture deliberative aspects of the design decision-making process. This representation describes design issues and arguments in favor of or against the proposed design actions. These systems have helped explain the deliberative nature of the design process; however, they suffer from the inherent difficulty of encoding design knowledge in computational constructs. Hence this issue should also be tackled.

Another step would be to provide designers with flexibility of methodology use, i.e., designers should be able to choose the methodology to be used during the processes of ontology integration and conflict resolution. However, this strategy should be considered only when really experienced users take part in the design project. Because though a correct choice of methodology can provide more accurate outputs, a wrong choice can hamper the whole process of integrating ontologies and solving conflicts.

The formalization process can be ameliorated by including rules as output files. In the case of the EXP2OWL prototype, for instance, files containing SWRL rules could be generated along with the OWL files that represent the EXPRESS schemas. This could be suitable for representing EXPRESS domain rules and the uniqueness feature, as well as attenuating some possible loss of information during the translation, as OWL is a declarative language and, consequently, cannot be used to represent function statements.

Furthermore, contexts should be taken into consideration in future works. Contexts are "the interrelated conditions in which something exists or occurs"[27]. A context-based approach could improve interoperability between heterogeneous ontologies developed and evolving autonomously. During the collaborative process, organizations exchange their data whose meaning is described by concepts defined by ontologies. Hence the process of generating a generic ontology could be done under a context-based approach, in order to enhance the result of the merging. This process could disambiguate possible semantic problems by analyzing the perspectives from which the original concepts have been developed, and also taking into consideration each agent's context.

[27] http://www.merriam-webster.com/

Some efforts toward the reduction of the product complexity are also desirable. Products and product development processes have become significantly more complex in recent years. This reflects not only the product complexity but also the multiple stakeholders that take part in the design process. We consider that this is a major problem to be faced in the coming years.

We believe that the development of multingual resources is another good step to be taken. Though English is commonly agreed to be the *lingua franca* of the world, much effort is being spent to preserve other languages. Keeping expressions/concepts belonging to different cultures and languages is important to preserve and respect the cultural heritage of every country. At the same time, the role of multilinguality is to make ontological information explicitly accessible in different languages, thus reducing the cost of mediating its content across different domains and languages.

And last but not least, we consider that further projects in collaborative design should pay attention to the cloud computing paradigm. Cloud computing is a recent trend in the information technology area, which intends to move computing and data away from the desktop and portable PCs into large data centers. The current large corporate data centers' capacity of storage, the ubiquity of broadband and wireless networking, the falling cost of storage, and the progressive improvement of Internet technologies and infrastructure contribute to make this paradigm a very promising approach for the coming years.

9.1.1 Improvement of Scoop

Further steps also encompass the integration of Scoop with some visual CAD tools (e.g. a 3D viewer), in order to enhance the collaboration especially during the structure definition stage. A 3D viewer is used to assist normal users, without CAD software, to be able to view the 3D modeling in real time. In order to allow non-CAD users to communicate with CAD designers or view 3D models, the system would incorporate some tool to provide an online 3D model visualization. This tool should make data visible to everyone without installing a CAD system. The users could not only view a 3D model, but also add predefined views of their viewpoint, assign colors to the model, zoom in on details, add bitmaps or text annotations, and draw online as an interactive 3D slide show, creating several scenes. Each scene could provide text notes and different views of the same file.

A virtual conference room could also be aggregated into the system, as it represents a convergence of video, audio and real-time collaborative environments to support real-time communications and images among team participants. When team participants enter the online conference room, they would be able to assemble a meeting. The virtual conference room would help global team participants enhance collaboration by a virtual face-to-face conference. Dispersed team participants would view, comment on, and evaluate feature-rich 3D design data as well as many 2D formats in real time. The virtual conference room is intended to provide an environment with a set of tools that allows team participants to work in collaborative situations. The virtual conference room would encompass electronic whiteboards, instant messaging, online conference techniques, and a 3D viewer to assist team participants in discussing or modifying projects through the Internet.

Another approach to be used in order to assist team participants is the online detection of team members. Whenever the users log in, the system would detect the online status of everybody. An online list displaying logins, time, e-mail, etc., should be integrated into the agents'

workspaces. From this list, each team participant would be aware of who is logging into the system.

Finally, we also think that it is very important to adapt Scoop in order to enable it to handle natural-language requirements. A number of projects have appeared in recent years within this research area, and some interesting propositions have been made.

References

Adelson, B. (1999). *Developing strategic alliances: A framework for collaborative negotiation in design*. Research in Engineering. Design, Springer-Verlag: London, 11(3), pp. 133–144, 1999.

Agostinho C., Delgado M., Steiger-Garção A., Jardim-Gonçalves R. (2006). *Enabling Adoption of STEP Standards Through the Use of Popular Technologies*. In Proceedings of the 13th ISPE International Conference on Concurrent Engineering (CE2006): Research and Application, Antibes – Juan-Les-Pins, France, September 2006.

AlMellor et al. (2004). *Introduction to Model Driven Architecture*. Addison-Wesley, ISBN: 0-201-78891-8, 2004.

Alvares A. J., Ferreira J.C.E. (2008). *A system for the design and manufacture of feature-based parts through the Internet*. The International Journal of Advanced Manufacturing Technology, vol. 35, pp. 646–664, 2008.

Baader F., Lutz C., Sturm H., Wolter F. (2003). *Basic description logics*. Journal of Logic and Computation, 2003.

Badr N., Reilly D., TalebBendiab A. (2002). *A Conflict Resolution Control Architecture For Self-Adaptive Software*. In Proceedings of the International Workshop on Architecting Dependable Systems (WADS/ICSE 2002), Orlando, Florida, USA, May 2002.

Bao J., Caragea D., Honavar V. (2006). *Towards Collaborative Environments for Ontology Construction and Sharing*. In Proceedings of the International Symposium on Collaborative Technologies and Systems, pp. 99 – 108, Las Vegas, Nevada, USA, May 2006.

Barber K.S., Liu T.H., Ramaswamy S. (2001). *Conflict Detection During Plan Integration for Multi-Agent Systems*. IEEE Transactions on Systems, Man, and Cybernetics – Part B: Cybernetics, Vol. 31, No. 4, August 2001.

Benslimane D., Arara A., Falquet G., Maamar Z., Thiran P., Gargouri F. (2006). *Contextual Ontologies - Motivations, Challenges, and Solutions*. In Proceedings of the 4th International Conference (ADVIS 2006), Izmir, Turkey, October 2006.

Berners-Lee T., Hendler J., Lassila O. (2001). *The semantic web*. Scientific American, 284(5), May 2001.

Bhamra T., Lofthouse V. (2007). *Design for sustainability: a practical approach Gower*. ISBN: 978-0-566-08704-2, UK, 2007.

Bilek J., Hartmann D. (2006). *Agent-Based Collaborative Work Environment for Concurrent Structural Design Process*. Joint International Conference on Computing and Decision Making in Civil and Building Engineering, Montréal, Canada, June 2006.

Bloehdorn S., Haase P., Huang Z., Sure Y., Völker J., van Harmelen F., Studer R. (2009). *Ontology Management*. In J. Davies et al. (eds.), Semantic Knowledge Management, Springer Berlin Heidelberg, 2009.

Brambilla M., Ceri S., Facca F.M., Celino I., Cerizza D., Della Valle E. (2007). *Model-Driven design and development of semantic Web service applications*. ACM Transactions on Internet Technology, Vol. 8, Issue 1, Article 3, November 2007.

Brockman J.B., Sutton P.R., Director S.W. (1996). *A Description Language for Design Process Management*. In Proceedings of 33rd DAC, 1996.

Burgess G., Burgess H. (1999). *International online training program on intractable conflict.* Conflict Research Consortium, 1999. Available online at: www.colorado.edu/conflict/peace/index.html, last access in March 2009.

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

Chen X., Li M., Gao S. (2005). *A Web Services Based Platform for Exchange of Procedural CAD Models*. In proceedings of the 9th International Conference on Computer Supported Cooperative Work in Design, Coventry, United Kingdom, Vol. 1, pp. 605-610, 2005.

Chi Y.-L. (2009). *Rule-based ontological knowledge base for monitoring partners across supply networks*. Expert Systems with Applications, 2009.

Chira C., Brennan A., Roche T., Tormey D., Chira O. (2003). *The Use of Ontologies for refining Collaborative Design Processes*. In Proceedings of the 32nd International Conference on Computers and Industrial Engineering, University of Limerick, Ireland, 2003.

Chungoora N., Young R.I.M. (2008). *Ontology Mapping to Support Semantic Interoperability in Product Design and Manufacture*. In Proceedings of MDISIS'08 – Model Driven Interoperability for Sustainable Information Systems, Montpellier, France, June 2008.

Cointe C. (1998). *Support to the Management of Collaborative Design Conflicts in a Distributed System* [in French]. PhD thesis, University of Montpellier II, Montpellier, France, 1998.

Cooper S., Taleb-Bendiab A. (1998). *CONCENSUS: Multi-party negotiation support for conflict resolution in concurrent engineering design*. Journal of Intelligent Manufacturing, 8, 9(2), Chapman and Hall, pp. 155–159, 1998.

Correndo G., Alani H. (2008). *Collaborative Support for Community Data Sharing*. 2nd Workshop on Collective Intelligence in Semantic Web and Social Networks, Sydney, Australia, December 2008.

Cutkosky M., Mori T. (1998). *Agent-base collaborative design of parts in assembly*. In Proceedings of ASME Design Engineering Technical Conferences, 1998.

De Bruijn J., Ehrig M., Feier C., Martín-Recuerda F., Scharffe F., Moritz Weiten M. (2006). *Ontology mediation, merging and aligning*. In Semantic Web Technologies, Wiley, 2006.

De Nicola A., Missikoff M., Navigli R. (2005). *A proposal for a Unified Process for Ontology building: UPON*. In Proceedings of DEXA 05, International Conference on Database and Expert Systems Applications, pp. 655-664, Copenhagen, Denmark, 2005.

Delgado M., Agostinho C., Gonçalves R. (2006). *Taking advantage of STEP, MDA, and SOA to push SMEs towards a Single European Information Space*. In Proceedings of eChallenges e2006, Barcelona, Spain, October 2006.

Description Logics Reasoners (2009). WWW page. Available at: http://www.cs.man.ac.uk/~sattler/reasoners.html, last access in August 2009.

Ding L., Davies D., McMahon C.A. (2009). *The integration of lightweight representation and annotation for collaborative design representation*. *Research in Engineering Design*, vol. 19, issue 4, pp. 223-238, 2009.

Dong X.L., Halevy A., Yu C. (2009). *Data integration with uncertainty*. The VLDB Journal vol. 18, pp. 469–500, 2009.

Dutra M., Slimani K., Ghodous P. (2006). *A Distributed Architecture for Collaborative Design*. In Proceedings of the 13th ISPE International Conference on Concurrent Engineering (CE2006): Research and Application, Antibes – Juan-Les-Pins, France, September 2006.

Dutra M., Agostinho C., Jardim-Gonçalves R., Ghodous P., Steiger-Garção A. (2007). *EXPRESS to OWL morphism: making possible to enrich ISO10303 Modules*. In Proceedings of the 14th ISPE International Conference on Concurrent Engineering (CE2007), pp. 391-402, São José dos Campos, Brazil, July 2007.

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

Dutra M., Ferreira da Silva C., Ghodous P., Gonçalves R. (2008). *Using an Inference Engine to Detect Conflicts in Collaborative Design*. In the Proceedings of the 14th International Conference on Concurrent Enterprising (ICE 2008) – Lisbon, Portugal, June 2008.

Dutra M., Ghodous P., Gonçalves R. (2008b). *Resolving Collaborative Design Conflicts Through an Ontology-based Approach*. In Proceedings of the 15th ISPE International Conference on Concurrent Engineering (CE 2008), Belfast, Northern Ireland, August 2008.

Dutra M., Kuhn O., Ghodous P., Tri N.M. (2009). *A Heterogeneous and Synchronous Ontology-based Architecture for Collaborative Design*. Submitted to the International Journal of Concurrent Engineering: Reasearch & Applications (CERA), 2009.

Easterbrook S.M., Beck E.E., Goodlet J.S., Plowman L., Sharples M., Wood C.C. (1993). *A Survey of Empirical Studies of Conflict*. CSCW: Cooperation or Conflict?, Springer-Verlag New York, 1993.

Eiter T., Ianni G., Krennwallner T., Polleres A. (2008). *Rules and Ontologies for the Semantic Web*. In C. Baroglio et al. (Eds.): Reasoning Web, pp. 1–53, 2008.

Falconer S.M., Noy N.F., Storey M.-A. (2007). *Ontology Mapping - A User Survey*. Ontology Matching Workshop (OM2007), held in connection with the 6th International Semantic Web Conference and the 2nd Asian Semantic Web Conference, Busan, South Korea, November 2007.

Falquet G., Mottaz Jiang C-L. (2000). *Conflict Resolution in the Collaborative Design of Terminological Knowledge Bases*. In Proceedings of the 12th European Workshop on Knowledge Acquisition, Modeling and Management, Juan-les-Pins, France, October 2000.

Fedrizzi M., Kacpryzk J., Zadrozny, S. (1998). *An Interactive Multi-user Decision Support System for Consensus Reaching Using Fuzzy Logic with Linguistic Quantifiers*. Vol. 4, pp. 313–327, Elsevier Science Publishers: Amsterdam, 1998.

Ferber J. (1999). *Multi-Agent Systems: An Introduction to Distributed Artificial Intelligence.* Addison-Wesley Professional, Harlow, Essex, UK.

Ferreira da Silva C. (2003). *Knowledge Management and Conflict Resolution in a Collaborative Environment* [in French]. M.Sc. Thesis. Laboratory of Computer Graphics, Images, and Information Systems (LIRIS), University of Lyon 1, Lyon, France, 2003.

Ferreira da Silva C. (2007). *Discovery of Semantic Correspondences Between Heterogeneous Resources in a Collaborative Environment* [in French]. Ph.D. Thesis. Laboratory of Computer Graphics, Images, and Information Systems (LIRIS), University of Lyon 1, Lyon, France, 2007.

Ferreira da Silva C. ; Médini L. ; Ghodous P. (2004). *Conflict Attenuation in Collaborative Design* [in French]. In the 15èmes Journées Francophones d'Ingénierie des Connaissances (IC'2004), pp. 127-138. University Press of Grenoble, Grenoble, France, 2004.

Figay N., Ghodous P. (2008). *Collaborative Product Development: EADS Pilot Based on ATHENA. Enterprise Interoperability III – New Challenges and Industrial Approaches*. Springer London, 2008.

Frost, R.A. (2006). *Realization of natural language interfaces using lazy functional programming*. ACM Computing Surveys, Vol. 38, Issue 4, Article 11, December 2006.

Fulczyk M., Florkowski M., Karandikar H., Nowak T., and Banas M. (2002). *Synchronous collaborative design system functionality and architecture*. In Proceedings of the 9th International Conference on Concurrent Engineering (CE'02): Advances in Concurrent Engineering, Cranfeld University, UK, 2002.

Gero J.S., Kannengiesser U. (2004). *The situated function–behaviour–structure framework*. Design Studies, vol. 25, pp. 373–391, 2004.

Gero J.S., Smith G.J. (2009). *Context, Situations, and Design Agents*. Knowledge-Based Systems (*to appear*).

Ghafour S.A. (2009). *Semantic Interoperability of Feature-based Model Knowledge* [in French]. Ph.D. Thesis. Laboratory of Computer Graphics, Images, and Information Systems (LIRIS), University of Lyon 1, Lyon, France, 2009.

Ghodous P. (2002). *Models and Architectures for Cooperative Engineering* [in French]. 'Habilitation' Thesis, Laboratory of Computer Graphics, Images, and Modelization, University of Lyon 1, Lyon, France, 2002.

Ghodous P., Martinez M., Hassas S., Pimont S. (2002). *Distributed architecture for design activities*. International Journal of IT in Architecture, Engineering and Construction, Milpress, 2002.

Gulla J.A. (2008). *Experiences with Industrial Ontology Engineering*. In Proceedings of the 10th International Conference on Enterprise Information Systems. Barcelona, Spain, June 2008.

Gzara Yesilbas L., Rose B., Lombard M. (2006). *Specification of a repository to support collaborative knowledge exchanges in IPPOP project*. Computers in Industry 57, pp. 690–710, Elsevier B.V., 2006.

Hepp M. (2006). *The True Complexity of Product Representation in the Semantic Web*. In Proceedings of the 14th European Conference on Information System (ECIS 2006), Gothenburg, Sweden, June 2006.

Hoffmann P. (2008). *Context-based Semantic Similarity across Ontologies*. Ph.D. Thesis. Laboratory of Computer Graphics, Images, and Information Systems (LIRIS), University of Lyon 1, Lyon, France, 2008.

Hooey B.L. (2006). *Requirements for a Design Rationale Capture Tool to Support*. NASA paper, USA, 2006.

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

Horrocks I., Patel-Schneider P.F., Boley H., Tabet S., Grosof B., Dean M. (2004). *SWRL: A Semantic Web Rule Language Combining OWL and RuleML*. http://www.w3.org/Submission/SWRL, last access in August 2009.

Hu J., Peng Y., Xiong G. (2007). *Parameter coordination and optimization for collaborative design based on the constraints network*. International Journal of Advanced Manufacturing Technology 32, pp. 1053–1063, Springer-Verlag London, 2007.

Huang C.-C., Tseng T.-L., Kwonc Y., Choua Y. Y. (2004). *An Agent-based Web Services Solution to Collaborative Product Design*. In Proceedings of SPIE, vol. 5605, Bellingham, WA, USA, 2004.

Huang S., Fan Y. (2007). *Web-Based Engineering Portal for Collaborative Product Development*. In Proceedings of the 4th International Conference, CDVE 2007, Shanghai, China, September 2007.

Jennings N.R., Parsons S., Noriega P., and Sierra C. (1998). *On argumentation-based negotiation*. In Proceedings of the Intl Workshop on Multi-Agent Systems: 1-7, 1998.

Jiao Y., Wu B., Zhu K., Yu Q. (2006). *Towards a Systematic Conflict Resolution Policy in Multi-agent System: A Conceptual Framework*. In Proceedings of the 9th Computer Supported Cooperative Work in Design Conference (CSCWD 2005), vol. 3865, pp. 274-283, Coventry, United Kingdom, 2006.

Jin Y., Geslin M. (2008). *Roles of Negotiation Protocol and Strategy in Collaborative Design*. In Proceedings of the 3rd International Conference on Design Computing and Cognition (DCC'08), Georgia Institute of Technology, Atlanta, USA, June 2008.

Jussien N., Boizumault P. (1996). *CSP Dinamics and Constraint Relaxation* [in French]. In Proceedings of CNPC'96, 1996.

Kalfoglou Y., Schorlemmer M. (2003). *Ontology Mapping: The State of the Art*. The Knowledge Engineering Review, Vol. 18, Issue 1, pp. 1 – 31, 2003.

Kalyanpur A. (2006). *Debugging and Repair of OWL Ontologies*. Ph.D. Thesis, Faculty of the Graduate School of the University of Maryland, USA, 2006.

Kannengiesser U., Gero, J.S. (2009). *An ontology of computer-aided design*. In CM De Smet and JA Peeters (eds), Computer-Aided Design Research and Development, Nova Science Publishers, 2009.

Kim K.-Y., Manley D.G., Yang H. (2006). *Ontology-based assembly design and information sharing for collaborative product development*. Computer-Aided Design, vol. 38, pp. 1233–1250, 2006.

Klein M. (2000). *Towards a Systematic Repository of Knowledge About Managing Collaborative Design Conflicts*. In Proceedings of the 6th International Conference on Artificial Intelligence in Design, Worcester, MA, USA, June 2000.

Klein M., Sayama H., Faratin P., Bar-Yam Y. (2003). *The Dynamics of Collaborative Design: Insights From Complex Systems and Negotiation Research*. Concurrent Engineering Research & Applications, 2003.

Klein M., Sayama H., Faratin P., Bar-Yam Y. (2006). *The Dynamics of Collaborative Design: Insights from Complex Systems and Negotiation Research*. Complex Engineered Systems, Springer Berlin / Heidelberg, pp. 158-174, 2006.

Kotis K., Vouros G.A., Stergiou KI. (2004). *Capturing Semantics Towards Automatic Coordination of Domain Ontologies*. In Proceedings of the 11th International Conference on Artificial Intelligence: Methodology, Systems, and Applications (AIMSA 2004), pp. 22–32, Varna, Bulgaria, September 2004.

Kotis K., Vouros G. (2007). *Semantic Integration and Interoperability among Portals*. Encyclopedia of Portal Technology and Applications . Arthur Tatnall (Ed.) Hershey, PA: Idea Group, 2007.

Kuhn O., Dutra M., Ghodous P., Dusch T., Collet P. (2008). *Collaborative Architecture Based on Web-Services*. In Proceedings of the 15th ISPE International Conference on Concurrent Engineering (CE 2008), Belfast, Northern Ireland, August 2008.

Li W.D., Lu W.F., Fuh J.Y.H., and Wong Y.S. (2005). *Collaborative computer-aided design —research and development status.* Computer-Aided Design 37, pages 931–940, 2005.

Li W.D., Ong S.K., Nee A.Y.C., McMahon, C.A. (2007). *Collaborative Product Design and Manufacturing Methodologies and Applications. Springer*, 2007.

Lilley D. (2009). *Design for sustainable behaviour: strategies and perceptions*. Design Studies, doi:10.1016/j.destud.2009.05.001, 2009.

Liu X.Q., Raorane S., Leu M.C. (2007). *A web-based intelligent collaborative system for engineering design*. In Li, W.D., Ong, S.K., Nee, A.Y.C. and McMahon C.A. (ed.) Collaborative Product Design and Manufacturing Methodologies and Applications, Springer, pp. 37-58, 2007.

Lombard M., Gzara Yesilbas L. (2006). *Towards a framework to manage formalised exchanges during collaborative design*. Mathematics and Computers in Simulation 70, pp. 343–357, Elsevier B.V., 2006.

López-Ortega O., López-Morales V., Villar-Medina I. (2008). *Intelligent and collaborative Multi-Agent System to generate and schedule production orders*. Journal of Intelligent Manufacturing, vol. 19, pp. 677–687, 2008.

Lottaz C., Smith I.F.C., Robert-Nicoud Y., and Faltings B.V. (2000). *Constraint-based support for negotiation in collaborative design*. Artificial Intelligence in Engineering, Volume 14, Issue 3, Pages 261-280, Elsevier Science Ltd, July 2000.

Lu S.C.-Y., Cai J., Burkett W., Udwadia F. (2000). *A Methodology for Collaborative Design Process and Conflict Analysis.* CIRP Annals - Manufacturing Technology, Volume 49, Issue 1, pp. 69-73, 2000.

Mahdjoub M., Monticolo D., Gomes S., Sagot J.-C. (2009). *A collaborative Design for Usability approach supported by Virtual Reality and a Multi-Agent System embedded in a PLM environment*. Computer-Aided Design, doi:10.1016/j.cad.2009.02.009, 2009.

Mahesh M., Ong S.K., Nee A.Y.C. (2007). *A Web-based Framework for Distributed and Collaborative Manufacturing*. Collaborative Product Design and Manufacturing Methodologies and Applications, Springer London, pp. 137-150, 2007.

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

McGuinness D., van Harmelen F. (2004). *OWL web ontology language*. Technical report, World Wide Web Consortium (W3C). http://www.w3.org/TR/2004/REC-owl-features-20040210/, last access in August 2009.

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

Meng X., Liu W., Xu Y. (2006). *Real-Time Collaborative Design System For Product Assembly Over The Internet*. In IFIP International Federation for Information Processing, Volume 220, Information Technology for Balanced Manufacturing Systems, pp. 253-260, Santiago de Chile, August 2006.

Mervyn F., Senthil Kumar A., Nee A.Y.C. (2007). *A 'plug-and-play' computing environment for collaborative product design and manufacturing across an extended enterprise*. In Li, W.D., Ong, S.K., Nee, A.Y.C. and McMahon C.A. (ed.) Collaborative Product Design and Manufacturing Methodologies and Applications, Springer, pp. 71-92, 2007.

Michel Klein (2001). *Combining and relating ontologies: an analysis of problems and solutions*. In Asuncion Gomez-Perez, Michael Gruninger, Heiner Stuckenschmidt, and Michael Uschold, editors, Workshop on Ontologies and Information Sharing, IJCAI'01, Seattle, USA, August 2001.

Mohammed J., May J., Alavi A. (2008). *Application of Computer Aided Design (CAD) in Knowledge Based Engineering*. In Proceedings of the 2008 IAJC-IJME International Conference, Paper 83, Nashville, Tennessee, USA, November 2008.

Movahed-Khah R., Ostrosi E., Garro O. (2009). *Analysis of interaction dynamics in collaborative and distributed design process*. Computers in Industry, doi:10.1016/j.compind.2009.05.007, 2009.

Mühlenfeld A., Maier F., Mayer W., Stumptnera M. (2008). *Modelling and Management of Design Artefacts in Design Optimisation*. In Proceedings of the 15th ISPE International Conference on Concurrent Engineering, Belfast, Northern Ireland, August 2008.

Nahm Y.-E., Ishikawa H. (2006). *An Internet-based integrated product design environment. Part II: its applications to concurrent engineering design*. The International Journal of Advanced Manufacturing Technology, vol. 27, pp. 431–444, 2006.

Noy N.F. (2004). *Semantic Integration: A Survey Of Ontology-Based Approaches*. SIGMOD Record, Vol. 33, No. 4, December 2004

Noy N.F., Musen M.A. (2000). *Prompt: Algorithm and tool for automated ontology merging and alignment*. In Proceedings if the 17th National Conference on Artificial Intelligence (AAAI2000), Austin, Texas, USA, July/August 2000.

Noy N.F., McGuinness D.L. (2001). *Ontology Development 101: A Guide to Creating Your First Ontology*. Stanford Knowledge Systems Laboratory Technical Report KSL-01-05 and Stanford Medical Informatics Technical Report SMI-2001-0880, March 2001.

Ölvander J., Lundéna B., Gavel H. (2009). *A computerized optimization framework for the morphological matrix applied to aircraft conceptual design*. Computer-Aided Design, vol. 41, pp. 187-196, 2009.

Ouertani M.Z., Gzara L. (2007). *Tracking product specification dependencies in collaborative design for conflict management*. Computer-Aided Design, doi:10.1016/j.cad.2007.07.002, 2007.

Pahl C. (2007). *Semantic model-driven architecting of service-based software systems*. Information and Software Technology, vol. 49, pp. 838–850, 2007.

Paschke A., Boley H., Kozlenkov A. (2007). *Rule Responder: RuleML-Based Agents for Distributed Collaboration on the Pragmatic Web*. In Proceedings of the 2nd International Conference on the Pragmatic Web, Tilburg, The Netherlands, October 2007.

Pazienza M.T., Stellato A. (2005). *Linguistically motivated Ontology Mapping for the Semantic Web*. SWAP 2005 – 2nd Italian Semantic Web Workshop, Trento, Italy, December 2005.

Pirró G., Mastroianni C., Talia D. (2009). *A framework for distributed knowledge management: Design and Implementation*. Future Generation Computer Systems, doi:10.1016/j.future.2009.06.004, 2009.

Predoiu L., Feier C., Scharffe F., de Bruijn J., Martín-Recuerda F., Manov D., Ehrig M. (2006). *D4.2.2 State-of-the-art survey on Ontology Merging and Aligning V2*. EU-IST Integrated Project (IP) IST-2003-506826 SEKT, Deliverable D4.2.2 (WP4), 2006.

Rahm E., Bernstein P.A. (2001). *A survey of approaches to automatic schema matching*. VLDB Journal: Very Large Data Bases, 10(4):334.350, 2001.

Raiffa H., Richardson J., Metcalfe D. (2002). *Negotiation analysis: the science and art of collaborative decision making*. The Belknap Press of Harvard University Press, 2002.

Qin S.F., Sun G. (2006). *Analysis and control of complex collaborative design systems*. International Journal of General Systems, Vol 35 (3), pp. 377-386, 2006.

Qiu X. (2005). *Agent Interaction in a Semantic Web Environment : A State-of-the-art Survey and Prospects in Knowledge Mobilization*. In Proceedings of Information Systems Research in Scandinavia (IRIS 28), Kristiansand, Norway, 2005.

Robin V., Rose B., Girard P. (2007). *Modelling collaborative knowledge to support engineering design project manager*. Computers in Industry 58, pp. 188–198, 2007.

Rose B., Robin V., Girard P., Lombard M. (2007). *Management of engineering design process in collaborative situation*. International Journal of Product Lifecycle Management, vol. 2, n°1, pp. 84-103, 2007.

Roummieh Y., Ghodous P., Vanehms P., Verdier C., Dutra M. (2007). *Collaborative Environment for the Detection and the Follow-up of Legionella*. In Proceedings of the 1st International Workshop on Methods and Design models for Health Information Systems (MeDeHIS'07), in conjunction with The 2nd IEEE International Conference on Digital Information Management (ICDIM'2007), Lyon, France, October 2007.

Sakao T., Shimomura Y., Sundin E., Comstock M. (2009). *Modeling design objects in CAD system for Service/Product Engineering*. Computer-Aided Design, vol. 41, pp. 197-213, 2009.

Sarraipa J., Silva J.P.M.A., Jardim-Goncalves R., Monteiro A.A.C. (2008). *MENTOR – A Methodology for Enterprise Reference Ontology Development*. In Proceedings of the 4th International IEEE Conference (IS'08), vol. 1, pp. 632-640. Varna, Bulgaria, September 2008.

Schorlemmer M., Kalfoglou Y. (2008). *Institutionalising Ontology-Based Semantic Integration*. Applied Ontology, Volume 3, Issue 3, pp. 131-150, August 2008.

Shen W., Wang L. (2003). *Web-based and agent-based approaches for collaborative product design: an overview*. International Journal of Computer Applications in Technology, 16(2/3): pp. 103-112, 2003.

Shen W., Hao Q., Li W. (2008). *Computer supported collaborative design: retrospective and perspective*. National research Council, Canada, Computers in Industry, v. 59, no. 9, pp. 855-862, December 2008.

Simperl E. (2009). *Reusing ontologies on the Semantic Web: A feasibility study*. Data and Knowledge Engineering, January 2009.

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, 2007.

Slimani K. (2006). *A System for Knowledge Exchanging and Sharing in Collaborative Design* [in French]. Ph.D. Thesis. Laboratory of Computer Graphics, Images, and Information Systems (LIRIS), University of Lyon 1, Lyon, France, September 2006.

Slimani K., Ferreira da Silva C., Médini L., Ghodous P. (2006). *Conflict mitigation in collaborative design*. International Journal of Production Research, Vol. 44, No. 9, pp. 1681–1702, May 2006.

Sprow E. (1992). *Chrysler's concurrent engineering challenge*. Manufacturing Engineering, 108(4): 35-42, 1992.

Sriram, R.D. (2002). *Distributed and Integrated Collaborative Engineering Design*. Savren, 2002.

STEP Part 11 (2002). *ISO10303-11 – Product data representation and exchange—Part 11: Description methods: The EXPRESS language reference manual*, 2002.

Sun C., Chen D. (2002). *Consistency maintenance in real-time collaborative graphics editing systems*. ACM Transactions on Computer-Human Interaction 9(1), pp. 1–41, 2002.

Sycara K. (1994). *The PERSUADER project, 1994*. Available online at: www-2.cs.cmu.edu/afs/cs.cmu.edu/user/katia/www/persuader.html, last access in March 2009.

Systematic Design (2009). http://www.appropedia.org/Systematic_design, last access in June 2009.

Toinard C. (2001). *Distributed Systems for Collaborative Virtual Prototyping and Control of Industrial Processes* [in French]. 'Habilitation' Thesis, Université Bordeaux 1, Bordeaux, France, 2001.

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

Wang Y., Shen W., Ghenniwa H. (2003). *WebBlow: a Web/agent-based multidisciplinary design optimization environment*. Computers in Industry, 52(1): pp. 17-28, 2003.

Wang J.X., Tang M.X., Song L.N., Jiang S.Q. (2009). *Design and implementation of an agent-based collaborative product design system*. Computers in Industry, vol. 60, pp. 520–535, 2009.

Yu Q., Liu X., Bouguettaya A., Medjahed B. (2008). *Deploying and managing Web services: issues, solutions, and directions*. The VLDB Journal, vol. 17, pp. 537–572, 2008.

Yuan-Lung L. (2007). *A constraint-based system for product design and manufacturing*. Robotics and Computer-Integrated Manufacturing, Volume 25, Issue 1, Pages 246-258, 2007.

Zhang W.Y., Yin J.W. (2008). *Exploring Semantic Web technologies for ontology-based modeling in collaborative engineering design*. The International Journal of Advanced Manufacturing Technology, vol. 36, pp. 833–843, 2008.

Zheng Y., Shen H., Xia S., Sun C. (2007). *Conflict Resolution of Boolean Operations by Integration in Real-Time Collaborative CAD Systems*. In Proceedings of OTM 2007 - On the Move to Meaningful Internet Systems, pages 238-252, Vilamoura, Portugal, November 2007.

Zheng Y., Shen H.F., and Sun C. (2008). *Adapting Single-user AutoCAD System to Support Realtime Collaborative Design: Issues, Challenges and Achievements*. Tenth International Workshop on Collaborative Editing Systems in conjunction with ACM Conference on Computer Supported Cooperative Work, San Diego, California, USA, November 2008.

Appendix A – EXPRESS Schemas

PCI Schema

```
(*
 $Id: arm.exp,v 1.5 2004/10/22 14:44:52 darla Exp $
 ISO TC184/SC4/WG12 N1177 - ISO/TS 10303-1060 Product concept identification - EXPRESS ARM
*)

SCHEMA Product_concept_identification_arm;

ENTITY Market;
  name                      : STRING;
  market_segment_type       : OPTIONAL STRING;
END_ENTITY;


ENTITY Product_concept;
  id                        : STRING;
  name                      : STRING;
  description               : OPTIONAL STRING;
  target_market             : OPTIONAL Market;
UNIQUE
  UR1: id;
END_ENTITY;


END_SCHEMA;
```

PC Schema

```
(*
 $Id: arm.exp,v 1.11 2005/07/04 15:08:36 rocc Exp $
 ISO TC184/SC4/WG12 N2640 - ISO/TS 10303-1103 Product class - EXPRESS ARM
 Supersedes ISO TC184/SC4/WG12 N1586
*)

SCHEMA Product_class_arm;

USE FROM Product_concept_identification_arm;

TYPE expression_operator = ENUMERATION OF (or_operator, and_operator, oneof_operator,
not_operator);
END_TYPE;

 TYPE specification_operand_select = SELECT
      (Specification_expression, Specification);
 END_TYPE;


 ENTITY Class_category_association;
  associated_product_class : Product_class;
  mandatory : BOOLEAN;
  associated_category : Specification_category;
 END_ENTITY;

 ENTITY Class_condition_association;
  condition_type : STRING;
  associated_product_class : Product_class;
  description : OPTIONAL STRING;
  associated_condition : Specification_expression;
 END_ENTITY;

 ENTITY Class_inclusion_association;
  associated_product_class : Product_class;
  description : OPTIONAL STRING;
  associated_inclusion : Specification_inclusion;
 END_ENTITY;

 ENTITY Class_specification_association;
  associated_product_class : Product_class;
  association_type : STRING;
```

```
  associated_specification : Specification;
 END_ENTITY;

ENTITY Product_class
  SUBTYPE OF (Product_concept);
  version_id : OPTIONAL STRING;
  level_type : OPTIONAL STRING;
WHERE
WR1: NOT EXISTS(SELF\Product_concept.target_market);
END_ENTITY;

 ENTITY Product_class_relationship;
  description : OPTIONAL STRING;
  relating : Product_class;
  related : Product_class;
  relation_type : STRING;
 END_ENTITY;

 ENTITY Specification;
  id : STRING;
  version_id : OPTIONAL STRING;
  name : OPTIONAL STRING;
  description : OPTIONAL STRING;
  category : Specification_category;
  package : BOOLEAN;
 END_ENTITY;

 ENTITY Specification_category;
  id : STRING;
  description : STRING;
  implicit_exclusive_condition : BOOLEAN;
 END_ENTITY;

 ENTITY Specification_category_hierarchy;
  sub_category : Specification_category;
  super_category : Specification_category;
 END_ENTITY;

 ENTITY Specification_expression;
  id : OPTIONAL STRING;
  description : OPTIONAL STRING;
  operation : expression_operator;
  operand : SET[1:?] OF specification_operand_select;
 WHERE
 WR1: (operation <> not_operator) OR (SIZEOF(operand)=1);
 END_ENTITY;

 ENTITY Specification_inclusion;
  id : OPTIONAL STRING;
  description : OPTIONAL STRING;
  if_condition : specification_operand_select;
  included_specification : specification_operand_select;
 END_ENTITY;


END_SCHEMA;
```

Appendix B – OWL Classes (after translation)

PCI.owl


```
<?xml version="1.0" encoding="WINDOWS-1252"?>
<rdf:RDF
    xmlns="http://www.uninova.pt/ontology#"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xml:base="http://www.uninova.pt/ontology">

            <owl:Ontology rdf:about="Product_concept_identification_arm">
              <owl:versionInfo>1.0</owl:versionInfo>
              <rdfs:comment>Conversion of Product_concept_identification_arm EXPRESS SCHEMA</rdfs:comment>
            </owl:Ontology>

            <owl:Class rdf:about="#Market">

                <rdfs:subClassOf>
                  <owl:Restriction>
                    <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int">1</owl:cardinality>
                    <owl:onProperty>
                      <owl:DatatypeProperty rdf:ID="Market_name">
                                    <rdfs:domain rdf:resource="#Market"/>
                                    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
                            </owl:DatatypeProperty>
                    </owl:onProperty>
            </owl:Restriction>
                </rdfs:subClassOf>

                <rdfs:subClassOf>
                  <owl:Restriction>
                    <owl:minCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int">0</owl:minCardinality>
                    <owl:onProperty>
                      <owl:DatatypeProperty rdf:ID="Market_market_segment_type">
                                    <rdfs:domain rdf:resource="#Market"/>
                                    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
                            </owl:DatatypeProperty>
                    </owl:onProperty>
            </owl:Restriction>
                </rdfs:subClassOf>

            </owl:Class>

            <owl:Class rdf:about="#Product_concept">

                <rdfs:subClassOf>
                  <owl:Restriction>
                    <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int">1</owl:cardinality>
                    <owl:onProperty>
                      <owl:DatatypeProperty rdf:ID="Product_concept_id">
                                    <rdfs:domain rdf:resource="#Product_concept"/>
                                    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
                            </owl:DatatypeProperty>
                    </owl:onProperty>
            </owl:Restriction>
                </rdfs:subClassOf>

                <rdfs:subClassOf>
                  <owl:Restriction>
                    <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int">1</owl:cardinality>
                    <owl:onProperty>
                      <owl:DatatypeProperty rdf:ID="Product_concept_name">
                                    <rdfs:domain rdf:resource="#Product_concept"/>
                                    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
                            </owl:DatatypeProperty>
                    </owl:onProperty>
            </owl:Restriction>
                </rdfs:subClassOf>

                <rdfs:subClassOf>
                  <owl:Restriction>
                    <owl:minCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int">0</owl:minCardinality>
                    <owl:onProperty>
                      <owl:DatatypeProperty rdf:ID="Product_concept_description">
                                    <rdfs:domain rdf:resource="#Product_concept"/>
                                    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
                            </owl:DatatypeProperty>
                    </owl:onProperty>
            </owl:Restriction>
                </rdfs:subClassOf>

                <rdfs:subClassOf>
                  <owl:Restriction>
                    <owl:minCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int">0</owl:minCardinality>
                    <owl:onProperty>
                                    <owl:ObjectProperty rdf:ID="Product_concept_target_market">
                                      <rdfs:range rdf:resource="#Market"/>
                                    </owl:ObjectProperty>
                    </owl:onProperty>
            </owl:Restriction>
                </rdfs:subClassOf>

                <rdfs:subClassOf>
                  <owl:Restriction>
                    <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int">1</owl:cardinality>
                    <owl:onProperty>
```

```
                <owl:DatatypeProperty rdf:ID="Product_concept_UR1">
                                <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
                                <rdfs:domain rdf:resource="#Product_concept"/>
                                <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
                        </owl:DatatypeProperty>
                </owl:onProperty>
        </owl:Restriction>
            </rdfs:subClassOf>

            <rdfs:subClassOf>
              <owl:Restriction>
                <owl:minCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int">1</owl:minCardinality>
                <owl:onProperty>
                                <owl:ObjectProperty rdf:ID="Product_concept_market_operand_1">
                                  <rdfs:range rdf:resource="#Market_Set"/>
                                </owl:ObjectProperty>
                </owl:onProperty>
        </owl:Restriction>
            </rdfs:subClassOf>

            <rdfs:subClassOf>
              <owl:Restriction>
                <owl:minCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int">1</owl:minCardinality>
                <owl:onProperty>
                                <owl:ObjectProperty rdf:ID="Product_concept_market_operand_2">
                                  <rdfs:range rdf:resource="#Market_Bag"/>
                                </owl:ObjectProperty>
                </owl:onProperty>
        </owl:Restriction>
            </rdfs:subClassOf>

        </owl:Class>

        <Market rdf:ID="Market1">
                    <Market_name>Market Name</Market_name>
                    <Market_market_segment_type>Segment Type</Market_market_segment_type>
        </Market>

        <Market rdf:ID="Market2">
                    <Market_name>Market Name</Market_name>
                    <Market_market_segment_type>Segment Type</Market_market_segment_type>
        </Market>

        <Market rdf:ID="Market3">
                    <Market_name>Market Name</Market_name>
                    <Market_market_segment_type>Segment Type</Market_market_segment_type>
        </Market>

        <owl:Class rdf:ID="Market_Set">

                    <rdfs:subClassOf>
                       <owl:Restriction>
                         <owl:onProperty rdf:resource="#belongTo"/>
                         <owl:allValuesFrom rdf:resource="#Market"/>
                       </owl:Restriction>
                    </rdfs:subClassOf>

        </owl:Class>

        <owl:Class rdf:ID="Market_Bag">

                    <rdfs:subClassOf>
                       <owl:Restriction>
                         <owl:onProperty>
                                  <owl:ObjectProperty rdf:ID="Market_Bag_Element">
                                                                  <rdfs:range rdf:resource="#Market"/>
                                                        </owl:ObjectProperty>
                         </owl:onProperty>
                       </owl:Restriction>
                    </rdfs:subClassOf>

        </owl:Class>

  <owl:ObjectProperty rdf:ID="belongTo"/>


        <Market_Set rdf:ID="MSet">
                    <belongTo rdf:resource="#Market1"/>
                    <belongTo rdf:resource="#Market2"/>
                    <belongTo rdf:resource="#Market3"/>
        </Market_Set>

        <Market_Bag rdf:ID="MBag">
                    <Market_Bag_Element rdf:resource="#Market1"/>
                    <Market_Bag_Element rdf:resource="#Market2"/>
                    <Market_Bag_Element rdf:resource="#Market3"/>
        </Market_Bag>

        <Product_concept rdf:ID="PConcept1">
                    <Product_concept_id>Concept ID</Product_concept_id>
                    <Product_concept_name>Concept Name</Product_concept_name>
                    <Product_concept_description>Concept Description</Product_concept_description>
                    <Product_concept_target_market rdf:resource="#Market1"/>
                    <Product_concept_UR1>PC_UR1</Product_concept_UR1>
                    <Product_concept_market_operand_1 rdf:resource="#MSet"/>
                    <Product_concept_market_operand_2 rdf:resource="#MBag"/>
        </Product_concept>

</rdf:RDF>
```

PC.owl


```
<?xml version="1.0" encoding="WINDOWS-1252"?>
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">

            <owl:Ontology rdf:about="Product_class_arm">
              <owl:versionInfo>1.0</owl:versionInfo>
              <rdfs:comment>Conversion of Product_class_arm EXPRESS SCHEMA</rdfs:comment>
              <owl:imports rdf:resource="pci.owl"/>
            </owl:Ontology>

            <owl:Class rdf:about="#Label">
                        <owl:sameAs rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
            </owl:Class>

            <owl:Class rdf:ID="Specification_expression">
                        <owl:disjointWith rdf:resource="#Specification"/>
                        <owl:disjointWith rdf:resource="#Label"/>
            </owl:Class>

            <owl:Class rdf:ID="Specification">
                        <owl:disjointWith rdf:resource="#Specification_expression"/>
                        <owl:disjointWith rdf:resource="#Label"/>
            </owl:Class>

            <owl:Class rdf:ID="Label">
                        <owl:disjointWith rdf:resource="#Specification"/>
                        <owl:disjointWith rdf:resource="#Specification_expression"/>
            </owl:Class>

            <owl:Class rdf:ID="specification_operand_select">
                        <owl:unionOf rdf:parseType="Collection">
                                    <owl:Class rdf:about="#Specification_expression"/>
                                    <owl:Class rdf:about="#Specification"/>
                                    <owl:Class rdf:about="#Label"/>
                        </owl:unionOf>
            </owl:Class>

            <owl:Class rdf:about="#expression_operator">
                        <owl:oneOf rdf:parseType="Collection">
                                    <owl:Thing rdf:about="#or_operator"/>
                                    <owl:Thing rdf:about="#and_operator"/>
                                    <owl:Thing rdf:about="#oneof_operator"/>
                                    <owl:Thing rdf:about="#not_operator"/>
                        </owl:oneOf>
            </owl:Class>

            <owl:Class rdf:about="#Class_category_association">

                <rdfs:subClassOf>
                  <owl:Restriction>
                    <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int">1</owl:cardinality>
                    <owl:onProperty>
                      <owl:DatatypeProperty rdf:ID="Class_category_association_mandatory">
                                    <rdfs:domain rdf:resource="#Class_category_association"/>
                                    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#boolean"/>
                              </owl:DatatypeProperty>
                    </owl:onProperty>
            </owl:Restriction>
                </rdfs:subClassOf>

                <rdfs:subClassOf>
                  <owl:Restriction>
                    <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int">1</owl:cardinality>
                    <owl:onProperty>
                                    <owl:ObjectProperty rdf:ID="Class_category_association_associated_product_class">
                                      <rdfs:range rdf:resource="#Product_class"/>
                                    </owl:ObjectProperty>
                    </owl:onProperty>
            </owl:Restriction>
                </rdfs:subClassOf>

                <rdfs:subClassOf>
                  <owl:Restriction>
                    <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int">1</owl:cardinality>
                    <owl:onProperty>
                                    <owl:ObjectProperty rdf:ID="Class_category_association_associated_category">
                                      <rdfs:range rdf:resource="#Specification_category"/>
                                    </owl:ObjectProperty>
                    </owl:onProperty>
            </owl:Restriction>
                </rdfs:subClassOf>

            </owl:Class>

            <owl:Class rdf:about="#Class_condition_association">

                <rdfs:subClassOf>
                  <owl:Restriction>
                    <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int">1</owl:cardinality>
                    <owl:onProperty>
                      <owl:DatatypeProperty rdf:ID="Class_condition_association_condition_type">
                                    <rdfs:domain rdf:resource="#Class_condition_association"/>
                                    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
                              </owl:DatatypeProperty>
                    </owl:onProperty>
            </owl:Restriction>
                </rdfs:subClassOf>
```

```
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:minCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int">0</owl:minCardinality>
        <owl:onProperty>
          <owl:DatatypeProperty rdf:ID="Class_condition_association_description">
                          <rdfs:domain rdf:resource="#Class_condition_association"/>
                          <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
                 </owl:DatatypeProperty>
        </owl:onProperty>
</owl:Restriction>
    </rdfs:subClassOf>

    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int">1</owl:cardinality>
        <owl:onProperty>
                          <owl:ObjectProperty rdf:ID="Class_condition_association_associated_product_class">
                            <rdfs:range rdf:resource="#Product_class"/>
                          </owl:ObjectProperty>
        </owl:onProperty>
</owl:Restriction>
    </rdfs:subClassOf>

    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int">1</owl:cardinality>
        <owl:onProperty>
                          <owl:ObjectProperty rdf:ID="Class_condition_association_associated_condition">
                            <rdfs:range rdf:resource="#Specification_expression"/>
                          </owl:ObjectProperty>
        </owl:onProperty>
</owl:Restriction>
    </rdfs:subClassOf>

</owl:Class>

<owl:Class rdf:about="#Class_inclusion_association">

    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:minCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int">0</owl:minCardinality>
        <owl:onProperty>
          <owl:DatatypeProperty rdf:ID="Class_inclusion_association_description">
                          <rdfs:domain rdf:resource="#Class_inclusion_association"/>
                          <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
                 </owl:DatatypeProperty>
        </owl:onProperty>
</owl:Restriction>
    </rdfs:subClassOf>

    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int">1</owl:cardinality>
        <owl:onProperty>
                          <owl:ObjectProperty rdf:ID="Class_inclusion_association_associated_product_class">
                            <rdfs:range rdf:resource="#Product_class"/>
                          </owl:ObjectProperty>
        </owl:onProperty>
</owl:Restriction>
    </rdfs:subClassOf>

    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int">1</owl:cardinality>
        <owl:onProperty>
                          <owl:ObjectProperty rdf:ID="Class_inclusion_association_associated_inclusion">
                            <rdfs:range rdf:resource="#Specification_inclusion"/>
                          </owl:ObjectProperty>
        </owl:onProperty>
</owl:Restriction>
    </rdfs:subClassOf>

</owl:Class>

<owl:Class rdf:about="#Class_specification_association">

    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int">1</owl:cardinality>
        <owl:onProperty>
          <owl:DatatypeProperty rdf:ID="Class_specification_association_association_type">
                          <rdfs:domain rdf:resource="#Class_specification_association"/>
                          <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
                 </owl:DatatypeProperty>
        </owl:onProperty>
</owl:Restriction>
    </rdfs:subClassOf>

    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int">1</owl:cardinality>
        <owl:onProperty>
                          <owl:ObjectProperty rdf:ID="Class_specification_association_associated_product_class">
                            <rdfs:range rdf:resource="#Product_class"/>
                          </owl:ObjectProperty>
        </owl:onProperty>
</owl:Restriction>
    </rdfs:subClassOf>

    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int">1</owl:cardinality>
        <owl:onProperty>
```

```
                        <owl:ObjectProperty rdf:ID="Class_specification_association_associated_specification">
                          <rdfs:range rdf:resource="#Specification"/>
                        </owl:ObjectProperty>
        </owl:onProperty>
</owl:Restriction>
    </rdfs:subClassOf>

</owl:Class>

<owl:Class rdf:about="#Product_class">

    <rdfs:subClassOf rdf:resource="#Product_concept"/>
    <rdfs:subClassOf rdf:resource="#Specification"/>

    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:minCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int">0</owl:minCardinality>
        <owl:onProperty>
          <owl:DatatypeProperty rdf:ID="Product_class_version_id">
                        <rdfs:domain rdf:resource="#Product_class"/>
                        <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
                </owl:DatatypeProperty>
        </owl:onProperty>
</owl:Restriction>
    </rdfs:subClassOf>

    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:minCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int">0</owl:minCardinality>
        <owl:onProperty>
          <owl:DatatypeProperty rdf:ID="Product_class_level_type">
                        <rdfs:domain rdf:resource="#Product_class"/>
                        <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
                </owl:DatatypeProperty>
        </owl:onProperty>
</owl:Restriction>
    </rdfs:subClassOf>

</owl:Class>

<owl:Class rdf:about="#Product_class_relationship">

    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int">1</owl:cardinality>
        <owl:onProperty>
          <owl:DatatypeProperty rdf:ID="Product_class_relationship_relation_type">
                        <rdfs:domain rdf:resource="#Product_class_relationship"/>
                        <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
                </owl:DatatypeProperty>
        </owl:onProperty>
</owl:Restriction>
    </rdfs:subClassOf>

    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:minCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int">0</owl:minCardinality>
        <owl:onProperty>
          <owl:DatatypeProperty rdf:ID="Product_class_relationship_description">
                        <rdfs:domain rdf:resource="#Product_class_relationship"/>
                        <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
                </owl:DatatypeProperty>
        </owl:onProperty>
</owl:Restriction>
    </rdfs:subClassOf>

    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int">1</owl:cardinality>
        <owl:onProperty>
                        <owl:ObjectProperty rdf:ID="Product_class_relationship_relating">
                          <rdfs:range rdf:resource="#Product_Class"/>
                        </owl:ObjectProperty>
        </owl:onProperty>
</owl:Restriction>
    </rdfs:subClassOf>

    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int">1</owl:cardinality>
        <owl:onProperty>
                        <owl:ObjectProperty rdf:ID="Product_class_relationship_related">
                          <rdfs:range rdf:resource="#Product_Class"/>
                        </owl:ObjectProperty>
        </owl:onProperty>
</owl:Restriction>
    </rdfs:subClassOf>

</owl:Class>

<owl:Class rdf:about="#Specification">

    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int">1</owl:cardinality>
        <owl:onProperty>
          <owl:DatatypeProperty rdf:ID="Specification_id">
                        <rdfs:domain rdf:resource="#Specification"/>
                        <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
                </owl:DatatypeProperty>
        </owl:onProperty>
</owl:Restriction>
    </rdfs:subClassOf>
```

```
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int">1</owl:cardinality>
        <owl:onProperty>
          <owl:DatatypeProperty rdf:ID="Specification_package">
                        <rdfs:domain rdf:resource="#Specification"/>
                        <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#boolean"/>
                </owl:DatatypeProperty>
        </owl:onProperty>
</owl:Restriction>
    </rdfs:subClassOf>

    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:minCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int">0</owl:minCardinality>
        <owl:onProperty>
          <owl:DatatypeProperty rdf:ID="Specification_version_id">
                        <rdfs:domain rdf:resource="#Specification"/>
                        <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
                </owl:DatatypeProperty>
        </owl:onProperty>
</owl:Restriction>
    </rdfs:subClassOf>

    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:minCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int">0</owl:minCardinality>
        <owl:onProperty>
          <owl:DatatypeProperty rdf:ID="Specification_name">
                        <rdfs:domain rdf:resource="#Specification"/>
                        <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
                </owl:DatatypeProperty>
        </owl:onProperty>
</owl:Restriction>
    </rdfs:subClassOf>

    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:minCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int">0</owl:minCardinality>
        <owl:onProperty>
          <owl:DatatypeProperty rdf:ID="Specification_description">
                        <rdfs:domain rdf:resource="#Specification"/>
                        <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
                </owl:DatatypeProperty>
        </owl:onProperty>
</owl:Restriction>
    </rdfs:subClassOf>

    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int">1</owl:cardinality>
        <owl:onProperty>
                        <owl:ObjectProperty rdf:ID="Specification_category1">
                          <rdfs:range rdf:resource="#Specification_category"/>
                        </owl:ObjectProperty>
        </owl:onProperty>
</owl:Restriction>
    </rdfs:subClassOf>

</owl:Class>

<owl:Class rdf:about="#Specification_category">

    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int">1</owl:cardinality>
        <owl:onProperty>
          <owl:DatatypeProperty rdf:ID="Specification_category_id">
                        <rdfs:domain rdf:resource="#Specification_category"/>
                        <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
                </owl:DatatypeProperty>
        </owl:onProperty>
</owl:Restriction>
    </rdfs:subClassOf>

    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int">1</owl:cardinality>
        <owl:onProperty>
          <owl:DatatypeProperty rdf:ID="Specification_category_description">
                        <rdfs:domain rdf:resource="#Specification_category"/>
                        <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
                </owl:DatatypeProperty>
        </owl:onProperty>
</owl:Restriction>
    </rdfs:subClassOf>

    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int">1</owl:cardinality>
        <owl:onProperty>
          <owl:DatatypeProperty rdf:ID="Specification_category_implicity_exclusive_condition">
                        <rdfs:domain rdf:resource="#Specification_category"/>
                        <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#boolean"/>
                </owl:DatatypeProperty>
        </owl:onProperty>
</owl:Restriction>
    </rdfs:subClassOf>

</owl:Class>

<owl:Class rdf:about="#Specification_category_hierarchy">
```

```
            <rdfs:subClassOf>
              <owl:Restriction>
                <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int">1</owl:cardinality>
                <owl:onProperty>
                                <owl:ObjectProperty rdf:ID="Specification_category_hierarchy_sub_category">
                                  <rdfs:range rdf:resource="#Specification_category"/>
                                </owl:ObjectProperty>
                </owl:onProperty>
        </owl:Restriction>
            </rdfs:subClassOf>

            <rdfs:subClassOf>
              <owl:Restriction>
                <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int">1</owl:cardinality>
                <owl:onProperty>
                                <owl:ObjectProperty rdf:ID="Specification_category_hierarchy_super_category">
                                  <rdfs:range rdf:resource="#Specification_category"/>
                                </owl:ObjectProperty>
                </owl:onProperty>
        </owl:Restriction>
            </rdfs:subClassOf>

        </owl:Class>

        <owl:Class rdf:ID="specification_operand_select_Set">

                    <rdfs:subClassOf>
                       <owl:Restriction>
                         <owl:onProperty rdf:resource="#belongTo"/>
                         <owl:allValuesFrom rdf:resource="#specification_operand_select"/>
                       </owl:Restriction>
                    </rdfs:subClassOf>

        </owl:Class>

<owl:ObjectProperty rdf:ID="belongTo"/>

        <owl:Class rdf:about="#Specification_expression">

            <rdfs:subClassOf>
              <owl:Restriction>
                <owl:minCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int">0</owl:minCardinality>
                <owl:onProperty>
                  <owl:DatatypeProperty rdf:ID="Specification_expression_id">
                                <rdfs:domain rdf:resource="#Specification_expression"/>
                                <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
                         </owl:DatatypeProperty>
                </owl:onProperty>
        </owl:Restriction>
            </rdfs:subClassOf>

            <rdfs:subClassOf>
              <owl:Restriction>
                <owl:minCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int">0</owl:minCardinality>
                <owl:onProperty>
                  <owl:DatatypeProperty rdf:ID="Specification_expression_description">
                                <rdfs:domain rdf:resource="#Specification_expression"/>
                                <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
                         </owl:DatatypeProperty>
                </owl:onProperty>
        </owl:Restriction>
            </rdfs:subClassOf>

            <rdfs:subClassOf>
              <owl:Restriction>
                <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int">1</owl:cardinality>
                <owl:onProperty>
                                <owl:ObjectProperty rdf:ID="Specification_expression_operation">
                                  <rdfs:range rdf:resource="#expression_operator"/>
                                </owl:ObjectProperty>
                </owl:onProperty>
        </owl:Restriction>
            </rdfs:subClassOf>

            <rdfs:subClassOf>
              <owl:Restriction>
                <owl:minCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int">1</owl:minCardinality>
                <owl:onProperty>
                                <owl:ObjectProperty rdf:ID="Specification_expression_operand">
                                  <rdfs:range rdf:resource="#specification_operand_select_Set"/>
                                </owl:ObjectProperty>
                </owl:onProperty>
        </owl:Restriction>
            </rdfs:subClassOf>

        </owl:Class>

        <owl:Class rdf:about="#Specification_inclusion">

            <rdfs:subClassOf>
              <owl:Restriction>
                <owl:minCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int">0</owl:minCardinality>
                <owl:onProperty>
                  <owl:DatatypeProperty rdf:ID="Specification_inclusion_id">
                                <rdfs:domain rdf:resource="#Specification_inclusion"/>
                                <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
                         </owl:DatatypeProperty>
                </owl:onProperty>
        </owl:Restriction>
            </rdfs:subClassOf>

            <rdfs:subClassOf>
```

```
            <owl:Restriction>
              <owl:minCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int">0</owl:minCardinality>
              <owl:onProperty>
                <owl:DatatypeProperty rdf:ID="Specification_inclusion_description">
                                <rdfs:domain rdf:resource="#Specification_inclusion"/>
                                <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
                        </owl:DatatypeProperty>
              </owl:onProperty>
        </owl:Restriction>
            </rdfs:subClassOf>

            <rdfs:subClassOf>
              <owl:Restriction>
              <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int">1</owl:cardinality>
              <owl:onProperty>
                                <owl:ObjectProperty rdf:ID="Specification_inclusion_if_condition">
                                  <rdfs:range rdf:resource="#specification_operand_select"/>
                                </owl:ObjectProperty>
              </owl:onProperty>
        </owl:Restriction>
            </rdfs:subClassOf>

            <rdfs:subClassOf>
              <owl:Restriction>
              <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int">1</owl:cardinality>
              <owl:onProperty>
                                <owl:ObjectProperty rdf:ID="Specification_inclusion_included_specification">
                                  <rdfs:range rdf:resource="#specification_operand_select"/>
                                </owl:ObjectProperty>
              </owl:onProperty>
        </owl:Restriction>
            </rdfs:subClassOf>

        </owl:Class>

</rdf:RDF>
```

Appendix C – Connector 1 OWL Representation


```
<?xml version="1.0"?>
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:protege="http://protege.stanford.edu/plugins/owl/protege#"
    xmlns:p2="http://www.owl-ontologies.com/assert.owl#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns="http://www.owl-ontologies.com/ICE-Ontology_2.owl#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xml:base="http://www.owl-ontologies.com/ICE-Ontology_2.owl">
  <owl:Ontology rdf:about="">
    <owl:imports rdf:resource="http://www.owl-ontologies.com/ICE-Ontology_1.owl"/>
  </owl:Ontology>
  <owl:Class rdf:ID="Monitoring_Opening">
    <owl:disjointWith>
      <rdf:Description rdf:about="http://www.owl-ontologies.com/ICE-Ontology_1.owl#Mechanical_Indicator">
        <owl:disjointWith>
          <owl:Class rdf:ID="Screw_Fixation"/>
        </owl:disjointWith>
        <owl:disjointWith>
          <owl:Class rdf:ID="Screw"/>
        </owl:disjointWith>
        <owl:disjointWith>
          <owl:Class rdf:ID="Mechanical_Indicator"/>
        </owl:disjointWith>
        <owl:disjointWith>
          <rdf:Description rdf:about="http://www.owl-ontologies.com/ICE-Ontology_1.owl#Case">
            <owl:disjointWith>
              <owl:Class rdf:about="#Screw_Fixation"/>
            </owl:disjointWith>
            <owl:disjointWith>
              <owl:Class rdf:about="#Mechanical_Indicator"/>
            </owl:disjointWith>
            <owl:disjointWith>
              <rdf:Description rdf:about="http://www.owl-ontologies.com/ICE-Ontology_1.owl#Screw">
                <owl:disjointWith rdf:resource="http://www.owl-ontologies.com/ICE-Ontology_1.owl#Mechanical_Indicator"/>
                <owl:disjointWith>
                  <rdf:Description rdf:about="http://www.owl-ontologies.com/ICE-Ontology_1.owl#Screw_Fixation">
                    <owl:disjointWith rdf:resource="http://www.owl-ontologies.com/ICE-Ontology_1.owl#Case"/>
                    <owl:disjointWith rdf:resource="http://www.owl-ontologies.com/ICE-Ontology_1.owl#Screw"/>
                    <owl:disjointWith>
                      <owl:Class rdf:about="#Mechanical_Indicator"/>
                    </owl:disjointWith>
                    <owl:disjointWith>
                      <rdf:Description rdf:about="http://www.owl-ontologies.com/ICE-Ontology_1.owl#Spring">
                        <owl:disjointWith>
                          <owl:Class rdf:about="#Screw"/>
                        </owl:disjointWith>
                        <owl:disjointWith>
                          <owl:Class rdf:about="#Mechanical_Indicator"/>
                        </owl:disjointWith>
                        <owl:disjointWith>
                          <owl:Class rdf:ID="Spring"/>
                        </owl:disjointWith>
                        <owl:disjointWith rdf:resource="http://www.owl-ontologies.com/ICE-Ontology_1.owl#Screw_Fixation"/>
                        <owl:disjointWith>
                          <owl:Class rdf:about="#Screw_Fixation"/>
                        </owl:disjointWith>
                        <owl:disjointWith rdf:resource="http://www.owl-ontologies.com/ICE-Ontology_1.owl#Screw"/>
                        <owl:disjointWith rdf:resource="http://www.owl-ontologies.com/ICE-Ontology_1.owl#Case"/>
                        <owl:disjointWith>
                          <rdf:Description rdf:about="http://www.owl-ontologies.com/ICE-Ontology_1.owl#Monitoring_Opening">
                            <owl:disjointWith rdf:resource="http://www.owl-ontologies.com/ICE-Ontology_1.owl#Spring"/>
                            <owl:disjointWith>
                              <owl:Class rdf:about="#Spring"/>
                            </owl:disjointWith>
                            <owl:disjointWith>
                              <owl:Class rdf:about="#Mechanical_Indicator"/>
                            </owl:disjointWith>
                            <owl:disjointWith rdf:resource="#Monitoring_Opening"/>
                            <owl:disjointWith>
                              <owl:Class rdf:about="#Screw_Fixation"/>
                            </owl:disjointWith>
                            <owl:disjointWith                                rdf:resource="http://www.owl-ontologies.com/ICE-
Ontology_1.owl#Screw_Fixation"/>
                            <owl:disjointWith rdf:resource="http://www.owl-ontologies.com/ICE-Ontology_1.owl#Screw"/>
                            <owl:disjointWith>
                              <owl:Class rdf:about="#Screw"/>
                            </owl:disjointWith>
                            <owl:disjointWith rdf:resource="http://www.owl-ontologies.com/ICE-Ontology_1.owl#Case"/>
                            <owl:disjointWith                                rdf:resource="http://www.owl-ontologies.com/ICE-
Ontology_1.owl#Mechanical_Indicator"/>
                          </rdf:Description>
                        </owl:disjointWith>
                        <owl:disjointWith                                rdf:resource="http://www.owl-ontologies.com/ICE-
Ontology_1.owl#Mechanical_Indicator"/>
                        <owl:disjointWith rdf:resource="#Monitoring_Opening"/>
                      </rdf:Description>
                    </owl:disjointWith>
                    <owl:disjointWith rdf:resource="http://www.owl-ontologies.com/ICE-Ontology_1.owl#Monitoring_Opening"/>
                    <owl:disjointWith>
                      <owl:Class rdf:about="#Screw_Fixation"/>
                    </owl:disjointWith>
                    <owl:disjointWith rdf:resource="http://www.owl-ontologies.com/ICE-Ontology_1.owl#Mechanical_Indicator"/>
```

```
                <owl:disjointWith>
                  <owl:Class rdf:about="#Screw"/>
                </owl:disjointWith>
                <owl:disjointWith>
                  <owl:Class rdf:about="#Spring"/>
                </owl:disjointWith>
                <owl:disjointWith rdf:resource="#Monitoring_Opening"/>
              </rdf:Description>
            </owl:disjointWith>
            <owl:disjointWith>
              <owl:Class rdf:about="#Screw"/>
            </owl:disjointWith>
            <owl:disjointWith rdf:resource="http://www.owl-ontologies.com/ICE-Ontology_1.owl#Monitoring_Opening"/>
            <owl:disjointWith rdf:resource="http://www.owl-ontologies.com/ICE-Ontology_1.owl#Spring"/>
            <owl:disjointWith rdf:resource="http://www.owl-ontologies.com/ICE-Ontology_1.owl#Case"/>
            <owl:disjointWith>
              <owl:Class rdf:about="#Mechanical_Indicator"/>
            </owl:disjointWith>
            <owl:disjointWith>
              <owl:Class rdf:about="#Spring"/>
            </owl:disjointWith>
            <owl:disjointWith>
              <owl:Class rdf:about="#Screw_Fixation"/>
            </owl:disjointWith>
            <owl:disjointWith rdf:resource="#Monitoring_Opening"/>
          </rdf:Description>
        </owl:disjointWith>
        <owl:disjointWith>
          <owl:Class rdf:about="#Screw"/>
        </owl:disjointWith>
        <owl:disjointWith>
          <owl:Class rdf:about="#Spring"/>
        </owl:disjointWith>
        <owl:disjointWith rdf:resource="http://www.owl-ontologies.com/ICE-Ontology_1.owl#Screw_Fixation"/>
        <owl:disjointWith rdf:resource="http://www.owl-ontologies.com/ICE-Ontology_1.owl#Monitoring_Opening"/>
        <owl:disjointWith rdf:resource="http://www.owl-ontologies.com/ICE-Ontology_1.owl#Spring"/>
        <owl:disjointWith rdf:resource="http://www.owl-ontologies.com/ICE-Ontology_1.owl#Mechanical_Indicator"/>
        <owl:disjointWith rdf:resource="#Monitoring_Opening"/>
      </rdf:Description>
    </owl:disjointWith>
    <owl:disjointWith rdf:resource="http://www.owl-ontologies.com/ICE-Ontology_1.owl#Spring"/>
    <owl:disjointWith rdf:resource="#Monitoring_Opening"/>
    <owl:disjointWith>
      <owl:Class rdf:about="#Spring"/>
    </owl:disjointWith>
    <owl:disjointWith rdf:resource="http://www.owl-ontologies.com/ICE-Ontology_1.owl#Screw_Fixation"/>
    <owl:disjointWith rdf:resource="http://www.owl-ontologies.com/ICE-Ontology_1.owl#Monitoring_Opening"/>
    <owl:disjointWith rdf:resource="http://www.owl-ontologies.com/ICE-Ontology_1.owl#Screw"/>
  </rdf:Description>
</owl:disjointWith>
<owl:disjointWith>
  <owl:Class rdf:about="#Screw_Fixation"/>
</owl:disjointWith>
<owl:disjointWith rdf:resource="http://www.owl-ontologies.com/ICE-Ontology_1.owl#Case"/>
<owl:disjointWith rdf:resource="http://www.owl-ontologies.com/ICE-Ontology_1.owl#Screw"/>
<owl:disjointWith>
  <owl:Class rdf:about="#Mechanical_Indicator"/>
</owl:disjointWith>
<owl:disjointWith rdf:resource="http://www.owl-ontologies.com/ICE-Ontology_1.owl#Screw_Fixation"/>
<owl:disjointWith>
  <owl:Class rdf:about="#Screw"/>
</owl:disjointWith>
<owl:disjointWith rdf:resource="http://www.owl-ontologies.com/ICE-Ontology_1.owl#Monitoring_Opening"/>
<owl:disjointWith rdf:resource="http://www.owl-ontologies.com/ICE-Ontology_1.owl#Spring"/>
<owl:disjointWith>
  <owl:Class rdf:about="#Spring"/>
</owl:disjointWith>
</owl:Class>
<owl:Class rdf:about="#Screw">
<owl:disjointWith rdf:resource="#Monitoring_Opening"/>
<owl:disjointWith rdf:resource="http://www.owl-ontologies.com/ICE-Ontology_1.owl#Monitoring_Opening"/>
<owl:disjointWith rdf:resource="http://www.owl-ontologies.com/ICE-Ontology_1.owl#Mechanical_Indicator"/>
<owl:disjointWith>
  <owl:Class rdf:about="#Screw_Fixation"/>
</owl:disjointWith>
<owl:disjointWith rdf:resource="http://www.owl-ontologies.com/ICE-Ontology_1.owl#Screw"/>
<owl:disjointWith>
  <owl:Class rdf:about="#Spring"/>
</owl:disjointWith>
<owl:disjointWith rdf:resource="http://www.owl-ontologies.com/ICE-Ontology_1.owl#Spring"/>
<owl:disjointWith>
  <owl:Class rdf:about="#Mechanical_Indicator"/>
</owl:disjointWith>
<owl:disjointWith rdf:resource="http://www.owl-ontologies.com/ICE-Ontology_1.owl#Case"/>
<owl:disjointWith rdf:resource="http://www.owl-ontologies.com/ICE-Ontology_1.owl#Screw_Fixation"/>
</owl:Class>
<owl:Class rdf:about="#Mechanical_Indicator">
<owl:disjointWith rdf:resource="#Monitoring_Opening"/>
<owl:disjointWith rdf:resource="http://www.owl-ontologies.com/ICE-Ontology_1.owl#Mechanical_Indicator"/>
<owl:disjointWith rdf:resource="http://www.owl-ontologies.com/ICE-Ontology_1.owl#Case"/>
<owl:disjointWith rdf:resource="http://www.owl-ontologies.com/ICE-Ontology_1.owl#Screw"/>
<owl:disjointWith rdf:resource="http://www.owl-ontologies.com/ICE-Ontology_1.owl#Screw_Fixation"/>
<owl:disjointWith>
  <owl:Class rdf:about="#Spring"/>
</owl:disjointWith>
<owl:disjointWith rdf:resource="#Screw"/>
<owl:disjointWith>
  <owl:Class rdf:about="#Screw_Fixation"/>
</owl:disjointWith>
<owl:disjointWith rdf:resource="http://www.owl-ontologies.com/ICE-Ontology_1.owl#Monitoring_Opening"/>
<owl:disjointWith rdf:resource="http://www.owl-ontologies.com/ICE-Ontology_1.owl#Spring"/>
</owl:Class>
<owl:Class rdf:about="#Spring">
```

```
  <owl:disjointWith rdf:resource="http://www.owl-ontologies.com/ICE-Ontology_1.owl#Screw_Fixation"/>
  <owl:disjointWith>
    <owl:Class rdf:about="#Screw_Fixation"/>
  </owl:disjointWith>
  <owl:disjointWith rdf:resource="#Screw"/>
  <owl:disjointWith rdf:resource="http://www.owl-ontologies.com/ICE-Ontology_1.owl#Mechanical_Indicator"/>
  <owl:disjointWith rdf:resource="http://www.owl-ontologies.com/ICE-Ontology_1.owl#Monitoring_Opening"/>
  <owl:disjointWith rdf:resource="#Mechanical_Indicator"/>
  <owl:disjointWith rdf:resource="http://www.owl-ontologies.com/ICE-Ontology_1.owl#Spring"/>
  <owl:disjointWith rdf:resource="http://www.owl-ontologies.com/ICE-Ontology_1.owl#Screw"/>
  <owl:disjointWith rdf:resource="#Monitoring_Opening"/>
  <owl:disjointWith rdf:resource="http://www.owl-ontologies.com/ICE-Ontology_1.owl#Case"/>
</owl:Class>
<owl:Class rdf:ID="Conductor"/>
<owl:Class rdf:about="#Screw_Fixation">
  <owl:disjointWith rdf:resource="#Screw"/>
  <owl:disjointWith rdf:resource="#Spring"/>
  <owl:disjointWith rdf:resource="http://www.owl-ontologies.com/ICE-Ontology_1.owl#Mechanical_Indicator"/>
  <owl:disjointWith rdf:resource="#Mechanical_Indicator"/>
  <owl:disjointWith rdf:resource="http://www.owl-ontologies.com/ICE-Ontology_1.owl#Spring"/>
  <owl:disjointWith rdf:resource="http://www.owl-ontologies.com/ICE-Ontology_1.owl#Case"/>
  <owl:disjointWith rdf:resource="http://www.owl-ontologies.com/ICE-Ontology_1.owl#Screw_Fixation"/>
  <owl:disjointWith rdf:resource="#Monitoring_Opening"/>
  <owl:disjointWith rdf:resource="http://www.owl-ontologies.com/ICE-Ontology_1.owl#Monitoring_Opening"/>
  <owl:disjointWith rdf:resource="http://www.owl-ontologies.com/ICE-Ontology_1.owl#Screw"/>
</owl:Class>
<owl:Class rdf:ID="Case">
  <owl:equivalentClass>
    <owl:Class>
      <owl:intersectionOf rdf:parseType="Collection">
        <owl:Restriction>
          <owl:onProperty>
            <owl:ObjectProperty rdf:ID="composedOf_screw_fixation"/>
          </owl:onProperty>
          <owl:allValuesFrom rdf:resource="#Screw_Fixation"/>
        </owl:Restriction>
        <owl:Restriction>
          <owl:onProperty>
            <owl:ObjectProperty rdf:about="#composedOf_screw_fixation"/>
          </owl:onProperty>
          <owl:someValuesFrom rdf:resource="#Screw_Fixation"/>
        </owl:Restriction>
        <owl:Class>
          <owl:intersectionOf rdf:parseType="Collection">
            <owl:Restriction>
              <owl:onProperty>
                <owl:ObjectProperty rdf:ID="composedOf_spring"/>
              </owl:onProperty>
              <owl:allValuesFrom rdf:resource="#Spring"/>
            </owl:Restriction>
            <owl:Restriction>
              <owl:onProperty>
                <owl:ObjectProperty rdf:about="#composedOf_spring"/>
              </owl:onProperty>
              <owl:someValuesFrom rdf:resource="#Spring"/>
            </owl:Restriction>
          </owl:intersectionOf>
        </owl:Class>
      </owl:intersectionOf>
    </owl:Class>
  </owl:equivalentClass>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty>
        <owl:ObjectProperty rdf:about="#composedOf_spring"/>
      </owl:onProperty>
      <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
      >1</owl:cardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:minCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
      >1</owl:minCardinality>
      <owl:onProperty>
        <owl:ObjectProperty rdf:about="#composedOf_screw_fixation"/>
      </owl:onProperty>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
</owl:Class>
<owl:Class rdf:ID="Shell">
  <owl:equivalentClass>
    <owl:Class>
      <owl:intersectionOf rdf:parseType="Collection">
        <owl:Restriction>
          <owl:onProperty>
            <owl:ObjectProperty rdf:ID="composedOf_case"/>
          </owl:onProperty>
          <owl:allValuesFrom rdf:resource="#Case"/>
        </owl:Restriction>
        <owl:Restriction>
          <owl:onProperty>
            <owl:ObjectProperty rdf:about="#composedOf_case"/>
          </owl:onProperty>
          <owl:someValuesFrom rdf:resource="#Case"/>
        </owl:Restriction>
        <owl:Class>
          <owl:intersectionOf rdf:parseType="Collection">
            <owl:Restriction>
              <owl:onProperty>
                <owl:ObjectProperty rdf:ID="composedOf_mechanical_indicator"/>
              </owl:onProperty>
```

```
            <owl:allValuesFrom rdf:resource="#Mechanical_Indicator"/>
          </owl:Restriction>
          <owl:Restriction>
            <owl:onProperty>
              <owl:ObjectProperty rdf:about="#composedOf_mechanical_indicator"/>
            </owl:onProperty>
            <owl:someValuesFrom rdf:resource="#Mechanical_Indicator"/>
          </owl:Restriction>
        </owl:intersectionOf>
      </owl:Class>
      <owl:Class>
        <owl:intersectionOf rdf:parseType="Collection">
          <owl:Restriction>
            <owl:onProperty>
              <owl:ObjectProperty rdf:ID="composedOf_monitoring_opening"/>
            </owl:onProperty>
            <owl:allValuesFrom rdf:resource="#Monitoring_Opening"/>
          </owl:Restriction>
          <owl:Restriction>
            <owl:onProperty>
              <owl:ObjectProperty rdf:about="#composedOf_monitoring_opening"/>
            </owl:onProperty>
            <owl:someValuesFrom rdf:resource="#Monitoring_Opening"/>
          </owl:Restriction>
        </owl:intersectionOf>
      </owl:Class>
    </owl:intersectionOf>
  </owl:Class>
</owl:equivalentClass>
<rdfs:subClassOf>
  <owl:Restriction>
    <owl:onProperty>
      <owl:ObjectProperty rdf:about="#composedOf_mechanical_indicator"/>
    </owl:onProperty>
    <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
    >2</owl:cardinality>
  </owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf>
  <owl:Restriction>
    <owl:onProperty>
      <owl:ObjectProperty rdf:about="#composedOf_monitoring_opening"/>
    </owl:onProperty>
    <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
    >2</owl:cardinality>
  </owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf>
  <owl:Restriction>
    <owl:onProperty>
      <owl:ObjectProperty rdf:about="#composedOf_case"/>
    </owl:onProperty>
    <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
    >1</owl:cardinality>
  </owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
</owl:Class>
<owl:Class rdf:ID="Connector">
  <owl:equivalentClass>
    <owl:Class>
      <owl:intersectionOf rdf:parseType="Collection">
        <owl:Restriction>
          <owl:onProperty>
            <owl:ObjectProperty rdf:ID="composedOf_shell"/>
          </owl:onProperty>
          <owl:allValuesFrom rdf:resource="#Shell"/>
        </owl:Restriction>
        <owl:Restriction>
          <owl:onProperty>
            <owl:ObjectProperty rdf:about="#composedOf_shell"/>
          </owl:onProperty>
          <owl:someValuesFrom rdf:resource="#Shell"/>
        </owl:Restriction>
        <owl:Class>
          <owl:intersectionOf rdf:parseType="Collection">
            <owl:Restriction>
              <owl:onProperty>
                <owl:ObjectProperty rdf:ID="composedOf_conductor"/>
              </owl:onProperty>
              <owl:allValuesFrom rdf:resource="#Conductor"/>
            </owl:Restriction>
            <owl:Restriction>
              <owl:onProperty>
                <owl:ObjectProperty rdf:about="#composedOf_conductor"/>
              </owl:onProperty>
              <owl:someValuesFrom rdf:resource="#Conductor"/>
            </owl:Restriction>
          </owl:intersectionOf>
        </owl:Class>
        <owl:Class>
          <owl:intersectionOf rdf:parseType="Collection">
            <owl:Restriction>
              <owl:allValuesFrom rdf:resource="#Screw"/>
              <owl:onProperty>
                <owl:ObjectProperty rdf:ID="composedOf_screw"/>
              </owl:onProperty>
            </owl:Restriction>
            <owl:Restriction>
              <owl:onProperty>
                <owl:ObjectProperty rdf:about="#composedOf_screw"/>
              </owl:onProperty>
              <owl:someValuesFrom rdf:resource="#Screw"/>
```

```
            </owl:Restriction>
          </owl:intersectionOf>
        </owl:Class>
      </owl:intersectionOf>
    </owl:Class>
  </owl:equivalentClass>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty>
        <owl:ObjectProperty rdf:about="#composedOf_conductor"/>
      </owl:onProperty>
      <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
      >2</owl:cardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty>
        <owl:ObjectProperty rdf:about="#composedOf_screw"/>
      </owl:onProperty>
      <owl:minCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
      >1</owl:minCardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
      >1</owl:cardinality>
      <owl:onProperty>
        <owl:ObjectProperty rdf:about="#composedOf_shell"/>
      </owl:onProperty>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
</owl:Class>
<owl:ObjectProperty rdf:about="#composedOf_shell">
  <rdfs:subPropertyOf>
    <owl:ObjectProperty rdf:ID="composedOf"/>
  </rdfs:subPropertyOf>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#composedOf">
  <owl:inverseOf>
    <owl:ObjectProperty rdf:ID="partOf"/>
  </owl:inverseOf>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#composedOf_spring">
  <rdfs:subPropertyOf rdf:resource="#composedOf"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="inverse_of_composedOf_17">
  <rdfs:subPropertyOf>
    <owl:ObjectProperty rdf:about="#partOf"/>
  </rdfs:subPropertyOf>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="inverse_of_composedOf_18">
  <rdfs:subPropertyOf>
    <owl:ObjectProperty rdf:about="#partOf"/>
  </rdfs:subPropertyOf>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#composedOf_screw">
  <rdfs:subPropertyOf rdf:resource="#composedOf"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="inverse_of_composedOf_16">
  <rdfs:subPropertyOf>
    <owl:ObjectProperty rdf:about="#partOf"/>
  </rdfs:subPropertyOf>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#composedOf_mechanical_indicator">
  <rdfs:subPropertyOf rdf:resource="#composedOf"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="inverse_of_composedOf_15">
  <rdfs:subPropertyOf>
    <owl:ObjectProperty rdf:about="#partOf"/>
  </rdfs:subPropertyOf>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#composedOf_case">
  <rdfs:subPropertyOf rdf:resource="#composedOf"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="inverse_of_composedOf_14">
  <rdfs:subPropertyOf>
    <owl:ObjectProperty rdf:about="#partOf"/>
  </rdfs:subPropertyOf>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="inverse_of_composedOf_13">
  <rdfs:subPropertyOf>
    <owl:ObjectProperty rdf:about="#partOf"/>
  </rdfs:subPropertyOf>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="inverse_of_composedOf_12">
  <rdfs:subPropertyOf>
    <owl:ObjectProperty rdf:about="#partOf"/>
  </rdfs:subPropertyOf>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#composedOf_conductor">
  <rdfs:subPropertyOf rdf:resource="#composedOf"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#partOf">
  <owl:inverseOf rdf:resource="#composedOf"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#composedOf_screw_fixation">
  <rdfs:subPropertyOf rdf:resource="#composedOf"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="inverse_of_composedOf_19">
  <rdfs:subPropertyOf rdf:resource="#partOf"/>
```

```
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:about="#composedOf_monitoring_opening">
    <rdfs:subPropertyOf rdf:resource="#composedOf"/>
  </owl:ObjectProperty>
</rdf:RDF>

<!-- Created with Protege (with OWL Plugin 3.3.1, Build 430)  http://protege.stanford.edu -->
```

Appendix D – Connector 2 OWL Representation


```
<?xml version="1.0"?>
<rdf:RDF
    xmlns:p1="http://www.owl-ontologies.com/ICE-Ontology_1.owl#"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xml:base="http://www.owl-ontologies.com/ICE-Ontology_1.owl">
  <owl:Ontology rdf:about=""/>
  <owl:Class rdf:ID="Screw_Fixation"/>
  <owl:Class rdf:ID="Body">
    <owl:equivalentClass>
      <owl:Class>
        <owl:intersectionOf rdf:parseType="Collection">
          <owl:Restriction>
            <owl:allValuesFrom>
              <owl:Class rdf:ID="Case"/>
            </owl:allValuesFrom>
            <owl:onProperty>
              <owl:ObjectProperty rdf:ID="composedOf_case"/>
            </owl:onProperty>
          </owl:Restriction>
          <owl:Restriction>
            <owl:someValuesFrom rdf:resource="#Case"/>
            <owl:onProperty>
              <owl:ObjectProperty rdf:about="#composedOf_case"/>
            </owl:onProperty>
          </owl:Restriction>
          <owl:Class>
            <owl:intersectionOf rdf:parseType="Collection">
              <owl:Restriction>
                <owl:onProperty>
                  <owl:ObjectProperty rdf:ID="composedOf_mechanical_indicators"/>
                </owl:onProperty>
                <owl:allValuesFrom>
                  <owl:Class rdf:ID="Mechanical_Indicator"/>
                </owl:allValuesFrom>
              </owl:Restriction>
              <owl:Restriction>
                <owl:onProperty>
                  <owl:ObjectProperty rdf:about="#composedOf_mechanical_indicators"/>
                </owl:onProperty>
                <owl:someValuesFrom rdf:resource="#Mechanical_Indicator"/>
              </owl:Restriction>
            </owl:intersectionOf>
          </owl:Class>
          <owl:Class>
            <owl:intersectionOf rdf:parseType="Collection">
              <owl:Restriction>
                <owl:allValuesFrom rdf:resource="#Screw_Fixation"/>
                <owl:onProperty>
                  <owl:ObjectProperty rdf:ID="composedOf_screw_fixation"/>
                </owl:onProperty>
              </owl:Restriction>
              <owl:Restriction>
                <owl:onProperty>
                  <owl:ObjectProperty rdf:about="#composedOf_screw_fixation"/>
                </owl:onProperty>
                <owl:someValuesFrom rdf:resource="#Screw_Fixation"/>
              </owl:Restriction>
            </owl:intersectionOf>
          </owl:Class>
          <owl:Class>
            <owl:intersectionOf rdf:parseType="Collection">
              <owl:Restriction>
                <owl:allValuesFrom>
                  <owl:Class rdf:ID="Spring"/>
                </owl:allValuesFrom>
                <owl:onProperty>
                  <owl:ObjectProperty rdf:ID="composedOf_spring"/>
                </owl:onProperty>
              </owl:Restriction>
              <owl:Restriction>
                <owl:someValuesFrom rdf:resource="#Spring"/>
                <owl:onProperty>
                  <owl:ObjectProperty rdf:about="#composedOf_spring"/>
                </owl:onProperty>
              </owl:Restriction>
            </owl:intersectionOf>
          </owl:Class>
          <owl:Class>
            <owl:intersectionOf rdf:parseType="Collection">
              <owl:Restriction>
                <owl:allValuesFrom>
                  <owl:Class rdf:ID="Monitoring_Opening"/>
                </owl:allValuesFrom>
                <owl:onProperty>
                  <owl:ObjectProperty rdf:ID="composedOf_monitoring_opening"/>
                </owl:onProperty>
              </owl:Restriction>
              <owl:Restriction>
                <owl:someValuesFrom rdf:resource="#Monitoring_Opening"/>
                <owl:onProperty>
                  <owl:ObjectProperty rdf:about="#composedOf_monitoring_opening"/>
```

```
            </owl:onProperty>
          </owl:Restriction>
        </owl:intersectionOf>
      </owl:Class>
    </owl:intersectionOf>
  </owl:Class>
</owl:equivalentClass>
<rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
<rdfs:subClassOf>
  <owl:Restriction>
    <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
    >1</owl:cardinality>
    <owl:onProperty>
      <owl:ObjectProperty rdf:about="#composedOf_case"/>
    </owl:onProperty>
  </owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf>
  <owl:Restriction>
    <owl:onProperty>
      <owl:ObjectProperty rdf:about="#composedOf_screw_fixation"/>
    </owl:onProperty>
    <owl:minCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
    >1</owl:minCardinality>
  </owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf>
  <owl:Restriction>
    <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
    >1</owl:cardinality>
    <owl:onProperty>
      <owl:ObjectProperty rdf:about="#composedOf_spring"/>
    </owl:onProperty>
  </owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf>
  <owl:Restriction>
    <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
    >2</owl:cardinality>
    <owl:onProperty>
      <owl:ObjectProperty rdf:about="#composedOf_monitoring_opening"/>
    </owl:onProperty>
  </owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf>
  <owl:Restriction>
    <owl:onProperty>
      <owl:ObjectProperty rdf:about="#composedOf_mechanical_indicators"/>
    </owl:onProperty>
    <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
    >2</owl:cardinality>
  </owl:Restriction>
</rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="Connector">
  <owl:equivalentClass>
    <owl:Class>
      <owl:intersectionOf rdf:parseType="Collection">
        <owl:Restriction>
          <owl:allValuesFrom rdf:resource="#Body"/>
          <owl:onProperty>
            <owl:ObjectProperty rdf:ID="composedOf_body"/>
          </owl:onProperty>
        </owl:Restriction>
        <owl:Restriction>
          <owl:onProperty>
            <owl:ObjectProperty rdf:about="#composedOf_body"/>
          </owl:onProperty>
          <owl:someValuesFrom rdf:resource="#Body"/>
        </owl:Restriction>
        <owl:Class>
          <owl:intersectionOf rdf:parseType="Collection">
            <owl:Restriction>
              <owl:onProperty>
                <owl:ObjectProperty rdf:ID="composedOf_screw"/>
              </owl:onProperty>
              <owl:allValuesFrom>
                <owl:Class rdf:ID="Screw"/>
              </owl:allValuesFrom>
            </owl:Restriction>
            <owl:Restriction>
              <owl:someValuesFrom rdf:resource="#Screw"/>
              <owl:onProperty>
                <owl:ObjectProperty rdf:about="#composedOf_screw"/>
              </owl:onProperty>
            </owl:Restriction>
          </owl:intersectionOf>
        </owl:Class>
        <owl:Class>
          <owl:intersectionOf rdf:parseType="Collection">
            <owl:Restriction>
              <owl:onProperty>
                <owl:ObjectProperty rdf:ID="composedOf_cable"/>
              </owl:onProperty>
              <owl:allValuesFrom>
                <owl:Class rdf:ID="Cable"/>
              </owl:allValuesFrom>
            </owl:Restriction>
            <owl:Restriction>
              <owl:someValuesFrom rdf:resource="#Cable"/>
              <owl:onProperty>
                <owl:ObjectProperty rdf:about="#composedOf_cable"/>
              </owl:onProperty>
```

```
            </owl:Restriction>
          </owl:intersectionOf>
        </owl:Class>
      </owl:intersectionOf>
    </owl:Class>
  </owl:equivalentClass>
  <rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty>
        <owl:ObjectProperty rdf:about="#composedOf_cable"/>
      </owl:onProperty>
      <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
      >2</owl:cardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:minCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
      >1</owl:minCardinality>
      <owl:onProperty>
        <owl:ObjectProperty rdf:about="#composedOf_screw"/>
      </owl:onProperty>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
      >1</owl:cardinality>
      <owl:onProperty>
        <owl:ObjectProperty rdf:about="#composedOf_body"/>
      </owl:onProperty>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
<owl:ObjectProperty rdf:ID="partOf">
  <owl:inverseOf>
    <owl:ObjectProperty rdf:ID="composedOf"/>
  </owl:inverseOf>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#composedOf_screw_fixation">
  <rdfs:subPropertyOf>
    <owl:ObjectProperty rdf:about="#composedOf"/>
  </rdfs:subPropertyOf>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#composedOf_screw">
  <rdfs:subPropertyOf>
    <owl:ObjectProperty rdf:about="#composedOf"/>
  </rdfs:subPropertyOf>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="inverse_of_composedOf_17">
  <rdfs:subPropertyOf rdf:resource="#partOf"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#composedOf_monitoring_opening">
  <rdfs:subPropertyOf>
    <owl:ObjectProperty rdf:about="#composedOf"/>
  </rdfs:subPropertyOf>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#composedOf_case">
  <rdfs:subPropertyOf>
    <owl:ObjectProperty rdf:about="#composedOf"/>
  </rdfs:subPropertyOf>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#composedOf">
  <owl:inverseOf rdf:resource="#partOf"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#composedOf_body">
  <rdfs:subPropertyOf rdf:resource="#composedOf"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#composedOf_spring">
  <rdfs:subPropertyOf rdf:resource="#composedOf"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#composedOf_mechanical_indicators">
  <rdfs:subPropertyOf rdf:resource="#composedOf"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="inverse_of_composedOf_18">
  <rdfs:subPropertyOf rdf:resource="#partOf"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#composedOf_cable">
  <rdfs:subPropertyOf rdf:resource="#composedOf"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="inverse_of_composedOf_20">
  <rdfs:subPropertyOf rdf:resource="#partOf"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="inverse_of_composedOf_19">
  <rdfs:subPropertyOf rdf:resource="#partOf"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="inverse_of_composedOf_16">
  <rdfs:subPropertyOf rdf:resource="#partOf"/>
</owl:ObjectProperty>
</rdf:RDF>

<!-- Created with Protege (with OWL Plugin 3.3.1, Build 430)  http://protege.stanford.edu -->
```