Methodology for knowledge-based engineering template update : focus on decision support and instances update

Olivier Kuhn

N° d'ordre Année 2010

Thèse de l'Université de Lyon

Délivrée par
L'Université Claude Bernard Lyon 1
École Doctorale Informatique et Mathématiques Lyon

pour l'obtention du
Diplôme de Doctorat
Spécialité informatique (CNU 27)
(arrêté du 7 août 2006)

et soutenue publiquement le 22 octobre 2010
par
M. Kuhn Olivier

Methodology for Knowledge-Based Engineering Template Update

Focus on Decision Support and Instances Update

Directeurs de thèse: Pr Parisa Ghodous
Pr Pierre Collet

Composition du jury:

Rapporteur externe:	Pr Yamine Ait-Ameur	Professeur de l'Université de Poitier
Rapporteur externe:	Pr Nhan Le Thanh	Professeur de l'Université de Nice Sophia Antipolis
Examinateur:	Pr Yvon Gardan	Professeur de l'Université de Reims
Examinateur:	Pr Georg Rock	Professeur de l'Université de Trier, Germany
Directeur de thèse:	Pr Parisa Ghodous	Professeur de l'Université Lyon 1
Co-directeur de thèse:	Pr Pierre Collet	Professeur de l'Université de Strasbourg
Membre invité:	Dr Josip Stjepandic	Responsable industriel, PROSTEP AG
Membre invité:	Dr Thomas Dusch	Responsable industriel, PROSTEP AG

To my parents, Francine and Bernard Kuhn...

Summary

The present Ph.D. thesis addresses the problem of knowledge-based engineering template update in product design. The reuse of design knowledge has become a key asset for the company's competitiveness. Knowledge-based engineering templates allow to store best practices and know-how *via* formulas, rules, scripts, etc. This design knowledge can then be reused by instantiating the template. The instantiation results in the creation of an instance of the template in the specified context.

In the scope of complex and large products, such as cars or aircrafts, the maintenance of knowledge-based engineering templates is a challenging task. Several engineers from various disciplines work together and make evolve the templates in order to extend their capabilities or to fix bugs. Furthermore, in some cases, the modifications applied to templates should be forwarded to their instances in order that they benefit from the changes. These issues slow down the adoption of template technologies at a large scale within companies. The objective of this work is to propose an approach in order to support engineers in the template update related tasks.

In order to address these issues, a process supporting the template update related tasks is defined. Then a framework is proposed that helps design engineers during the template update process by providing a decision support system and a strategy for the update of template instances. The former is a system designed to ease the collaboration between various experts in order to solve template related problems. The latter aims at providing a sequence of updates to follow, in order to forward the templates' modifications to their instances. This sequence is computed with data extracted from models and templates, which are stored in an ontology designed for this purpose. The ontology is used to represent and to infer knowledge about templates, products and their relations. This facilitates the construction of update sequences as it provides an efficient overview of relationships, even implicit ones.

Keywords: Knowledge-Based Engineering, Knowledge Template, Update Strategy, Ontology, Scheduling, Issue-Based Information System, Computer Supported Collaborative Work.

Résumé

Les travaux de recherche présentés adressent des problèmes de mises à jour de knowledge-based engineering templates dans le cadre de la conception de produits. La réutilisation de connaissances de conception est devenue un avantage clé pour la compétitivité des entreprises. Le savoir faire ainsi que les bonnes pratiques peuvent être stockés au sein de templates par le biais de formules, règles, scripts, etc. Ces connaissances de conception peuvent alors être réutilisées en instanciant le template. L'instanciation résulte en la création d'une instance du template dans le contexte spécifié.

Dans le cadre de produit complexes et imposants tels que des voitures ou des avions, la maintenance des templates est une tache ambitieuse. Plusieurs ingénieurs de diverses disciplines travaillent ensemble et font évoluer les templates pour augmenter leurs aptitudes ou pour corriger des problèmes. De plus, dans certains cas, les modifications faites aux templates devraient être appliquées à leurs instances à fin qu'elles puissent bénéficier de ces modifications. Ces problèmes ralentissent l'adoption à grande envergure des templates au sein des entreprises. L'objectif de ce travail est de proposer une approche à fin d'épauler les ingénieurs dans les tâches relatives à la mise à jour des templates.

Pour traiter ces problèmes, un processus adressant les tâches relatives à la mise à jour des templates est défini. Ensuite, un framework est proposé dans le but d'aider les ingénieurs de conception au cours du processus de mise à jour, en fournissant un système d'aide à la décision ainsi qu'une stratégie de mise à jour des instances. Le premier est un système conçu pour faciliter la collaboration entre les différent experts dans le but de résoudre les problèmes liés aux templates. Le second a pour but d'élaborer une séquence de mise à jour à fin d'appliquer les modifications du template à ses instances. La séquence est calculée avec les données extraites à partir des modèles CAD et des templates. Ces données sont stockées dans une ontologie conçue spécialement à cet effet. L'ontologie est utilisée pour représenter et inférer des connaissances sur les templates, les produits et leur relations. Cela facilite la construction des séquences de mises à jour en fournissant une vue d'ensemble sur les relations entre documents, même implicites.

Mots clefs : Ingénierie à base de connaissances, Knowledge-based engineering template, Stratégie de mise à jour, Ontologie, Ordonnancement, Système d'information pour l'aide aux processus de décision, Travail collaboratif.

Discipline : Informatique

Laboratoire LIRIS – Laboratoire d'InfoRmatique en Image et Systèmes d'information – UMR 5205 – Université Claude Bernard Lyon 1
Bâtiment Nautibus, Campus de la Doua - 8, Bd Niels Bohr, 69622 Villeurbanne Cedex FRANCE

Laboratoire LSIIT – Laboratoire des Sciences de l'Image, de l'Informatique et de la Télédétection – UMR 7005 – Université de Strasbourg
Pôle API - Bd Sébastien Brant, BP 10413, 67412 Illkirch Cedex FRANCE

Acknowledgements

First of all, I owe my deepest gratitude to my academic advisors and tutors, Prof. Parisa Ghodous and Prof. Pierre Collet, for their support, encouragements and advices. I appreciated to work with them as well as the autonomy and freedom they gave me during these three years.

This thesis would not have been possible without the precious help and support from Dr Thomas Dusch, who has been present all along this work and who has given me a lot of his time. The discussion around my work with him have been very helpful. I want also to thank him for his meticulous review of this manuscript.

I owe my deepest gratitude to PROSTEP AG and ANRT, which have funded and supported this research during the three years this work lasted. I want also to show my gratitude to the LIRIS and LSIIT laboratories and respectively the SOC and FDBT teams, which have welcomed me in their facilities and provided the necessary equipments in order to allow me to work within the best conditions.

I would like to thank Dr Josip Stjepandic, Dr Harald Liese and Lionel Antegnard from PROSTEP AG, who initiated and supported this work and who guided me in the company.

I am deeply grateful to Prof. Yamine Ait Ameur and Prof. Nhan Le Thanh who accepted the hard task of reviewing this manuscript, as well as to Prof. Yvon Gardan and Prof. Georg Rock for having accepted to be members of the jury.

This manuscript is the result of three years of work but also of human relationships. So I say "thank you" to Romain and Ogier with whom I successively shared the office and with whom I exchanged a lot during one year and a half.
Besides these two "officemates," I want to thank Thomas, Camille, Johnathan, Aurelie, Fred, Arijit, Deepak, Emmanuel, Nicolas, Melia, Thorsten, Gunnar, Andreas, Moisés, Tri and those that I forgot here for their good mood and the pleasant working atmosphere around them.
I am also grateful to the Doudou family, and to Fabien and Chlem for hosting me during my stays in Lyon.

Last but not least, I thank my parents Francine and Bernard, for supporting me even if they do not know exactly what I do and for giving me the chance to study. As they do not understand English, I will also write "un grand merci à vous, maman et papa, pour tout ce qui vous avez fait pour moi depuis près de 26 années."

Table of Contents

I Introduction

II State of the art

VI Final conclusion

Appendices

List of Figures

List of Tables

List of Algorithms

Acronyms

AGPL	GNU Affero General Public License
AM	Applications Modules
API	Application Programming Interface
CAD	Computer Aided Design
CAE	Computer Aided Engineering
CAM	Computer Aided Manufacturing
CE	Concurrent/Collaborative Engineering
CIFRE	Conventions Industrielles de Formation par la REcherche
CSCW	Computer Supported Collaborative Work
DAML	DARPA Agent Markup Language
DAML-ONT	DAML Ontology
DARPA	Defense Advanced Research Project Agency
DEE	Design and Engineering Engine
DFS	Depth-First Search
DG	Dependency Graph
DICE	Distributed and Integrated Collaborative Engineering Design
DL	Description Logic
DMU	Digital Mock-Up
DR	Design Rationale
DRIM	Design Recommendation-Intent Model
DRL	Decision Representation Language
DS	Decision Support
FDBT	Fouille de Données et Bioinformatique Théorique
FOL	First-Order logic
HLP	High-Level Primitive
HTML	Hypertext Markup Language
IBIS	Issue-Based Information System
IEEE	Institute of Electrical and Electronics Engineers
IPP	Intellectual Property Protection
IT	Information Technology
KADS	Knowledge Acquisition and Documentation Structuring
KBE	Knowledge-Based Engineering
KBS	Knowledge-Based System
KIF	Knowledge Interchange Format
KR	Knowledge Representation
LGPL	Lesser General Public License
LIRIS	Laboratoire d'InfoRmatique en Image et Systèmes d'information
LSIIT	Laboratoire des Sciences de l'Image, de l'Informatique et de la Télédétection

Continued on next page...

MDO	Multidisciplinary Design Optimisation
MIT	Massachusetts Institute of Technology
MMG	Multi-Model Generator
MML	Multi-Models
MOKA	Methodology and tools Oriented to KBE Applications
OIL	Ontology Inference Layer
OLAP	Online Analytical Processing
OWL	Web Ontology Language
PDM	Product Data Management
PwC	PowerCopy
QOC	Questions, Options and Criteria
RDF	Resource Description Framework
RDFS	Resource Description Framework Schema
SOA	Services-Oriented Architecture
SOAP	Simple Object Access Protocol
SOC	Service Oriented Computing
SQWRL	Semantic Query-Enhanced Web Rule Language
STEP	Standard for the Exchange of Product model data
SUO WG	Standard Upper Ontology Working Group
SW	Semantic Web
SWRL	Semantic Web Rule Language
UDDI	Universal Description Discovery and Integration
UDF	User-Defined Feature
UML	Unified Modeling Language
URI	Uniform Resource Identifier
UUID	Universal Unique Identifier
W3C	World Wide Web Consortium
WSDL	Web Services Description Language
WWW	World World Web
XML	Extensible Markup Language

PART I

Introduction

General Introduction

Context and objectives

NOWADAYS, high-end industries such as the automotive or aerospace industries are designing products that are more and more complex and that integrate various disciplines in their design. The increasing sophistication of products, short development cycles and heavy competition lead to an increased complexity in the product development.

Since few decades ago, computers are used to assist designers in several tasks, such as design (Computer-Aided Design), manufacturing (Computer-Aided Manufacturing) or analysis (Computer-Aided Engineering). Computers have given the opportunity to create more complex products and to face new challenges.

One of these challenges concerns the collaboration between designers. The life cycle of industrial products is complex and involves several engineers with different knowledge and expertises, who are engaged in different activities for, under circumstances, a multitude of years. Furthermore they have different views on the product design according to their functional concerns. These views are translated into different models of a product, which need to be accommodated in a comprehensive description of the design product. Moreover these engineers can be located at different places of the world. This has led to the emergence of collaborative platforms that allow several stakeholders (or even teams) to work efficiently together on a project from distant places. In the design domain, collaboration is essential in order to successfully release a product in time and with good quality. However, the interaction between several engineers with different points of view is a source of conflicts and misunderstandings due to, for instance, differences between domains vocabulary.

Besides collaboration, another key factor and challenge in modern product design is the ability to reuse existing knowledge in design, products or processes. The standardisation and the use of common parts and platforms is a key factor for efficiency in several high-end industries, such as the automotive industry. Product diversification and the increase of the model range have motivated new Information Technology (IT) tools and have impacted the product development process. In this area, one major change during the last years is the emergence of Knowledge-Based Engineering (KBE). KBE is a large field at the crossroads of Computer-Aided Design (CAD), artificial intelligence and programming. KBE aims at facilitating the reuse of knowledge between designs. It results in a speed-up of the design and thus leads to a reduction of product design time and costs. From the various tools provided by KBE, this work is focussing on knowledge-based engineering templates.

Knowledge or KBE templates are intelligent applications that aim at storing know-how and facilitate its reuse. A knowledge template can be, for instance, a CAD model defined through KBE elements like formulas or scripts, in order to create dynamic components that can adapt themselves to various contexts. Templates are efficient solutions to share, for example, intelligent CAD models between several assemblies like cars within a model range. KBE templates are a recent technology that, in spite of their benefits, are currently used on a small scale by the industries. A hindrance to their adoption concerns their maintenance.

The two major issues of this emerging technology were defined and investigated: the collaborative search of design solutions for the update of a template, and the forwarding of the applied modifications to the instances of templates, which are the result of the use of a template in an assembly.

This work has been carried out in the scope of an industrial agreement for training through

research[1] (CIFRE). This agreement resulted in a collaboration between PROSTEP AG[2], a German company and two French laboratories. On the one hand there is the Laboratory of Computer Graphics, Images and Information Systems[3] (LIRIS) located in Lyon, wherein I was part of the Service Oriented Computing[4] (SOC) team. On the other hand is the Image Sciences, Computer Sciences and Remote Sensing Laboratory[5] (LSIIT) based near Strasbourg in which I was a member of the Data Mining and Theoretical BioInformatics[6] (FDBT) team. During this work I was supervised by Prof. Parisa Ghodous from the LIRIS laboratory and Prof. Pierre Collet from the LSIIT. Dr. Thomas Dusch was the industrial supervisor.

Contributions

This work resulted in the elaboration of a framework for the template update. A methodology and corresponding tools are presented in order to support defined tasks related to the template update.

A process has been defined in order to guide the users during the different steps of the methodology. The process is decomposed into three main parts, which contain the three main contributions.

The first part of the process supports engineers when looking for a design solution about the evolution of a template. For this purpose an argumentation-based decision support system is proposed. This system provides a structured decision process. It also allows to store and document the evolution of each template. This whole part eases the collaboration and allows to seamlessly document the template evolution.

The second part of the process is designed to create a computer processable and understandable representation of CAD documents and their relations. For this purpose an application ontology has been defined in order to represent and infer knowledge about CAD models, documents and their relationships. A software has been developed in order to extract information from CAD documents. The extracted information is then used to instantiate the ontology. The content of the ontology can be visualised in a tool that has been developed for this purpose. Thanks to the ontology and reasoning, explicit and implicit knowledge about documents are gathered in a unique location and can be used for many purposes.

In the third part, an algorithm has been designed in order to generate sequences of updates, which support engineers when forwarding the modifications applied to a template. This algorithm uses the knowledge present within the ontology for the computation of the sequence. The underlying CAD system is abstracted through generic concepts that are instantiated *via* inference on the ontology. The computed sequences allow to process documents concurrently if the necessary resources are available. Three approaches for the update of a template are also proposed. By using the generated sequence, the time consuming task of defining the update strategy is automated and thus the precious engineers' time is saved for more value-adding tasks.

The CATIA V5 CAD system has been used along this work for the illustration and as a foundation for the concrete applications. An analysis of this system has also be realised in the scope of this work.

Organisation of the manuscript

This manuscript is divided into six parts, for a total of thirteen chapters.
The first part is an introduction of the work.

- Chapter one presents the scope of the research work, describes knowledge templates, which are the main concepts addressed in this work, and introduces the studied issues.

[1] http://www.anrt.asso.fr/fr/espace_cifre/accueil.jsp?r=3&p=1
[2] http://www.prostep.com
[3] http://liris.cnrs.fr/?set_language=en&-C=
[4] http://liris.cnrs.fr/~soc/doku.php?id=current
[5] http://lsiit.u-strasbg.fr/
[6] https://lsiit-cnrs.unistra.fr/fdbt-en/index.php/Main_Page

The second part of the manuscript presents the state-of-the-art and related works.

- Chapter two presents different ways to represent knowledge. The focus is put on ontologies, the methodologies that are used to design them and also the available representation languages. Among the ontology representation language, the Web Ontology Language has been studied more extensively.
- Collaborative environments and decision support systems are presented in chapter three.
- Chapter four exposes some basics about graph theory, more especially about dependency graphs, dependency management and related algorithms.
- Chapter five introduces the current research status and applications of knowledge templates.

Part three is dedicated to a case study.

- Chapter six presents a study of the CATIA V5 CAD system that has been realised. It presents the template technologies, but also the links and relations within CATIA V5. A section is also dedicated to the available APIs to manipulate this system and access to the content of the CAD documents.

The contributions of this work are gathered in part four.

- Chapter seven provides an overview of the proposed approach to solve the templates update related issues. The designed process of the global methodology is described. It also gives a short introduction to each realised contribution.
- The proposed approach for collaborative issue solving is presented in chapter eight. In this chapter the main concepts of the system are detailed as well as their use and benefits.
- Chapter nine addresses the designed ontology. The followed ontology design methodology is presented as well as the result of each step of the methodology. Thereafter the resulting ontology is presented.
- Chapter ten describes how the modifications forwarding problem is defined and how the update sequences are computed. Two algorithms were tested in order to compute an update sequence. An approach to smartly retrieve documents from Product Data Management systems is also exposed. Finally three approaches for the actual template update task are described and compared.

Part five presents the realised developments as well as a scenario wherein the methodology is applied.

- Chapter eleven introduces the developed tools that support the methodology.
- Chapter twelve presents a scenario on which the methodology is applied and evaluated. In this scenario a template requires an update. The process will be followed in order to find a solution, apply it and forward the modifications to the instances of the modified templates.

Part six concludes the dissertation.

- Chapter thirteen summarises the realised work, explains the choices and points out the expected results from the application of this framework on real cases. Finally it opens perspectives and specifies possible enhancements to this work.

Chapter 1

Knowledge Templates Update Problematic

Contents

1.1 Knowledge in product design

1.1.1 Computer-aided design

Design is a creation process that in industry refers to the invention and development of a product or a service. Computer-Aided Design (CAD) is the use of computer technologies and software to assist persons in design activities, such as architecture, art or engineering. The engineering and product design field are the focus of this work.

CAD software started being used in the industry during the 1970's [Maculet and Daniel, 2004]. At this time CAD software were mainly two dimensional drawing tools. Their objective was to replace the drawings on drafting tables. CAD software provide graphical tools and component libraries that speed up the design. Furthermore they allow computer supported archiving of product models. The next step was the three dimensions design software that were the outcome of the increase of computers' processing power and research work in computer science fields like 3D computer graphics. This volume modelling was pushed by aerospace and automotive industries in the 1980's. Today in addition to the 3D design approach, most CAD systems are feature-based, which means that the design is based on the combination of sketches and operations enriched with a functionality (features), such as holes, chamfers, extrusion, rotation, etc. instead of primitives that can be added or subtracted.

Besides CAD, other technologies related to product design have to be mentioned:

Computer-Aided Engineering (CAE) is a domain that embraces CAD. Its aim is to support engineers in various tasks that include design, analysis, manufacturing, simulation, etc.

Computer-Aided Manufacturing (CAM) consists in assisting engineers in the manufacturing process. McLean [1993] described it as "the use of computerized tools in the application of scientific and engineering methods to the problem of design and implementation of manufacturing systems." The common process is that CAD models are transferred to a CAM system that allows the designer to define a sequence of manufacturing processes or instructions that can be understood by numerically controlled machines.

Digital Mock-Up (DMU) aims at reducing the number of physical product mock-ups in early design phases by providing digital ones. The DMU technology allows the virtual simulation of products and their components at a lower cost and lead to a reduced time to market [Jackson, 2006].

Nowadays CAD software allow to integrate more than just geometrical concepts into the CAD models. Within the two past decades, information about the product and its design intent, more specifically knowledge, has been integrated and is now managed by design software. The use of this knowledge has become popular in CAD and is referred to as Knowledge-Based Engineering.

1.1.2 Knowledge-based engineering

1.1.2.1 Definition of knowledge

In the literature various definitions of the term "knowledge" can be found. Some are philosophical such as "justified true belief" from Plato. Others are more recent and were proposed in the scope of knowledge-based systems. Frost [1986] defined knowledge as "the symbolic representation of aspects of some named universe of discourse." Milton [2008] proposed a more narrowed definition that focuses on tasks and on the context where it is used. His definition is condensed in a sentence that is presented in figure 1.1.

The Oxford English Dictionary's definition of knowledge is defined by the three following statements:

- The expertise and skills acquired by a person through experience or education: the theoretical or practical understanding of a subject.

$$\text{Knowledge is the } \left\{\begin{array}{c}\text{ability}\\\text{skill}\\\text{expertise}\end{array}\right\} \text{ to } \left\{\begin{array}{c}\text{manipulate}\\\text{transform}\\\text{create}\end{array}\right\} \left\{\begin{array}{c}\text{data}\\\text{information}\\\text{ideas}\end{array}\right\} \text{ to } \left\{\begin{array}{c}\text{perform skilfully}\\\text{make decisions}\\\text{solve problems}\end{array}\right\}$$

Figure 1.1 – Definition of knowledge [Milton, 2008].

- What is known in a particular field or in total: facts and information.
- Awareness or familiarity gained by experience of a fact or situation: "He denied all knowledge of the incident."

As you can see, there is no single agreed definition of knowledge.

In the design domain, the knowledge are the know-how and best practices. It usually comes from the experience and is a strategic resource.

Knowledge-based Engineering (KBE) is a large field at the crossroads of Computer-Aided Design, artificial intelligence and programming. It aims at the capture, the storage and the reuse/-transfer of domain expert design knowledge, design intent, best practices and know-how. As the engineering process becomes increasingly more complex and the competition requires shortened time to market and cost reductions in developments, KBE has become usual for the design of complex systems. Stokes [2001] defined KBE as "the use of advanced software techniques to capture and reuse product and process knowledge in an integrated way." To this definition, KBE can be seen as the meeting of design automation and knowledge management.

1.1.2.2 Design automation

Automation is the action of making "a process in a factory or an office operated by machines or computers, in order to reduce the amount of work done by humans and the time taken to do the work."[1] In mechanical design, the increase of the product complexity during the last decades has led to the automation of many tasks in manufacturing but also in design. In this domain, automation is the combination of parametric design, formulas, rules, scripts and software programs. With automation it is possible to create dynamic CAD models that react on parameter change and, for instance, modifying the contained geometry. The process of generating geometry by using a set of rules or algorithms is referred to as generative design [Phillips, 1997]. It allows to rapidly and easily create various design variants. The figure 1.2 presents three different configurations of a platform and its staircase created through automation. Nowadays automation aspects are included in most CAD software.

(a) (b) (c)

Figure 1.2 – Design variants generated by parameter changes within CATIA V5 CAD system.

[1] Definition from Cambridge dictionary

However Prasad [2005] considered that some KBE applications are not "true" KBE applications. He pointed out five qualities that describe "true" KBE application:

Dynamic: data is updated based on input changes.

Generic: it can be adapted to various situations.

Generative: new geometry or bodies can be created automatically from old ones based on changes in the input specifications.

High-level: a small amount of KBE code produces significant impact on objects.

Demand driven: the system is aware of the sequence in which the rules are triggered (the user has not to worry about it).

Alvarado *et al.* [2007] talked about intelligent automation within companies. Intelligent automation is a trend in industrial automation that needs systems that are able to handle knowledge and information more efficiently. This trend comes from numerous factors like competitive pressure, reduced time, complexity, flexibility, etc. It has an impact on CAD systems, process control, management, etc. and goes by knowledge management aspects. The application of intelligent automation in design is considered as KBE.

1.1.2.3 Knowledge management

In the automotive and aeronautic fields, manufacturers have been faced to several changes in their environment since several years. They have to adapt to the market where products become more and more complex and innovative. At the same time the products' time to market gets shorter and the amount of data to handle keeps on growing. All these changes have brought about the age of industrial automation where knowledge and know-how has become a key business asset [Alvarado *et al.*, 2007].

Knowledge Management is a topic that gathers techniques and tools in order to take advantage of intellectual assets in the company [Grundstein, 2000]. It aims at identify relevant knowledge, extract it, then store and represent it in a suitable way, and finally facilitate its reuse. Companies introduce comprehensive knowledge management at the end of the 90's for its various benefits:

- Prevent knowledge loss (retirement, turnover...)
- Facilitate knowledge sharing, transfer within the organisation
- Improve/support training
- Automate tasks
- Future knowledge reuse

Another aspect of knowledge management is the protection of this knowledge. As knowledge has become a key asset, companies need to protect it and prevent knowledge leaks. Today, large companies outsource parts of their production or development to companies that can be located anywhere in the world. Within this collaborative environment, documents containing company's know-how are exchanged. This can lead to knowledge leaks or plagiarism. To prevent company's intellectual property from being "stolen," it has to be protected. This is called Intellectual Property Protection (IPP). It has become a hot topic within KBE field. Examples of IPP solutions are access protection or knowledge filtering [Antegnard *et al.*, 2006].

However knowledge management is a complex task, especially the capture of experts' knowledge. It is hard to acquire and formalise implicit, or also called tacit knowledge, which is knowledge difficult to express as it can come from experience, know-how, intuition or automatisms. Explicit knowledge is, contrary to tacit knowledge, what can be written down, shared or expressed. Polanyi [1967] calls "tacit knowing" as the fact that "we can know more that we can tell." To support knowledge management steps, some methodologies and tools were developed.

1.1.2.4 Knowledge management methodologies in engineering design

In order to provide a structured way to identify, collect, structure and formalize engineering knowledge, several methodologies where created. These methodologies provide tools and define processes to guide and to allow the development of Knowledge-Based Systems (KBS). KBS are computer systems that can infer, explain or support decision by using artificial intelligence tools on knowledge. The main challenge remains to efficiently capture and represent the information.

One of these methodologies is the Knowledge Acquisition and Documentation Structuring (KADS) [Wielinga *et al.*, 1992]. It is a knowledge acquisition methodology to support knowledge-based systems development, whose origins are in the European ESPRIT project P1098a [ESPRIT]. CommonKADS [Schreiber *et al.*, 1999] is a new methodology that has grown out of KADS and that aims at covering the entire knowledge-based system life cycle.

Protégé-II [Musen *et al.*, 1995] is a framework that supports KBS creation. It is composed of two main tools, MAÎTRE that allows to browse and edit domain ontologies, and DASH that generates a specific graphical knowledge acquisition tool. A library of problem-solving methods is also provided in order to solve application tasks with the acquired knowledge.

Another available methodology is the Model-based and Incremental Knowledge Engineering (MIKE) [Angele *et al.*, 1998]. It is an incremental process that integrates a formal, a semi-formal and an operational description formalism. The formal and executable model is specified with the Knowledge Acquisition and Representation Language (KARL).

However these methodologies are considered as too generic to address KBE and, except CommonKADS, are not much used to build KBE applications. Thus more specific methodologies to address KBE were developed. The leading KBE methodology is called MOKA which stands for "Methodology and tools Oriented to KBE Applications." MOKA [OLDHAM *et al.*, 1998; Stokes, 2001] is the result of an European project, started in 1998 with a duration of 30 months, which the main goal was to provide a methodology for developing and maintaining KBE applications. It also aims at reducing the cost and time of developing KBE systems as well as to provide tools to support the methodology. Figure 1.3 presents the life cycle of KBE as it was identified in the MOKA methodology. The life cycle is composed of six phases:

1. Identify and define the requirements, aims, the scope, and knowledge sources for the KBE system.
2. Estimate the costs, resources requirements and the project risks.
3. Capture and model knowledge from domain experts by using ICARE (Illustration, Constraints, Activities, Rules, and Entities) forms to create an informal model [ICARE forms].
4. Convert the knowledge into a formal model based on the MOKA Modelling Language which is based on UML.
5. Develop software applications for the system based on the formal model.
6. Distribute and support the KBE system to end users.

More focussed on the storage, Liese [2003] proposed an object-oriented 3D-CAD representation for design knowledge. Knowledge relative to the function, shape, behaviour and methods are addressed. A matrix, called WA-RE matrix, organising in a hierarchy and classifying knowledge types and knowledge representations has been defined. It is a central component of this approach in order to represent each type of knowledge the right way. Then knowledge can then be stored into CAD models and parametrised CAD models by using the different tools provided by the various CAD software, in order to reuse knowledge.

1.1.3 Applications of KBE

Crabb [1998] has predicted that KBE will have as much importance in 2010 for companies as CAD/CAM/CAE had in the 1990's decade. The numerous applications that use and show the benefits of KBE confirm this statement.

[1] `http://web1.eng.coventry.ac.uk/moka/lifecycle.htm`

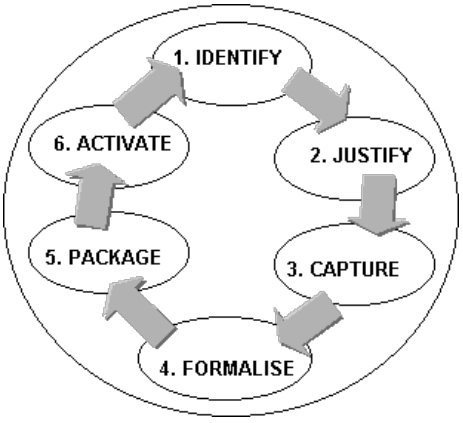


Figure 1.3 – KBE application life cycle as identified in MOKA[1].

Chapman and Pinfold [2001] presented a KBE system able to generate adequate models for the analysis of CAD models. An application on "Body in White" was presented, which is the metal sheet structure of a car, wherein finite elements analyses are undertaken to test the structure. By generating adequate models, the KBE system avoids the duplication of models and speeds up the response time after design changes. van der Elst and van Tooren [2008] presented a KBE application coupled with optimisation that reduced the reassignment time of aircraft electrical wire pins by 80%. This has been achieved through the reduction of recurring tasks.

More generally, KBE shows its advantages for reducing design time and thus saves time for creative or innovative tasks, but also gives the possibility to generate an increased number of design concepts with the same model. Figure 1.2 presents different configurations of a platform and its corresponding staircase. Configuration (a) shows the platform at three meters height. In configuration (b) the height is changed to two meters and also the type of the guardrail. From configuration (b) to (c), the spacing between the steps has been increased, the angle of the stair changed from 45° to 38° and the guardrail fixation was modified. Changing the configuration took only a few seconds to be effective whereas designing a new platform or modifying an existing one could take hours. Furthermore the validity of the models is checked by rules.

Figure 1.4 shows the theoretical influence of KBE on design tasks, where one can see that routine design tasks take a shorter time. This time reduction profits to creative design and to the reduction of global project duration. The efficiency of KBE is obviously depending on each applications and their knowledge reuse possibilities.

KBE also allows to represent the multidisciplinary aspects of products [La Rocca and van Tooren, 2005]. KBE started to be used in design but it has rapidly reached other fields like manufacturing or analysis such as for the Finite Element Method [Kulon *et al.*, 2006].

Thanks to KBE it is also possible to create intelligent CAD models in a generic way so they can be used in various contexts. For instance, Skarka [2007] applied the MOKA methodology to ease the creation of generative models in CATIA V5. MOKA's forms were used to capture knowledge whereas the formalisation and the packaging of the knowledge was realised within CATIA V5.

1.1.4 Summary

The main usage of KBE is to support and improve the design of complex mechanical systems by automating repetitive and non-productive activities. It also allows to enhance the product quality and to reduce time to market and costs. Hence KBE allows to take a competitive advantage [Gay, 2000]. This is achieved through the capture and the reuse of knowledge. Strength of KBE in product development is provided by automation tools like scripts or rules, which bring intelligence and a knowledge storage solution. By using automation, KBE also allows to easily generate various design variants from a single model.

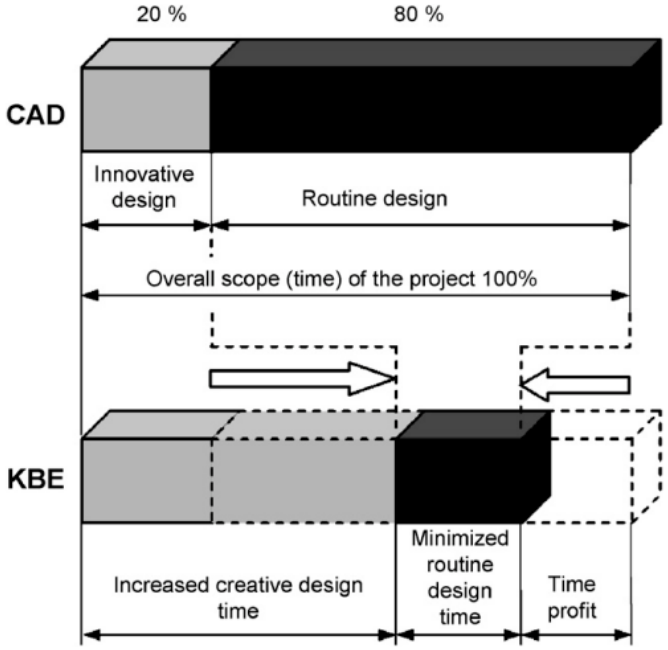


Figure 1.4 – Benefits of KBE use on main design tasks [Skarka, 2007].

Sandberg [2003] listed benefits and drawback of KBE. The major benefit is the reduction of the lead time for product development. The time reduction concerns products that own the three following properties: high degree of similarity to reuse knowledge, large amount of design configurations and large number of design processes that can be automated. Product optimisation is also easier because better configurations can be found within a shorter time. The knowledge captured within models reduces the risks related to staff turnover. The last listed benefit is related to the automation of repetitive work leading to an increase of time available for innovative tasks. As drawback Sandberg [2003] pointed out the time needed to build correctly KBE models. The added value of resorting to KBE should be evaluated regarding the time needed for its implementation. KBE should be avoided in some circumstances like when the design is too simple, when technologies are continuously changing or when it is not possible to access or isolate product knowledge [Stokes, 2001]. Regarding knowledge transfer efficiency, KBE users should still be able to access the definition and not use KBE elements as "black boxes."

As introduced in this section, KBE is a large domain that can not be comprehensively presented here. In the scope of this work, the focus is oriented towards KBE templates, which are generative and intelligent applications.

1.2 KBE templates

1.2.1 Template concept

Knowledge-based engineering templates are intelligent and generative applications that aim at storing design knowledge and restore it in an easy, fast and convenient way. They have the capability to adapt themselves to a given context regarding some provided inputs from the context, in order to fulfil a function, such as provide geometry or do some calculus. This is achieved thanks to KBE elements, *e.g.*, formulas, rules, scripts, etc. Templates can be considered as ready to use models that can be referenced and used from libraries in the same way as CAD design features. An example of KBE template could be a car wheel CAD model that takes as input the diameter, the width and the type of rim. With one model it is possible to generate multiple wheels for one

or more cars with different configurations in a few seconds. In this way, future developments will be faster and of a better quality as previous mistakes would be avoided.

The template concept is referred to by several terms in the literature: "High Level Primitives" [La Rocca and van Tooren, 2005], "CAD/knowledge/KBE templates" [Alani, 2007; Arndt, 2007; Katzenbach *et al.*, 2007] or simpler, "generative models" [Skarka, 2007]. All these designations are valid and it has been decided to refer to them as "KBE templates" as it is self explanatory. The term "template" refers to a generic and reusable element like text processor document template, which has a predefined and ready to use style. The acronym "KBE" defines the domain of the template as well as its foundations. KBE templates are also referred to as knowledge templates or simply templates in the rest of this manuscript.

The use of knowledge templates has been pushed by numerous factors. Templates are used to store and reuse specific know-how or best practices within a company. This knowledge can concern a component or a process. Production process related aspects can also be integrated into the template. Another aspect is that templates can contain workarounds to specific design problems that can appear in specific configuration. These workarounds can be triggered thanks to rules and thus avoid engineers to fall into common issues. The templates mechanisms can also be used to provide standardised models. The standardisation and the use of common parts and platforms is a key factor for efficiency in the automotive industry [Dudenhöffer, 2000]. So one template document can contain several standardised configurations of a part. This will simplify the maintenance of the set of standard models of the same type as all variants will be gathered within a single model.

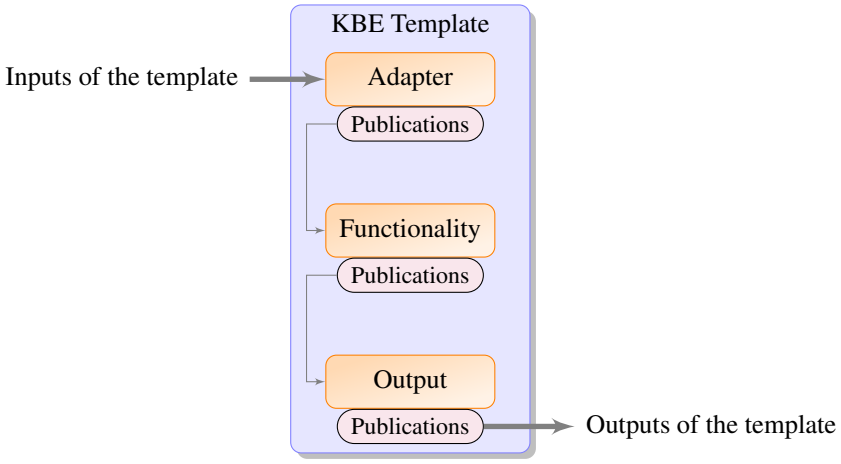


Figure 1.5 – Generic structure of a KBE template with data flow.

Figure 1.5 presents the generic structure of KBE templates. A template can be seen as a box with inputs that are set during the instantiation process (detailed in section 1.2.3) and outputs. The inputs and outputs can be of various types like geometry elements or parameters. Basically the content of the template can be divided into three parts. First the "adapter" that gathers the inputs of the template. The adapter model, which is also sometimes referred to as skeleton, is composed of basic geometry elements (point, lines, planes...) that drives the "functionality" part of the template. An example of an adapter model is depicted in figure 1.6, which presents a skeleton driven by KBE elements and some geometrical inputs. The main components of a template is the "functionality" part, which implements the function of the template, *e.g.*, calculus or geometry generation for a CAD template. It is driven by references on the adapter model and thus morphs with it. The "output" part is an interface, which provides references to specified elements within the functionality part. The template designer can expose important component or data of the template, so they can be quickly identified be the template users. Data flow (arrows) is based on "publications," which are formal outputs. The aim of using publications is to provide a named reference of an element within the document, that can be easily recognised and referred

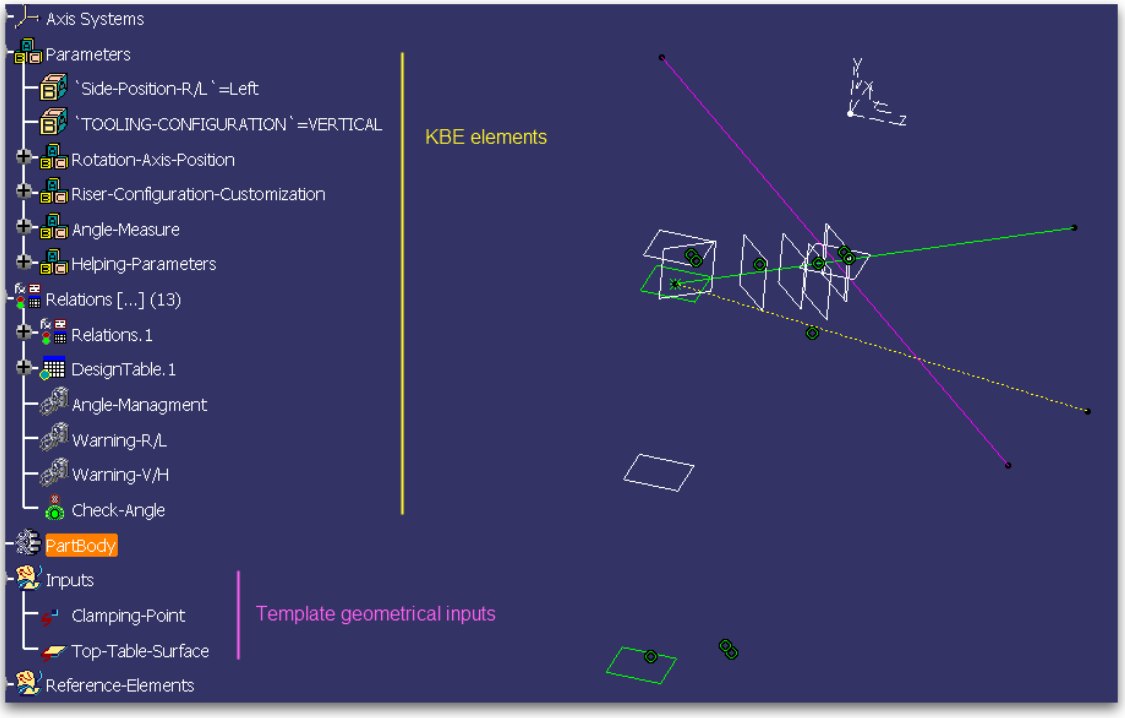


Figure 1.6 – Adapter model/skeleton example within CATIA V5 showing KBE elements and some inputs.

to. So if the content of a document changes, the links between documents will not be broken as the elements inside the document are not directly referred to. Figure 1.5 also shows that templates have a hierarchical structure.

Templates are "living elements," which evolve during their life cycle. Figure 1.7 shows the various phase of a knowledge template life cycle. Templates are designed to fulfil a function. They are an appropriate approach in order to integrate proven concepts into new products [Katzenbach *et al.*, 2007]. So the first step is to design the template structure and content in order to provide the functionality. Once created, it is tested in order to fix any issue before putting it into production. The package step aims at gathering template resources that are elements related to the template design, such as testing contexts or the template's documentation. Then comes the deployment step where the template is made available to engineers who can use it in their design. The following step concerns template maintenance. Templates can be updated for various purposes such as to fix

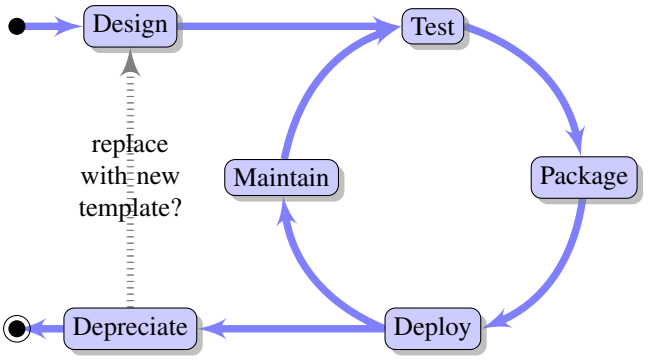


Figure 1.7 – Knowledge template life cycle.

Table 1.1 – Detail level classes of templates according to Arndt *et al.* [2006].

Template class	Detail level	Examples
Function template	10%	Automotive
Concept template	↓	Limousine, Cabriolet, SUV
Version template		Class C, Class S
Model	100%	Mercedes S 350

bugs or to fulfil new requirements. For each update a new version of the template is created. After the update, the template goes one more time through the test and package phases before being deployed. The last step of a template life cycle is its depreciation. Within this step, the template is removed from production and may be replaced by a new one.

Before deploying a template, it has to be packaged. A template package contains the template with a set of resources that are needed or used by the template during its life cycle. The package contains tests resources like specific contexts used to validate the template, the template documentation and scripts. Template resources can also be shared between several templates, for example, the same validation script can be used by various templates.

1.2.2 Classification of template types

Design templates can be utilised for a broad range of CAE/CAD/CAM design tasks. This section summarises the various template classifications found in the literature.

Arndt *et al.* [2007] defined three main template types. Geometrical templates or CAD templates that are parametric CAD models. Study templates that are CAx models for validation. Downstream or process templates that are CAx models based on reliable computation and manufacturing principles.

Another classification proposed by Mbang [2008] focused on feature templates. Feature templates are small construction elements that have the same purpose as templates. These feature templates are used to raise the detail level of models or other templates. Mbang [2008] differentiated two categories of feature templates. First, functional features that are features having a direct relationship with the design process, such as standardised holes or a flange. Second, formation features that are related to the forming process, *e.g.*, flanges or corrugations. Both categories are created according to the design process information stored in the template.

Arndt *et al.* [2006] introduced a classification according to the detail level of templates. This concerns especially geometrical templates. They defined four classes that are presented in table 1.1. The higher level class, called function template, provides few details, but a basic product structure and function elements. An automotive template with the position of the four wheels, the engine position at the front, etc. is given as example. The concept templates provide a more detailed version and variant of function concepts. For the car example, concept templates could be a type of car, such as a cabriolet or a SUV. Finally there are the implementation templates. They are assemblies or models that provide a high level of details, close to the complete geometry representation.

Katzenbach *et al.* [2007] presented another view that makes the correspondence between external factors and a template-based design phases (see figure 1.8). The concentric layers present the template-based design phases with the corresponding template types. These template types are close to those presented by Arndt *et al.* [2006]. The evolution of the phases goes from the outer ring towards the center. The first phase at the periphery corresponds to the function-template. Like the previous definition it is a model providing rough geometry. Then comes the concepts template that includes specific characteristics. Study-templates are used to validate the functional principles put in place in the previous phase. The part-template phase corresponds to a detailed geometry based on the previous implemented concepts within the model. Like in table 1.1, the degree of detail increases when going towards the centre. All over the figure are reported the external factors that should be taken into account for each phase.

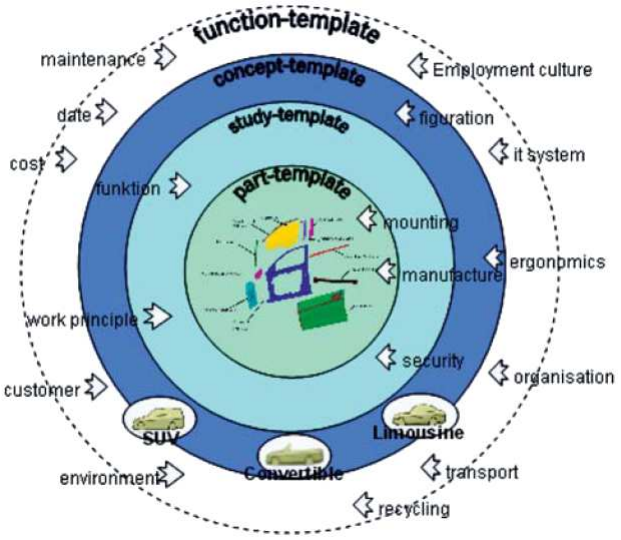


Figure 1.8 – Template-based design phases represented by the rings with their corresponding template types, levels of detail and external factors [Katzenbach *et al.*, 2007].

The process of using a template is called instantiation and results in the creation of a template instance.

1.2.3 Template instances

Instantiation is the process that consists in creating a copy of the template and of putting it into a specific context. The context defines where the template is used. It can be an assembly, a part, a process or even another template, where the latter would lead to template interweaving. During the instantiation process the inputs of the template instance are assigned. Figure 1.9 presents the structure resulting of the instantiation of a template. The result obtained is a template instance

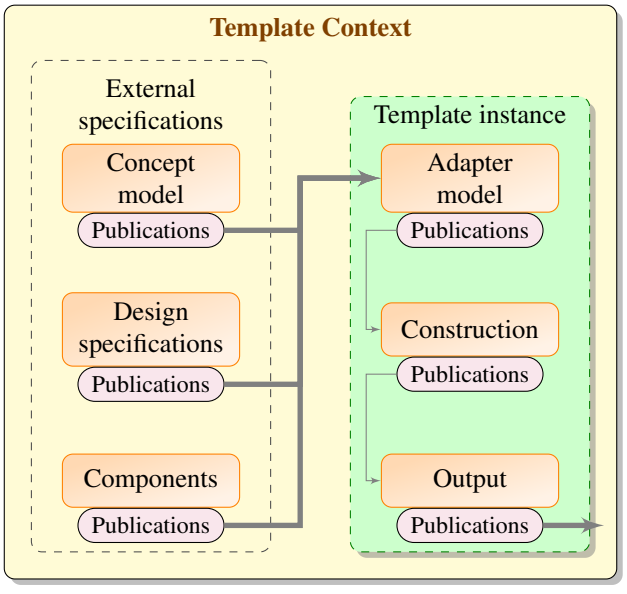
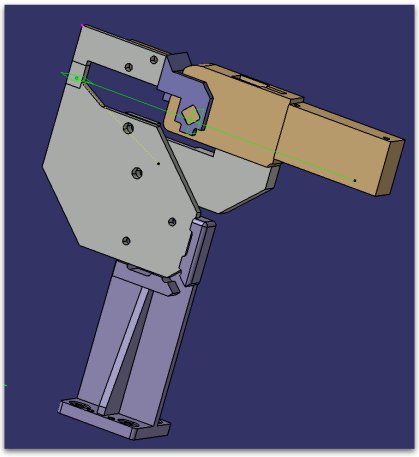


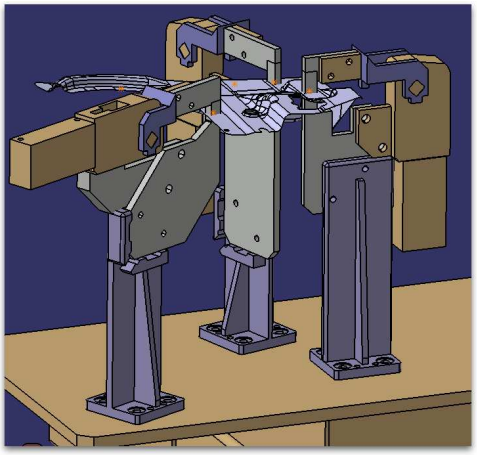
Figure 1.9 – CAD template in its context with link flow [Arndt *et al.*, 2006].

that receives input data from various sources, called "external specifications." Data sources are the concept model, which is a very high level model containing the global product structure. The design specifications are related to the requirements, for example, the length of a part. Components are other elements present in the context such as CAD parts. Once the instance is in the context and the inputs are assigned, it adapts itself according to the inputs. The link flow represents how the data is exchanged between the different parts. Thanks to KBE elements, such as formulas and rules, the adapter model containing basic geometry will be configured. Then elements present in the construction part will follow the modifications of the adapter model by reading data from the published elements they refer to and this way, generate or adapt their geometry in the case of CAD template instance.

Figure 1.10 illustrates the process. The figure 1.10(a) presents a clamp template, which was designed with CATIA V5 that is a CAD software from Dassault Systèmes. The result of the instantiation can be see in the figure 1.10(b). Three instances of the template can be seen that are present in an assembly, which is here the context. The instances are holding a metal sheet and present various configurations. Each instance has been created from the template in less than a minute by giving as input the table surface and the clamping point.



(a) Example of KBE template in CATIA V5 CAD software



(b) Three template instances from the same template with various configurations holding a metal sheet

Figure 1.10 – Example of a template and three of its instances in a context.

1.3 Addressed template update issues

In this dissertation issues related to template maintenance and update are investigated and solved. In large and complex assemblies like those present in automotive or aerospace industries, the number of templates and template instances can reach several thousands and even more. This implies a huge effort to manage and maintain templates as they become even more complex by incorporating new potential variants for future design [Katzenbach *et al.*, 2007]. To support engineers in template update management, two issues are addressed. First the decision support regarding the template updates, second the update of the template's instances.

1.3.1 Template evolution decision support

The template development is an incremental and collaborative process in which templates are continuously refined. As shown in figure 1.7, knowledge templates follow an incremental process during their life cycle. Templates undergo modifications in order to solve an issue or enhance the

template. For instance, a reason of modification could be a bug report of the request for additional capabilities.

The template update is a complex process. It involves various stakeholders whose goal is to find the best solution to the reported issue or the feature request. The stakeholders have their own point of view on the template design which can be related to a domain (like electrical requirements or costs) or the template ease to use. Due to the heterogeneity of competence fields involved in the design of a product, possible conflicts are evident and thus a trade-off has to be found. Finding a solution to repair or to enhance the template while taking into account the various constraints and aspects involved in its design is quite a complex task.

The situation of template update decision support in a collaborative and multi-disciplinary environment is studied. The objective is to propose a solution that supports the decision process and to provide new ideas that could enhance the template update task.

1.3.2 Template updates propagation support

A current problem when using templates is the synchronisation between the template definition and its instances. That means that the latest modifications made to the template should be forwarded to the instances. In this way instances can benefit of new functionalities and bug fixes. It also facilitates the management of template instances as they will all have the same definition, *i.e.* their content is the same as the content of the corresponding template. So template managers will not have to handle several versions at the same time. Thus the consistency between the template instances and their definition has to be assured.

However this synchronisation between templates and their instances is not fully handled by current Product Data Management systems [Lukibanov, 2005]. Hence the modifications have to be applied by hand to the instances of the template. When working with large assemblies that can contain several thousands of template instances, finding a feasible strategy to update them all is a challenging and time consuming task. The complexity of the interdependencies network within KBE assemblies make this task even harder as the order in which instances are updated has a significant impact on the result and might generate time consuming redundant updates. Hence the establishment of the order in which the instances have to be updated is a difficult task that requires a lot of time that can lead to errors. Moreover a template can contain instances from other templates creating a complex interweaving that has to be taken into account. Figure 1.11 illustrates the instances interweaving and how it appends. Template A has two instances, one in template B and the other in an instances of template B.

So the second addressed issue by this work concerns the forwarding of the template updates to theirs instances. To efficiently achieve this task, engineers need an efficient methodology and supporting tools.

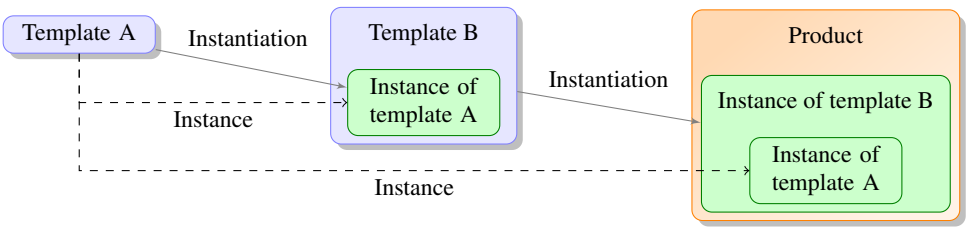


Figure 1.11 – Example of instances interweaving through the instantiation of a template containing the instance of another template.

1.4 Chapter summary

Knowledge-based engineering has revolutionised product design by bringing together knowledge management and automation. This mixture has resulted in the enhancement of the quality of products as well as the reduction of non-productive activities and of the global project duration. These benefits goes by the capture, the storage and the restitution of design knowledge, which is supported by KBE. KBE comprises various tools like rules, scripts and also knowledge templates.

Knowledge-based engineering templates are intelligent applications that are able to adapt themselves to a given context in order to fulfil a function and restore encapsulated knowledge. This adaptability is achieved by using KBE elements. Several types of template are available, which provide, for instance, geometry, calculus, processes, etc. The use of a template is called instantiation. It results in the insertion of an instance, which is a copy of the template, in the target context. The context could be an assembly or even another template. The use of KBE templates allows to reduce time and costs as well as to improve the quality of newly designed products.

The work presented in this manuscript addresses the maintenance and more especially the update of templates. As will be shown in the following chapters, few methodologies address template maintenance. However their maintenance is a complex and time consuming task. The first addressed issue is to support the collaborative decision making of a solution to solve a template issue. The second issue concerns the support of the propagation of the modifications made to a template towards its instances.

The following chapters will first introduce the related work found in the literature. Then a study case about template use is presented, in order to understand the real constraints and mechanisms before presenting the proposed approach to solve these issues.

PART II

State of the art

Chapter 2

Knowledge Representation

Contents

2.1 Knowledge and computer systems

Since the 1950's and the beginning of the researches in artificial intelligence (AI), humans wrote computer programs able to solve algebra problems or prove logical theorems [Russell and Norvig, 2003]. Computers were able to "think." This was achieved by integrating human knowledge into the software.

During the last decades several knowledge technologies appeared [Milton, 2008]. They are computer based technologies that are able to process formalised knowledge: identify important knowledge, capture, integrate, represent, store, reason, and decide. Milton [2008] identified four subject areas where knowledge and information technologies work together:

Knowledge management is a vast field that comprises methodologies with the purpose to manage, identify, capture, and distribute knowledge within an organisation.

Knowledge engineering "is an engineering discipline that involves integrating knowledge into computer systems in order to solve complex problems normally requiring a high level of human expertise" [Feigenbaum and McCorduck, 1983]. This field includes specialities such as data mining that consists in the extraction of unknown knowledge from databases, expert systems, case-based reasoning that consists in solving new problems by identifying patterns and rules from previous cases, software agents that are reactive, autonomous, social and independent entities that can sense and act within an environment in order to realise a goal.

Knowledge-based engineering aims at supporting design engineering (see section 1.1.2).

Ontological engineering tackles the design and maintenance of ontologies. An ontology is a formal representation of "what exists." More details about ontologies are provided in section 2.2

All these domains are faced to the same problem, which is the knowledge representation (KR). On the one hand, it is possible to represent objects, properties, facts, relations, causes and effects and many other thing that can be classified as explicit knowledge. On the other hand, it is hard to represent tacit knowledge. Polanyi [1967] defined "tacit knowing" as the fact that "we can know more that we can tell." For instance, in a chess game a player would avoid a particular move because he feels that his piece would be too exposed. In order to solve problems, to support people or to have an intelligent behaviour, machines require a wide range of knowledge on the context, the environment or the world.

Davis *et al.* [1993] proposed five crucial roles for KR:

- KR is a *surrogate* of the thing itself in order to allow an entity to think and reason about it.
- KR is a *set of ontological commitments*, *i.e.*, to focus on some parts of the world because the complexity of the natural world is overwhelming.
- KR is a *fragmentary theory of intelligent reasoning*. It is a fragment because the representation incorporates only part of the belief that motivated it, and the intelligent reasoning is related to what can be inferred.
- KR is a *medium for efficient computation* as the KR will be used by a computer. The representation will have an impact on the computations.
- KR is a *medium for human expression* about the world. The recipient can be either the machine or other people.

Woods [1975] provided a shorter definition of what a KR language should be: "A KR language must unambiguously represent any interpretation of a sentence (logical adequacy), have a method for translating from natural language to that representation, and must be usable for reasoning."

From these definitions, major elements of a KR can be pointed out. It should be able to provide a *unambiguous* representation of a *part of the world* for *reasoning* purposes. To go in this direction, ontologies will now be presented.

2.2 Ontologies

2.2.1 Definition

Ontologies originally appeared in philosophy, where it defines the systematic account of Existence or can be described as the study of being. The world *Ontology* comes from Greek where it means "be" -*onto*- and "science, study" -*logos*. However the word appeared only during the 17^{th} century. Since then the discipline has evolved and came closer to cognitive sciences two decades ago [Psyché *et al.*, 2003].

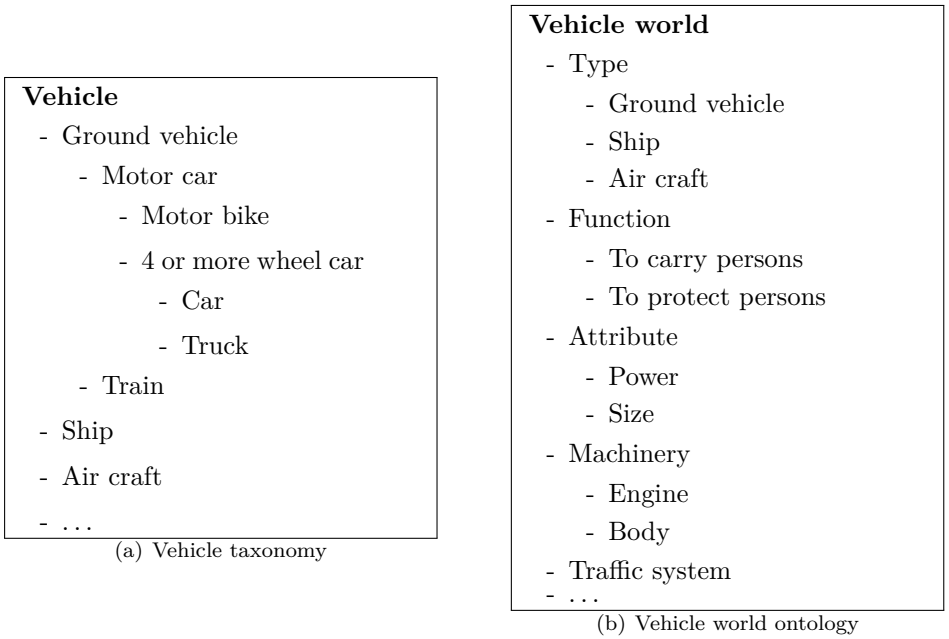
The first mention of ontology in the computer sciences field was by McCarthy [1980] during a work on ontologies in the philosophical meaning and the construction of logic theory of artificial intelligence systems. He stated that logic-based intelligent systems designers should enumerate all existing and build an ontology of the world. Nowadays one of the most popular definition of an ontology in Artificial Intelligence is the one from Gruber [1993a]: "An ontology is a specification of a conceptualisation. The term is borrowed from philosophy, where an Ontology is a systematic account of Existence. For AI systems, what 'exists' is that which can be represented."

To understand this definition, the term "conceptualisation" is must be clear. According to Andersen and Vasilakis [2007], "the world view is often conceived as a set of terms (such as entities, attributes, and processes), their definitions and interrelationships; terms denote important concepts (classes of objects) in the domain. This is referred to as a conceptualization." Borst [1997] stated that a conceptualisation is a structured interpretation of a part of the world that people use to think and communicate about the world. He has also given an extension of Gruber's definition: "An ontology is a *formal* specification of a *shared* conceptualisation." Borst [1997] definition brings forward the collaborative aspect of ontologies which takes sense in a shared context. When designing an ontology, there must be an agreement on the concepts to specify. Mizoguchi [2003] proposed another definition, which focuses on the content. "An ontology consists of concepts, hierarchical (*is-a*) organisation of them, relations among them (in addition to is-a and part-of), axioms to formalise the definitions and relations." It can be considered as the actual definition of ontologies in the computer sciences.

When the word "ontology" became popular in the knowledge engineering community, Guarino and Giaretta [1995] listed at least seven different interpretations of the term "ontology." They also proposed to weaken the definition from Gruber [1993a] to a "logical theory which gives an explicit, partial account of a conceptualization" to point out that an ontology can not specify a conceptual element in a comprehensive way. So Guarino and Giaretta [1995] noticed that there are some possible misunderstanding. Hepp [2007a] evaluated the situation recently and noticed that there is still a lot of inconsistency in the usage of the term "ontology." He listed three aspects that are the common roots of disagreement: what to model (the "true" structures or a consensus on them), how to represent the ontology (is it necessary that an ontology is represented with a formal logic) and if the ontology is the conceptual system or its specification.

Mizoguchi [2003] proposed several definitions of what an ontology is not. First an ontology is not just a set of terms. A clear distinction should be made between terms/words and concepts. An ontology is a theory of concepts and it doesn't matter of how the concepts are called. A name is put on each concept to make it human-readable. Ontologies have a hierarchical structure composed of "*is-a*" relations like taxonomies. There is sometimes a confusion between ontologies and taxonomies. A taxonomy is a classification in a hierarchical structure with subtype/supertype relationships. A good taxonomy separates elements of a group (taxon) into mutually exclusive subgroups (taxa). Figure 2.1(a) represents a vehicle taxonomy. This representation is not sufficient to understand what a vehicle is. An ontology is more than a classification as it also represents attributes, relations, functionalities, etc. Figure 2.1(b) presents an extract of a vehicle ontology. In an ontology more concepts are available such as what machinery the vehicle has, its size, how many persons it can carry... At least two things distinguish ontologies from taxonomies: ontologies have richer internal structure and they are the result of some consensus.

Ontologies are not a knowledge representation [Mizoguchi, 2003]. An ontology gives a mean to model the world and is independent of its representation. Ontologies are like conceptual schemas in



(a) Vehicle taxonomy

(b) Vehicle world ontology

Figure 2.1 – Example of vehicle taxonomy and ontology [Mizoguchi, 2003].

data base systems [Gruber, 1993b]. Conceptual schemas give a logical description of shared data so applications can interoperate without sharing the underlying data structure. In the same way, ontologies define terms with which knowledge can be represented. Ontologies define the vocabulary to compose coherent sentences, which can be understood by persons/applications committed to the shared ontology.

2.2.1.1 Ontology components

As said before, ontologies are a mean to model a domain. To achieve this, ontologies are composed of the following components [Andersen and Vasilakis, 2007; Gómez-Pérez, 1999; Gruber, 1993a]:

Concepts: Also called term or class of an ontology. A concept is used to encapsulated the meaning of anything about which something is said (task, function, colour, feeling...). Concepts are usually organised in taxonomy. Example of concepts could be *Human* and *Animal*.

Relation: Also called "slot" or "property." They are defined by properties and represent possible associations between two or several concepts of the domain. *Connected-to* and *child-of* are examples of relations. Properties can also be used to describe features of the class like *name* or *age*. A relation is described as a subset of the product of n sets: $F : C_1 \times C_2 \times \cdots \times C_n$.

Function: They are special cases of relations where the n^{th} of the relationship is unique for the $n-1$ preceding. An example of a binary function is *mother-of*. A function is formally described this way: $F : C_1 \times C_2 \times \cdots \times C_{n-1} \mapsto C_n$

Axiom: They are used to model sentences that are always true. Axioms represent knowledge that has to be accepted without proof. They allow to constraint the interpretation.

Instance: Also referred as individual. They are the representation of the elements of the domain. Noy and McGuinness [2001] stated that an ontology with a set of individuals constitutes a knowledge base.

2.2.1.2 Ontology goals

Emphasis on ontologies has begun in the 1990s. Ontologies in computer sciences have been developed mainly for the following reasons [Andersen and Vasilakis, 2007; Noy and McGuinness, 2001; Uschold and Gruninge, 1996]:

- For the reuse of domain knowledge: reusability.
- To share a common understanding in order to facilitate human-human, system-system and human-system communications: interoperability.
- To have explicit domain assumptions: specification.

The reuse of a domain knowledge is time saving because it prevents the construction from scratch of knowledge. It is common that some concepts are shared between different domains. However developing an ontology implies an agreement on the concepts to specify in order to facilitate its sharing and reuse. Once an ontology is developed, it could be reused and extended in another context in relation with the domain definition. So an ontology can be used by several applications.

The exchange or share of information is quite difficult if the sender and the receiver doesn't share the same definition of a concept. For example in AI, when two agents want to communicate they must use the same term for the same concept, otherwise they will not understand each other. This aspect meets the definition of a common vocabulary. This vocabulary must be defined and, in the case of AI, each agent must commit to the common ontology. This is a guarantee of a consistency agreement on the concept to specify, but not completeness with respect to the theory specified by an ontology [Gruber, 1993a].

Explicit assumptions allow to have concrete information to work with. It also facilitates the evolution of the ontology as things are explicitly stated, and can be used to teach the domain to persons unfamiliar with it.

2.2.1.3 Types of ontologies

The aim of this part is to give a state-of-the-art of existing ontology classifications to point out the various types of ontologies.

Object of conceptualisation classification The most common classification is based on the object of conceptualisation, *e.g.*, what types of concepts to formalise [Psyché *et al.*, 2003].

Knowledge representation ontologies [van Heijst *et al.*, 1997]: They are used to formalise a knowledge representation model. For example, Gruber's *Frame* ontology [Gruber, 1993b] can be cited. It gathers representation primitives used in frame-based languages: classes, relations, functions...

Upper ontologies [Mizoguchi, 2003]: Upper ontologies are ontologies gathering high level categories. They result of the philosophers' work when they tried to explain what exist in the world. Example of upper ontologies would be Aristotle's ten categories: substance, quantity, quality, relation, place, time, position, state, action, and affection. Another example is is C.S. Peirse [Sowa, 1995] firstness, secondness, and thirdness. Firstness is what can be defined without assuming any other things, such as human, iron.... Secondness is what can be defined in a certain context: that is defined with respect to a second but regardless of a third like mother, director.... Thirdness gives the context to the things defined in secondness. For instance *school* gives the context for *director*.
More recently, efforts to define a standard upper ontology have been undertaken by an IEEE working group named Standard Upper Ontology Working Group (SUO WG). SUO WG has been launched to develop a standard upper ontology to support computer applications [IEEE P1600.1, 2003]. Several existing ontologies have been evaluated and are in competition to be used as the foundations for a standard upper ontology for the computer science field.

Abstract ontologies: Also called core ontologies, they incorporate general concepts, less abstract than upper ontologies. They can be reused in various domains to specify more specific concepts in each domain. Example of abstract ontologies are mereology [Hovda, 2009].

Task ontologies [Gómez-Pérez, 1999; Mizoguchi, 2003]: They provide a vocabulary for tasks in a system, like scheduling, conception, selection and what is related to problem solving. Those tasks may or may not be of the same domain. They usually provide terms describing how to solve a problem such as generic names (plan, goal...), generic verbs (to select, to plan, to assign...) and generic adjectives.

Domain ontologies: They model the vocabulary and the relationships about concepts within a specific domain and enable the reuse of the ontology within this domain, *e.g.*, medical, electronics, or mechanics [Gómez-Pérez, 1999].

Domain-task ontologies: The domain-task ontologies are reusable in a given domain but not across domains [Gómez-Pérez, 1999]. An example in the education domain is the ontology of learning goals [Inaba *et al.*, 2000], which describes goals of teachers and agents in the collaborative learning field.

Application ontologies: They provide vocabulary for applications in a specified domain [Gómez-Pérez, 1999]. They are less suited for reuse by other applications.
Bylander and Chandrasekaran [1988] stated that "Representing knowledge for the purpose of solving some problem is strongly affected by the nature of the problem and the inference strategy to be applied to the problem." A pessimistic interpretation of this quotation could say that it is impossible to write a domain ontology which could be reused across many applications because each application has specific tasks and methods. So Borst [1997] argued that although it is not possible/desirable to write a domain ontology for all tasks in a domain, one ontology can be written that can be shared across several applications.

Weight classification Mizoguchi [2003] proposed a classification based on the weight of ontologies.

Light-weight ontologies: These ontologies are formal classification with relationships between concepts. However the focus is not put on the rigorous definition of concepts. The aim of such hierarchies are, for instance, to speed up search engines and therefore are very use dependant.

Heavy-weight ontologies: Heavy-weight ontologies define concepts more precisely by setting advance properties. An example of heavy-weight ontology is typically an upper ontology.

Granularity classification Psyché *et al.* [2003] proposed a classification based on the level of details. A thin granularity embraces ontologies with a rich vocabulary and high level of details. Large granularity ontologies correspond to ontologies that have weak level of detail. For instance, upper ontologies have a large granularity.

Six-characteristic classification Hepp [2007a] proposed a classification regarding six characteristics:

Expressiveness: The expressiveness concerns the formalism used to represent the ontology. The higher the expressiveness is, the more sophisticated the reasoning over the ontology can be. But the ontology will also be harder to produce and to understand for humans.

Size of the relevant community: Ontology properties are different depending on the number of actors concerned. An ontology designed for a large group should be well documented and of limited size. Moreover the consensus process will be different and more complex.

Conceptual dynamics in the domain: This aspect reflects the change in the domain such as new elements, definition changes... The more dynamism there is, the harder the maintenance of the ontology will be.

Number of conceptual elements in the domain: This correspond to the size of the ontology. Smaller ontologies are more popular and quickly adopted than larger ones [Hepp, 2007b].

Degree of subjectivity in a conceptualisation of the respective domain: Here is addressed how subjective concepts in a domain are. For example food or religion domains are more prone to subjective judgement than hard sciences as mathematics or computer sciences. Consensus disagreements are more likely to append as the definition gets more precise.

Average size of the specification per element: This criterion influences the effort needed for achieving consensus and for coding the ontology. The specification can be more or less comprehensive and for different types, *e.g.*, first-order logic axioms, attributes.

This classification take into account several characteristics of an ontology. These characteristics can be visualised on a radar chart in order to point out the strong and weak points of the ontology (see figure 2.2). This classification allows to easily compare ontologies by showing them on the same graph.

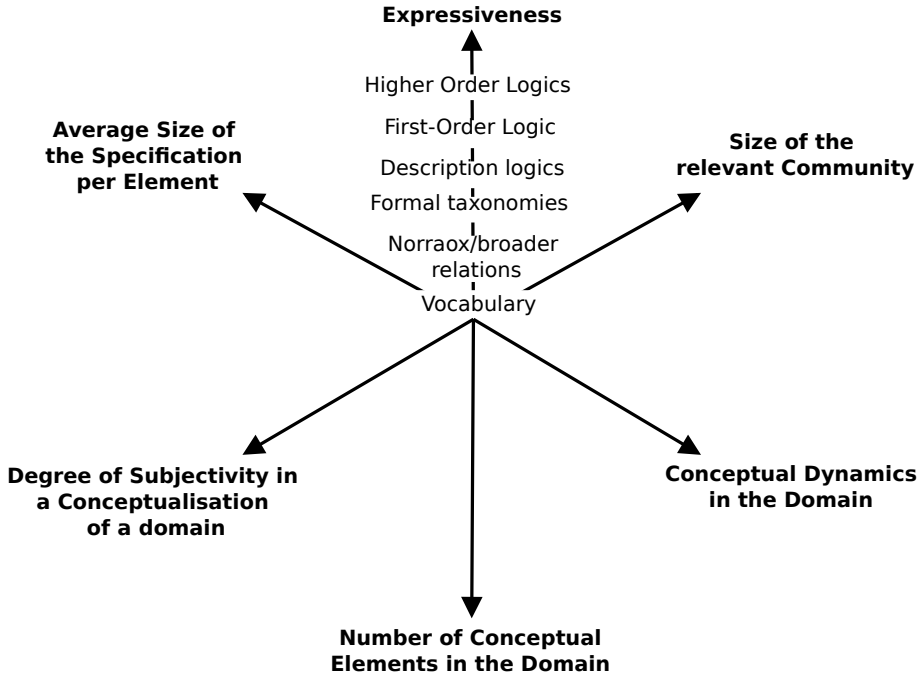


Figure 2.2 – Six-characteristics variables of an ontology [Hepp, 2007a]

2.2.1.4 Summary

Ontologies have been a hot research topic in artificial intelligence. No consensus has been found on the definition of the term "ontology," hence several definitions coexist. During the last years, ontologies started to be used in various domains. They allow to formalise and represent knowledge about a part of the world or of a domain. The core elements of an ontology are the concepts and the possible relations between these concepts. Ontologies are the result of a common agreement.

Major benefits of using ontologies are the capability to reuse knowledge among different applications. They allow to structure, standardise and specify domain assumptions. It is also a communication mean between humans, but also between computational systems. Computational systems can then manipulate the knowledge and even infer implicit facts.

Ontologies can be classified according to various criteria. The following citation summarises well what has to be reminded about the ontology types:

> "*There are several types of ontologies, and each type fulfils a different role in the process of building a domain model.*"
>
> [Studer *et al.*, 1998]

The design of an ontology is a difficult task. For this purpose, several methodologies have been developed.

2.2.2 Ontology engineering methodologies

Ontology engineering can be defined as a set of tasks to build, maintain, reuse ontologies. Figure 2.3 presents the ontology engineering process as defined by Gómez-Pérez *et al.* [2004]. It is decomposed into three main parts. First it has to be checked if it is worth developing the ontology. Then there are several interactions to design the ontology. Finally the ontology can be used and has to be maintained. However most methodologies focus on the design of the ontologies. According to Mizoguchi and Ikeda [1997], the "ultimate" purpose of ontology engineering is "to provide a basis of building models of all things in which computer science is interested."

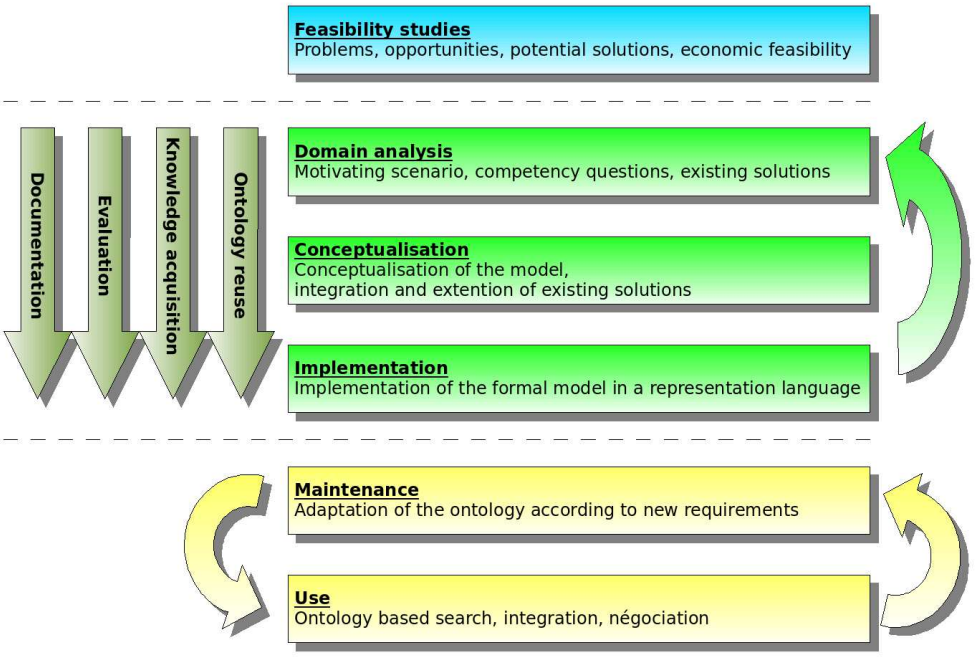


Figure 2.3 – The ontology engineering process presented by Gómez-Pérez *et al.* [2004].

Andersen and Vasilakis [2007] identified three basic steps in ontology engineering, which are nearly the same as those encountered for knowledge management.

1. Knowledge acquisition: extract knowledge from various information sources.
2. Knowledge representation: Structure and formalise the knowledge using an ontology language.
3. Knowledge use and reuse: allow search, navigation, reasoning, visualisation, etc. of the knowledge.

2.2.2.1 General guidelines

In order to guide the design of an ontology, some rules and principles have been introduced by several authors such as Arpìrez *et al.* [1998]; Gómez-Pérez [1999]; Psyché *et al.* [2003].

Clarity and objectivity: The ontology should provide the objective definition as well as the definition in natural language of the defined terms [Gruber, 1993a].

Completeness: A definition expressed by a necessary and sufficient condition is preferred over a partial definition [Gruber, 1993a].

Coherence: In order to allow inferences that are consistent with the definition of the ontology [Gruber, 1993a].

Maximise monotonic extendibility: The addition of new term should not impact existing concepts and definition [Gruber, 1993a].

Minimal ontological commitments: Keep the ontology generic so parties committed to the ontology can extend and specialise it freely.

Ontological distinction principle: Classes of an ontology should be disjoint. The criterion that determines the core properties of a class is called the Identity Criterion [Borgo *et al.*, 1996].

Diversification of hierarchies: The idea is to benefit from multiple inheritance mechanisms. So it will be easier to define new classes and new concepts as they can be specified from existing ones [Arpìrez *et al.*, 1998].

Modularity: Minimise coupling dependencies, between modules [Bernaras *et al.*, 1996].

Minimise the semantic distance between sibling concepts: Close or similar concepts should be defined as subclasses of one class and should be defined using the same primitives [Arpìrez *et al.*, 1998].

Standardise names whenever possible: It is recommended to standardise names in order to facilitate ontology understanding [Arpìrez *et al.*, 1998].

2.2.2.2 Existing methodologies

Psyché *et al.* [2003] listed about 29 ontology engineering methodologies. These methodologies allow to build an ontology from the beginning, from the integration or fusion of other ontologies, from re-engineering, from collaborative construction, and from the evaluation of existing ontologies. In this section are presented the ontologies engineering methodologies that are considered as the most significant according to this work.

One of the first ontology engineering methodology is the one proposed by Uschold and Gruninge [1996]. At this time there was no available methodology. So they proposed to fill this gap with a comprehensive methodology composed of the following steps:

- Identify the purpose and the scope: make clear the reasons why an ontology is being build and its usage.
- Build the ontology.
  - Capture: identify, define and agree on the key concepts and relations.
  - Code: represent explicitly the conceptualisation of the previous phase.
  - Integrate existing ontologies: adapt or reuse concepts of existing ontologies during both previous steps.
- Evaluation: make a judgement about the ontology.
- Documentation: document the ontology in order to facilitate knowledge sharing.
- Guidelines for each phase.
  - Clarity: document and clearly define concepts.
  - Coherence: the ontology should be consistent.
  - Extensibility: anticipate extension by providing conceptual foundations.

Bachimont [2000] presented a design methodology that is composed of three steps, corresponding to the following three commitments.

1. Semantic commitment. Structure the taxonomy of concepts based on four principles:
   - Closeness to its ancestor.

- Different specification against ancestor (otherwise no use to define this concept).
- Closeness with concepts at the same level (brothers concepts)
- Different specifications against brother concepts.

2. Ontological commitment: specify concepts and their extension with formal semantic and other concepts.
3. Computational commitment: select a formalism and define functions for computer systems.

The Ontology Development 101 [Noy and McGuinness, 2001] is a knowledge engineering methodology designed to create ontologies. It is an iterative methodology, *i.e.*, it starts with a rough first pass on the ontology in order to then refine it. This methodology is also independent from the representation language used. They have based their ontology on three fundamental rules for ontology design:

1. There is not only one way to model a domain.
2. Ontology development is necessarily an iterative process.
3. Concepts should be close to objects and relations present in the domain.

The methodology is composed of seven steps that will be repeated:

1. Determine the domain and the scope of the ontology.
2. Consider reusing existing ontologies.
3. Enumerate important terms.
4. Define the classes and the class hierarchy.
5. Define the relations between classes.
6. Define the restrictions on the slots.
7. Create instances.

Denker [2003] proposed a method to build ontologies using the DAML+OIL representation language and Protégé as the development tool. This method is quite simple. It is based on three main steps: create a new ontology by defining the classes, properties, instances, restrictions and boolean combinations, then load existing ontologies to reuse them, and save the resulting ontology.

METHONTOLOGY is a framework proposed by Fernandez *et al.* [1997]. It includes the identification of the ontology development process, a prototype-based life cycle for the ontology and the methodology itself. The identification phase describes the tasks to build the ontology. The life cycle based on a prototype allows to identify the different steps an ontology goes by. The methodology specifies the steps for the activities, the techniques used, and the evaluation method.

Another methodology has been proposed by Grüninger and Fox [1995]. It resulted from the experience in the development of the TOVE[1] project's ontology [Grüninger and Fox, 1994]. This methodology is composed of six steps:

1. Capture of motivating scenarios: the development of the ontology is motivated by problems or example.
2. Formulation of informal competency questions: question representing requirements about the scenarios.
3. Specification of the terminology of the ontology within a formal language: extract terms from the questions and use them as the basis for the ontology.
4. Formulation of formal competency questions using the terminology of the ontology: formalise the questions.
5. Specify axioms in First-Order logic: specify and constraint the definition of terms.
6. Establish conditions for characterising the completeness of the ontology: define the conditions under which the solutions to the questions are complete.

Bernaras *et al.* [1996] developed an approach to design ontologies during the KACTUS project, which aimed at investigating how ontologies could support knowledge reuse in large technical systems. Their methodology is based on three steps:

[1] TOronto Virtual Enterprise

1. Specification of the application.
2. Preliminary design of the ontology from the specifications.
3. Ontology refinement.

The SENSUS methodology [Swartout *et al.*, 1997] is a bit different from other methodologies. SENSUS is an existing ontology composed of 50,000 high and medium level concepts organised as a hierarchy. This ontology is used to create domain ontologies by:

1. Selection of terms from the domain. They will be the "seeds."
2. The seeds terms are linked to the SENSUS ontology.
3. All concepts from the seeds to the root of the SENSUS hierarchy are included in the new ontology.
4. Then the terms that could be relevant to the domain are added.
5. Finally terms are manually added according to the sub-trees of currently selected nodes.

After these steps, an extract of the SENSUS ontology is available that will be the new domain ontology.

2.2.2.3 Methodologies comparison

Hakkarainen *et al.* [2005] made a comparison of ontology design methodologies between the Ontology Development 101, Denker [2003] method and a method proposed by Knublauch *et al.* [2003] in a tutorial for the 2nd International Semantic Web Conferences. The tutorial from Knublauch *et al.* [2003] addresses the creation of ontologies with Protégé and presents briefly a methodology composed of seven steps that are: determine the scope, consider reuse, enumerate terms, define classes, define properties, create instances, classify ontology. Two evaluation methods were used: a quality framework proposed by Su and Ilebrekke [2002] based on six criteria and an application used to valuate each criterion. In both steps the Ontology Development 101 arrived first. Furthermore it is the only one of the three evaluated ontologies that is language independent.

Fernández-López and Gómez-Pérez [2002] analysed some existing methodologies and contrasted them with IEEE software development standards. This review covers the following methodologies: the approach proposed by Uschold and Gruninge [1996], methodology from Grüninger and Fox [1995], the KACTUS methodology [Bernaras *et al.*, 1996], METHONTOLOGY [Fernandez *et al.*, 1997] and the SENSUS methodology [Swartout *et al.*, 1997]. The conclusion they achieved is that the METHONTOLOGY is the most mature of them as it the most complete according to their criteria and it has been used to build ontologies and applications.

To be able to represent and use an ontology, engineers need to selected a suitable representation language.

2.2.3 Ontology representation languages

The representation of an ontology is a key factor for the adoption and the use of the ontology. Ontologies can be expressed using natural language. However this is not suitable for computers to understand and to perform reasoning on an ontology. For this purpose, specific languages have been developed to store and to manipulate the ontology in a formal way.

As ontologies are used by computers, semi-formal languages and formal representation languages will be presented. A semi-formal language has a vocabulary expressed in an explicitly defined artificial language. Formal languages are expressed with an artificial language defined with a formal semantic, proofs and theorems such as soundness and completeness [Uschold and Gruninge, 1996].

There are many ontology representation languages available. Many conceptual modellings exist and are used as a basis for the representation languages. Thus representation languages can be categorised.

Frame-based languages Frame-based languages are composed of frames. A frame is "a data-structure for representing a stereotyped situation, like being in a certain kind of living room, or going to a child's birthday party. Attached to each frame are several kinds of information" [Minsky, 1974]. The frames can contains properties called "slots" or "attributes." Ontology representation languages based on the Frame concept are for example F-Logic, Ontology Inference Layer or Knowledge Machine. F-logic [Kifer *et al.*, 1995], also called Frame-Logic, consists in defining schema facts (classes, inheritance, relations...) such as `man::person` or `person[hasFather=»man]`, facts (instances...) like `Thomas:man[hasBrother->Michael]`, rules like `FORALL X,Y X[hasSon->Y] <- Y:man[hasHasFather->X]`, and finally queries, such as `FORALL X,Y <- X:woman[hasSon->Y[hasFather->Thomas]]`.

First-Order Logic (FOL) based languages [Smullyan, 1995] . An ontology described using the FOL is composed of declarative propositions, predicates and quantifiers. Knowledge Interchange Format (KIF) and CycL are two languages based on FOL. KIF is a formal language designed for the exchange of knowledge between disparate computer software. Each computer can have its own internal representation and resort to KIF just for the information exchange. This language was defined within the Ontolingua project [Farquhar *et al.*, 1997; Gruber, 1993b]. Its syntax is based on LISP and its semantic on the FOL. The CycL is a language that has been developed in the scope of the Cyc project [Lenat *et al.*, 1990]. The aim of this project was to specify a large common-sense ontology for Artificial Intelligence. CycL is the language used to represent the Cyc knowledge. Like KIF, CycL is based on FOL and LISP.

Semantic networks allow to represent knowledge as a graph that is the structure of meaning [Lehmann, 1992]. The nodes represent conceptual units and the directed vertices represent the relations between the units. The relations can be of any type, such as *is-a*, *composed-of*, or *parent-of* relations. Each node and vertex has an attached meaning that allows to derive implicit knowledge.

The Description Logics (DL) are another language to represent ontologies. At the contrary to semantic networks, DLs are a knowledge representation language with a logic-based semantic. The following sentence "A man that is married to a doctor and has at least 5 children, all of whom are professors" can be represented with Description Logics: $Human \cap (\neg Female) \cap (\exists married.Doctor) \cap (\geq 5\_has\_Child) \cap (\forall hasChild.Professor)$ [Baader *et al.*, 2007]. Reasoning as a central service proposed by DLs [Baader and Nutt, 2002]. DL-based languages should come with reasoning procedures that are sound and complete with respect to specified semantics. That means that all the sentences that can be inferred from the ontology are true (soundness), and given an ontology, if there is a sentence that is true, then such a sentence can be inferred (completeness) [Lenzerini *et al.*, Year unknown]. The Web Ontology Language is an example for an ontology representation language based on DL.

The Web Ontology Language takes also place within the Semantic Web representation languages. This representation language category is detailed in the following section because it represents an important and recent evolution in the ontology representation domain.

2.3 Semantic Web

2.3.1 Definition

The World World Web (WWW) [Berners-Lee, 1989] has become a huge network nowadays. This network works on the Internet and is composed of pages accessible through a Web browser. The pages are connected together *via* hypertext links. Hypertext links allow to link anything to anything on the Web. The power of the WWW comes from its universality.

Today most of the content of the WWW is designed to be read by humans. The Web pages are defined *via* the Hypertext Markup Language (HTML), which provides rendering information, such as the structure of a page. Theses data for textual, structural, and graphical information is understood and processed by computers, but intended for Human consumption [Baader *et al.*,

2003]. The content of the Web pages is not designed for computer programs, they have no reliable mean to process the semantics of the pages [Lee *et al.*, 2001].

The Semantic Web (SW) [Lee *et al.*, 2001] is an extension of the current Web in which the information is given in a machine-understandable and well-defined meaning. It aims at creating a structure for the meaningful content of Web pages, such that information can be shared and processed by automated tools as well as by human users. These automatic tools are often referred to as agents or Web agents. To provide these agents with understandable information, the content of the page has to be annotated with metadata in a standardised and expressive language. To make sure that different agents have a common understanding on the annotation terms, they have to commit to the same ontology in which the terms are described. For this purpose several representation languages have been developed.

2.3.2 Semantic web representation languages

Semantic Web representation languages are developed to allow computer systems to process meaningful information. They present the information in a structured form. SW languages form a hierarchy composed of various layers. Languages located in one layer take advantage of the capabilities of the layers below. Figure 2.4 illustrates the hierarchy. The different layers with corresponding SW languages can be seen. For instances the Resource Description Framework (RDF) for data interchange and OWL for the ontology representation. As it will be seen later, OWL is based on RDF-Schema (RDFS). The layers Rule, Unifying Logic, Proof, and Trust do not have stabilised standard definition yet. However the figure presents the latest languages and not other SW languages that were superseded.

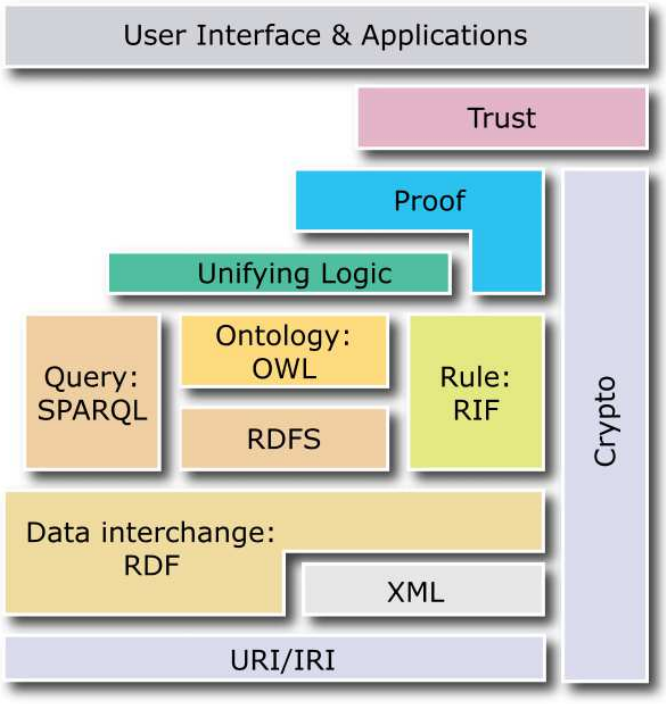


Figure 2.4 – W3C Semantic Web stack [1].

Figure 2.5 gives an overview of the main semantic web representation languages. It shows the date of first publication of each of them and it gives an overview on the evolution of the languages in the SW domain.

Most recent SW representation languages are based on XML[1] or have an XML representation.

[1] `http://www.w3.org/2007/03/layerCake.png`
[1] `http://www.w3.org/TR/REC-xml/`

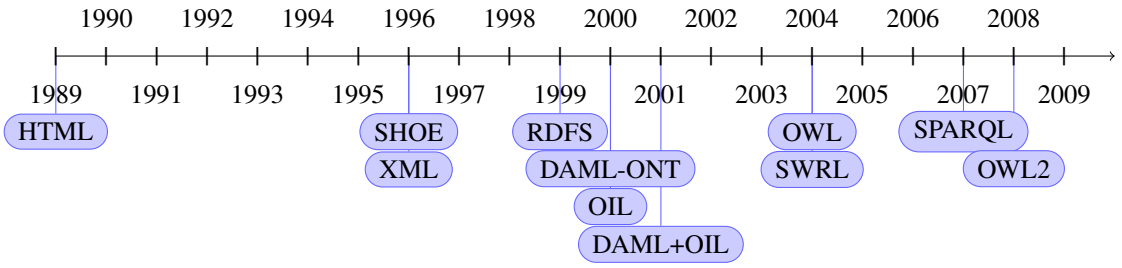


Figure 2.5 – Main semantic web languages with the corresponding year of first publication.

XML, which stands for eXtensible Markup Language, allows to structure a document with "elements." An element is delimited by tags. A tag begins with "<" and ends with ">." XML provides a syntax for hierarchically structured documents to facilitate the sharing of documents across different systems. It is an extensible language as it has no semantic constraints and everyone can define his own tags. The structure of XML document can be constraint to match a given scheme. For this purpose a W3C recommendation called XML Schema[1] was published. It defines how the XML document has to be structured. An example of XML document and schema is available in appendix A.

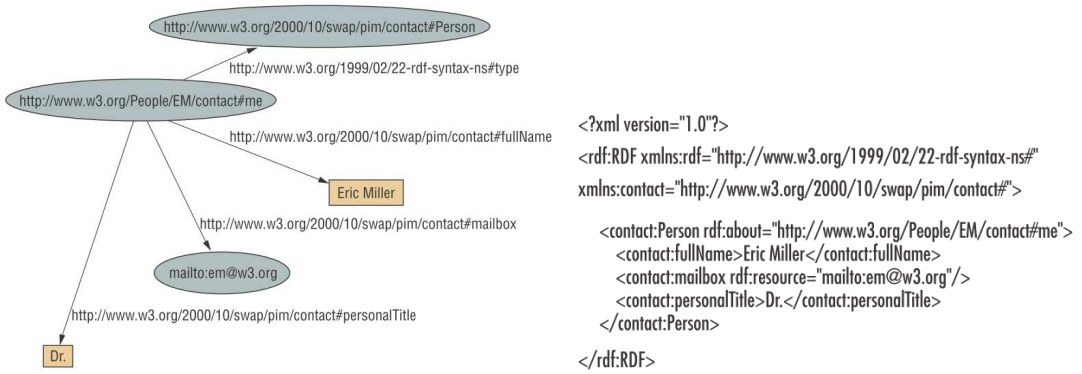
However XML and XML schema are not sufficient to represent ontologies as they provide users with no semantic meaning of the arbitrary structure. In fact they are tools that are used as foundations for dedicated models such as the Resources Description Framework (RDF[2]). RDF is a Frame-based based model for data interchange on the Web. This language has been originally designed for describing Web pages and their resources. RDF has become a W3C recommendation in 1999. It incorporates the meaning by defining statements. Each statement is called a "triple." Triples are composed of a subject, a predicate or verb, and an object. RDF encodes each information as an Uniform Resource Identifier (URI) in order to ensure that concepts have a unique definition. The whole set of triples creates an oriented and labelled graph. RDF proposes an XML-based syntax called RDF/XML. Figure 2.6 (a) illustrates a RDF graph representing a person while figure 2.6 (b) shows its representation using RDF/XML. On both figures, one can see the URIs used as unique identifiers. RDF names the relationship between things with an URI. Like for XML, it is possible to structure RDF documents with RDF schema (RDFS). RDFS provides a richer representation with basic ontological primitives like `rdfs:Class` or `rdfs:subClassOf` in order to define a class hierarchy. It defines various concepts like classes, subclasses, domain and range or properties, etc.

In 2000 the Defense Advanced Research Project Agency (DARPA) started a project to develop a language and tools to facilitate the design of the Semantic Web. The program was called DARPA Agent Markup Language (DAML). RDFS was a good starting point but has limited expressiveness as it provides no cardinality, no exclusions between classes, etc. The aim of the project was to provide a language to express more sophisticated class definitions and properties than RDFS. It resulted in the development of the DAML-ONT language. For instance the DAML-ONT allows to define equivalent properties or subproperties. However DAML-ONT suffers from an inadequate semantic specification like RDFS [Horrocks *et al.*, 2003].

The Ontology Inference Layer (OIL) is another ontology representation language [Fensel *et al.*, 2001]. It was developed approximately at the same time as DAML-ONT (mainly by European researchers) with the same objectives as the DAML program. This language has been defined to meet three requirements:

- It must be highly intuitive.
- It must have a well defined formal semantic with reasoning properties that ensues the completeness, correctness and efficiency.

[1] `http://www.w3.org/XML/Schema`
[2] `http://www.w3.org/RDF/`



(a) RDF graph (b) RDF/XML representation

Figure 2.6 – RDF graph and its RDF/XML description representing Eric Miller [Shadbolt *et al.*, 2006].

- It must ensure interoperability by reusing existing Web languages.

OIL incorporates the essential modelling primitives of Frames, which is the notion of concepts organised in a hierarchy, and attributes of the concepts. This allows OIL to be more intuitive as the Frame concepts are similar to the object-oriented programming paradigm. OIL is also based on the description logics in order to integrate formal semantic and to allow efficient reasoning. Finally OIL is based on RDFS. This fulfils the third requirement, which is to reuse existing Web languages. It was the first ontology language that combines elements from DL, RDFS, and Frames.

Quickly DAML and OIL working groups joined their efforts. It resulted in the merge of the two languages and the creation of the DAML+OIL language [Connolly *et al.*, 18 December 2001]. The frame structure representation was discarded and replaced by DL-style axioms, which are easier to represent with RDF syntax. The integration into RDFS was also tightened. DAML+OIL has been used as a basis for the development of the Web Ontology Language.

The W3C defined its own ontology language called the Web Ontology Language (OWL) [Smith *et al.*, 2004].

Some other ontology representation languages are available like XOL, SHOE or topic maps. No further details on these languages will be provided as they have been superseded by newer and more powerful languages like OWL.

2.3.3 Web Ontology Language

2.3.3.1 General description

The Web Ontology Language (OWL) [Smith *et al.*, 2004] is a description language for representing ontologies. Its aim is to facilitate the processing of the information content by computer systems instead of just presenting the information. It is a W3C recommendation, *i.e.*, a standard, for the Semantic Web since 2004. OWL is a revision of the DAML+OIL language with the aim of incorporating the lessons learned by the development, and the experiences with DAML+OIL. Nowadays OWL represent the prevailing standard for ontology design [Andersen and Vasilakis, 2007]. In the scope of the Semantic Web, OWL has been designed to:

- provide an exact meaning to Web information.
- allow information from the web to be integrated by computers.
- allow computers to process Web information.

Like DAML+OIL, OWL is an advanced and sophisticated language. Its development has been influenced by Description Logics, the frames paradigm, XML and RDF. The semantic in OWL is formalised by means of DL style model theory [Lenzerini *et al.*, Year unknown]. Hence OWL has more facilities to express the meaning and the semantics than XML, RDF, and RDF-S.

2.3.3.2 OWL and RDFS comparison

OWL is an extension of RDFS and provides a richer vocabulary for describing properties and classes. By using the RDFS/XML representation OWL facilitates the exchanges between information systems.

RDFS describes properties and classes of RDF resources. It provides language constructors such as *rdfs:domain,rdfs:range* and *rdfs:subClassOf*. Horrocks *et al.* [2003] gave a short example of a RDFS definition:

- declare classes like `Country`, `Person`, `Student` and `Canadian`.
- state that `Student` is a subclass of `Person`.
- state that `Canada` and `England` are both instances of the class `Country`.
- declare `Nationality` as a property relating the classes `Person` (its domain) and `Country` (its range).
- state that `age` is a property, with `Person` as its domain and integer as its range.
- state that `Peter` is an instance of the class `Canadian`, and that his `age` has value 48.

Compared with RDFS, OWL is more expressive as it contains new features such as the class definition, property types and also new concepts like equivalent classes and properties, equality, difference, opposite, symmetry of two resources. OWL reuses some constructors from RDFS such as `rdfs:domain`, `rdfs:range` and `rdfs:subClassOf`, and add new vocabulary for:

- logical relations: union, intersection, complement and disjunction.
- cardinality.
- richer properties typing.
- property characteristics (like symmetry and transitivity).
- enumerations.

OWL enables a better description of classes and properties by adding constraints on properties, cardinality, enumerations, equivalence, union, etc. The RDF schema is not sufficient if deduction from computers is expected. Horrocks *et al.* [2003] presented an OWL extension of the above description made with RDFS, wherein the enriched semantic compared to RDFS can be seen:

- state that `Country` and `Person` are disjoint classes.
- state that `Canada` and `England` are distinct individuals.
- declare `HasCitizen` as the inverse property of `Nationality`.
- state that the class `Stateless` is defined precisely as those members of the class `Person` that have no values for the property `Nationality`.
- state that the class `MultipleNationals` is defined precisely as those members of the class `Person` that have at least 2 values for the property `Nationality`.
- state that the class `Canadian` is defined precisely as those members of the class `Person` that have `Canada` as a value of the property `Nationality`.
- state that `age` is a functional property.

2.3.3.3 Reasoning

The OWL language is designed to support reasoning on the ontology.

The reasoning allows to to classify the ontology's concepts and instances. The classification is based on the subsumption. A concept C subsumes a concept D ($C \subseteq D$) when C contains all individuals of D. If $C \subseteq D$ and $D \subseteq C$ both concepts are equivalent. For example

- A bus driver is a person that drives a bus.
- A bus is a vehicle.
- A driver drives a vehicle.

From this definition it can be inferred that a bus driver drives a vehicle, so he must be a driver.

Instances are also subject to classification. Given the property *hasPet* that has the concept `Human` as domain and `Pet` as range. If "Donald is the pet of Thomas," it can be stated that: Thomas has pet Donald, hence Thomas must be a Human, and Donald must be a Pet.

One objective of the reasoning is to check the consistency (satisfiability) of the ontology. To be consistent an ontology should have no contradiction in its definition. To illustrate an inconsistent concept, the following definitions can be used:

- Vegetarians do not eat animals.
- Cows are vegetarians.
- A mad cow is a cow.
- A mad cow eats sheeps' brain.
- Sheeps are animals.

When checking the consistency of this definition, it can be stated that mad cows eat animals. Or mad cows are cows and thus are vegetarian, thus they can not eat animals. So the definition of mad cow is inconsistent with the definition of vegetarian.

Researchers are now focusing on the Web Ontology Language. Today OWL is considered as the most suitable language for the Semantic Web and for the representation of ontologies thanks to its expressiveness and reasoning capabilities. It is achieved because OWL is based on Description Logics.

2.3.3.4 Description logic expressiveness

The Description Logics are a family of logic-based representation formalisms for knowledge. They were designed as an extension to Frames [Gruber, 1993b]. They are also subsets of the First Order Logic [Baader and Nutt, 2002].

Description Logic has sound (yield correct results), complete (yield all correct results) and decidable (yield results in finite time) inference procedure. Contrary to databases, Description Logic envelopes the *Open World Assumption* as the domain can be infinite. In the Open World Assumption a lack of information does not mean a negative information like in databases. That means that when an information is not specified as true, it can not be stated that it is false. The opposite is the Closed World Assumption, in which missing or not know as true information is considered as false.

The different Description Logics are named according to their expressiveness by a succession of letters, for instance $\mathcal{SHOIN}$. Each letter has its own signification.

$\mathcal{AL}$: It means Attribute Language which can be considered as the base of other DL languages. It allows the following syntax rules:

- D, C are concepts
- S, R are properties
- Atomic concept (A)
- Universal concept ($\top$)
- Bottom concept ($\bot$)
- Atomic negation ($\neg A$)
- Intersection ($C \sqcap D$)
- Value restriction ($\forall R.C$)
- Limited existential quantification ($\exists R.\top$)

$\mathcal{FL}^-$: It is a part of AL in which the atomic negation has been removed.

$\mathcal{FL}_0$: Is is a part of FL^- in which the existential quantification has been removed.

$\mathcal{C}$: Complex concept negation ($\neg C$, not only atomic like in AL).

$\mathcal{S}$: Gather AL and C and add role transitivity ($Trans(R)$).

$\mathcal{H}$: Role hierarchy, subsumption ($R \subseteq S$).

$\mathcal{R}$: stands for complex role inclusions ($R \circ S \subseteq R, R \circ S \subseteq S$). It allows to express the fact that each car contains an engine $Car \sqsubseteq \exists hasPart.Engine$, an owner of a car is also an owner of an engine, *i.e.*, the following subsumption is implied $\exists owns.Car \sqsubseteq \exists owns.Engine$.

$\mathcal{Q}$: stands for qualifying number restrictions ($\leqslant n\ R.C$, $\geqslant n\ R.C, = n\ R.C$), *i.e.*, at least, at most. For instance: Car $v = 4\ hasComponent.Wheel$.

$\mathcal{O}$: Nominals, enumerations ($\{a\}$ or $\{a_1, \ldots, a_n\}$, *one-of* constructor).

$\mathcal{I}$: Inverse property (R^-).

$\mathcal{N}$: Cardinality restrictions ($\leqslant n\ R$, $\geqslant n\ R$, $= n\ R$).

$\mathcal{F}$: Functional properties ($\leqslant 1\ R$).

$\mathcal{E}$: Full existential qualification.

$\mathcal{U}$: Concept union.

(D): Use of datatype properties, data values or data types (concrete domain).

Different version of OWL are existing. Each of them provides a different level of expressiveness and corresponding reasoning capabilities.

2.3.3.5 OWL versions

The first version of OWL has been released as a standard in 2004. OWL kept on evolving according to users needs, tools and algorithm evolution, and new findings. Two major versions have been identified, OWL 1 and OWL 2 that are standards, and an intermediate version called OWL 1.1.

OWL 1 The first version of OWL became a standard in 2004. This first version is now referred to as OWL 1. It is composed of three increasingly expressive sublanguages OWL Lite, OWL DL, and OWL Full [Bechhofer *et al.*, 2004]. OWL Lite and OWL DL express a small subset of the First Order Logic and assure the decidability of inferences whereas OWL Full provides a greater expressive power and compatibility with RDFS.

OWL Lite has a very restricted expressive power but is easy to implement. It is designed for users needing a classification hierarchy and simple constraints. Examples of restrictions are the limitation to 0 or 1 of cardinality values, connectors and enumerations are forbidden. OWL Lite has an expressive tie similar to $\mathcal{SHIF}(D)$ Description Logic and has a reasoning time in `ExpTime`.

OWL DL is named in correspondence to *Description Logics*. It extends OWL Lite. It enables the highest degree of expressiveness while retaining computational completeness (*i.e.*, the computation of all conclusions is guaranteed) and decidability (*i.e.*, all conclusions can be computed in a finite time). However this language has restrictions such as a class can be a subclass but not an individual of another class. OWL DL is partly based on $\mathcal{SHOIN}(D)$ Description Logic, which is more expressive than $\mathcal{SHIF}(D)$ but more difficult to reason with. So the time complexity for reasoning is `NExpTime`. Appendix B presents the syntax of OWL DL.

OWL Full gives the maximum degree of expressiveness and freedom but without guarantees for computational results. It contains OWL DL but provide a higher expressiveness making reasoning undecidable. It is a real subset of RDFS where `owl:Class` is equivalent to `rdfs:Class` and `owl:Thing` is the equivalent to `rdfs:Resource`. In OWL Full a class can be treated simultaneously as a collection of individuals and as an individual in its own right. The table 2.1 summarises the different sublanguages with the corresponding DL and complexity.

Table 2.1 – OWL 1 sublanguages with DL equivalence and complexity.

Sublanguage	Description Logic	Complexity
OWL Lite	$\mathcal{SHIF}(D)$ [Horrocks2003]	`ExpTime` [Tobies2001]
OWL DL	$\mathcal{SHOIN}(D)$ [Horrocks2003]	`NExpTime` [Tobies2001]
OWL Full	no DL equivalence	not decidable, covers RDF-Schema [Horrocks2003]

OWL 1.1 OWL 1.1 [Grau *et al.*, 2006; Patel-Schneider and Horrocks, 2006] has been submitted on December 19th, 2006 and has not become a W3C recommendation. OWL 1.1 extends OWL 1 with a small but useful set of features that have been requested by users. These are features like additional property and qualified cardinality constructors, extended datatype support, simple metamodelling, extended annotations, and extra syntactic sugar, which make some idioms easier to write. OWL 1.1 corresponds to the $\mathcal{SROIQ}(D)$ logic by adding the $\mathcal{R}$ and $\mathcal{Q}$ expressiveness to $\mathcal{SHOIN}(D)$ [Horrocks *et al.*, 2006]. There is also other new points such as disjoint roles or it allows the expression of "local reflexivity" for example to define the concept of "narcist" as $\exists likes.Self$.

OWL 2 OWL 2 became a W3C standard in October 2009 [OWL 2, 2009]. Like OWL 1.1, OWL 2 corresponds to the $\mathcal{SROIQ}(D)$ logic and is referred to as OWL 2 DL. RDF graphs that are considered as OWL 2 ontology are referred to as OWL 2 Full ontologies.

OWL 2 is composed of three profiles (sublanguages) that are OWL 2 EL, OWL 2 QL and OWL 2 RL, which offer important advantages in particular scenarios [Motik *et al.*, 2009a]. Each of three main profiles from OWL 2 are more restrictive than OWL DL but provide better reasoning capabilities. In addition OWL 2 provides new representation syntaxes like the Manchester syntax, which is easier to read, or the OWL/XML that is easier to process with XML tools. The specifications of OWL 2 allow to classify OWL 1 Lite and OWL 1 DL as OWL 2 profiles too.

OWL 2 EL is suitable for ontologies with a very large number of classes and/or properties and where expressive power can be traded for performances guarantee. It provides polynomial time decision to check consistency, class expression subsumption and instance checking. OWL 2 QL is particularly suitable for relatively lightweight ontologies with a large number of individuals, in which data queries are useful or necessary. It provides answers to conjunctive queries in `LogSpace` by using standard relational database technology. OWL 2 RL enables the implementation of polynomial time reasoning algorithms by using rule-extended databases technologies. This profile aims at applications that require scalable reasoning without sacrificing too much expressive power.

The figure 2.7 presents the evolution of the reasoning complexity of ontology representation languages. One can clearly see on this graph that the current trend is to reduce the complexity of the languages, even if it results in a reduction of the expressiveness of the language.

2.3.3.6 Semantic Web Rule Language

The Semantic Web Rule Language (SWRL) is the result of the combination of OWL DL and a subset of the Rule Markup Language [1] ,*i.e.*, Unary/Binary Datalog [Horrocks *et al.*, 2004; O'Connor *et al.*, 2005]. SWRL allows users to write Horn-like rules expressed in terms of OWL concepts to reason about OWL individuals. The rules can be used to infer new knowledge from existing OWL knowledge bases. This allows to fulfil the deductive reasoning lack of OWL [O'Connor *et al.*, 2005]. SWRL rules are saved as part of the ontology. In the W3C Semantic Web stack presented in figure 2.4 rules are located at the same level as OWL. SWRL is not a W3C standard but has been submitted in May 2004. However it has been adopted by many research works and is implemented in several reasoners.

SWRL rules are composed of an antecedent-consequent pair. The SWRL terminology defines the antecedent as the "body" of the rule and the consequent as the "head." The hand and body

[1] `http://www.ruleml.org/`

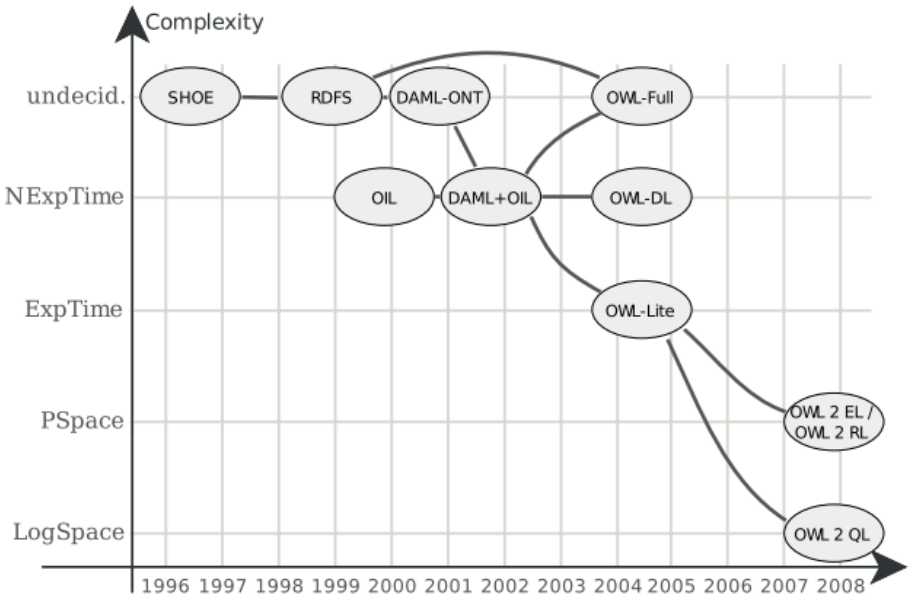


Figure 2.7 – History and complexity of ontology representation languages [Nordmann, 2009].

are composed of a conjunction of one or more atoms. An example of a SWRL rule would be the following:

`hasParent(?x1,?x2)` $\wedge$ `hasBrother(?x2,?x3)` $\Rightarrow$ `hasUncle(?x1,?x3)`

This would result in the creation of a new relation *hasUncle* between two individuals present in the ontology if the first one has a parent who has the second one as brother.

SWRL rules provides more expressibility power to OWL, but at the price of decidability and practical implementations [Parsia *et al.*, 2005]. They work on individuals present in the ontology and allow to deduce new knowledge.

It has to be noted that SWRL is monotonic and thus does not support negated atoms. Hence the following rule is not possible because it would be invalidated if the person gets later a car [O'Connor, 2009].

`Person(?p)` $\wedge$ `not hasCar(?p,?c)` $\Rightarrow$ `CarlessPerson(?p)`

2.4 Ontologies in the product design field

Ontologies have become more and more popular. They are now used in many application fields, such as medical [Dieng-Kuntz *et al.*, 2006], bioinformatic [Stevens *et al.*, 2002] or knowledge engineering [Maedche *et al.*, 2003]. The engineering and design domains have also interest in ontologies.

Cebrian-Tarrason and Vidal [2008] used an ontology to infer knowledge in product conceptual design. Angele *et al.* [2007] presented an application in which an ontology is used to represent the terminology and the complex dependencies between various car parts. The ontology is also used as a mediator between data from different sources. The result is a software assistant that helps engineers in the task of configuring test cars. Rules are used to validate some design choices. For instance, they have designed a rule representing restrictions like "the engine power must not exceed the power of the brakes" or "the devices connected to the battery must match the

amperage of the used battery." Ontologies are also used to enhance collaboration and resolve conflicts. Lima Dutra *et al.* [2010] provided an ontology-based architecture for collaborative design, in which ontologies are used to detect and solve conflicts during the design.

Ontologies can also be used as a standardised representation between tools [Uschold and Gruninge, 1996]. Standards are defined to be a shared understanding of a domain in order to enhance interoperability. Within CAD domain there are several existing standards. One of the most famous and most used is the ISO 10303, which is an international standard for the product data exchange. This standard is also referred as the Standard for the Exchange of Product model data (STEP) [ISO 10303, 1994]. Its aim is to allow a computer-interpretable system independent description of product data throughout its life cycle in order to allow the exchange of product data among different systems. This standard can be used to exchange data between several types of systems such as CAD, PDM, CAE or CAM systems. The development of STEP started in 1984. However the initial release of STEP as an international standard was ten years later, in 1994. STEP has been design to address several domains that are mechanical and electrical design, geometric dimensioning and tolerancing, analysis and manufacturing. It also include specific information related to various industries applications such as automotive, aerospace, ship, process plant, building construction and others. It also addresses the complete product life cycle, from the design to the maintenance. It is organised in parts that are:

- Integrated resources
- Application protocols
- Abstract test suites
- Implementation methods
- Conformance tests

One of the most important elements of STEP is the part 42 that addresses the explicit representation of models. The data modelling language used by STEP is called EXPRESS, and is part of the standard itself (ISO 10303-11). It also provides a graphical representation called EXPRESS-G.

Some persons consider STEP as an ontology. For example, Andersen and Vasilakis [2007] stated that STEP can be considered as an ontology for the representation of product model information. However STEP does not provide reasoning capabilities. EXPRESS includes the common ideas present in ontology design, such as classes and instances by using entities, multiple inheritance, properties, etc. Andersen and Vasilakis [2007] proposed a CAD model information ontology based on part 42 of STEP. This domain ontology formalises various aspects of digital shapes and boundary representation of the geometry.

Ferreira da Silva [2007] defined and developed a tool to convert EXPRESS representations to OWL. OWL is used here to find semantic correspondences between heterogeneous resources. These resources are converted to OWL in order to take advantage of the reasoning capabilities.

However STEP has some limitations, even for product data exchange. STEP does not represent the semantic and the design intent during the design of a product. But today, most CAD software are feature based, *i.e.*, the construction history of CAD model is stored. This history is lost when using the STEP data exchange format. Abdul-Ghafour [2009] proposed an ontology-based approach for product data exchange. This approach resorts to an OWL representation that also represents design intents of the designers through the representation of the features used during the design. The proposed ontology is called Common Design Features Ontology.

2.5 Chapter summary

Nowadays computers are used to store and manipulate knowledge. Knowledge has to be represented in a formal way in order to allow computers to understand and to process it. This is the objective of the Semantic Web.

The Semantic Web is an extension of the current Web in which the information is given in a machine-understandable and well-defined meaning. It is composed of several standards and

languages that allow to structure the knowledge. The most recent Semantic Web language is the Web Ontology Language. The Web Ontology Language is a W3C standard designed to represent ontologies in a formal way. It also allows reasoning thanks to Description Logics, on which the Web Ontology Language is based. Reasoning allows the discovery of new knowledge, and the classification and the consistency checking of the ontology.

Ontologies allow to define a domain by defining concepts and relations of this domain. They aim at reusing the knowledge, facilitating communication and interoperability. Hence an ontology is the result of a consensus. There are various types of ontologies, such as upper ontologies that gathers high level concepts or application ontologies that provide a specific definition for an application in a domain. The design of ontologies is a complex task. For this purpose several methodologies have been developed in order to define, to design and to maintain ontologies. Ontologies are nowadays used in numerous domains like medicine, artificial intelligence or product design.

Chapter 3

Decision Support in Collaborative Environments

Contents

3.1 Computer supported concurrent engineering

Concurrent Engineering (CE) has been defined by Pennell *et al.* [1989] as "a systematic approach to the integrated, concurrent design of products and their related processes, including, manufacturing and support. This approach is intended to cause the developers from the very outset to consider all elements of the product life cycle, from conception to disposal, including cost, schedule, quality and user requirements." CE involves many persons or teams, usually from various disciplines, which collaborate to reach a common goal. The particularity of CE is that tasks are processed simultaneously.

The life cycle of industrial products is complex. Usually, it involves many persons with different knowledge and expertises engaged in several activities for several years. Moreover these resources can be located at different places. Collaborative engineering suits well to this schema. The use of CE results in a reduction of the products' time-to-market by reducing development cycles duration. CE has also positive effects on the innovation as well as on product quality [Koufteros *et al.*, 2002]. In the design domain, collaboration is a key to successfully release a product in time and with a better quality [Shen *et al.*, 2008].

To efficiently apply CE, people need to cooperate, *i.e.*, work together, exchange information, etc. Nowadays cooperation at work is supported by computers. The use of computers to support collaborative activities and their coordination is designated by the term Computer Supported Cooperative Work (CSCW). CSCW appeared in the 1990's and has emerged from four main domains [Villemur, 2006]:

- Social sciences: people organisations, group efficiency...
- Artificial intelligence: cognitive sciences, semantic, scheduling...
- Human-computer interaction: development of multi users graphical user interface...
- Networking and distributed computing: data exchange and transfer, remote storage, distributed systems...

Software is playing an important role to facilitate collaboration. Software systems designed to support CE are often referenced as collaborative systems. Ellis *et al.* [1991] defined collaborative systems, also called groupware, as "computer-based systems that support groups of people engaged in a common task (or goal) and that provide an interface to a shared environment." The notions of "common task" and "shared environment" are crucial within his definition. CSCW systems embed tools that allow synchronous and/or asynchronous interactions between people located at the same or different places. These interactions are synthesised by the CSCW matrix, which is presented in table 3.1. Face-to-face interactions are for instance meeting rooms, in which a beamer or a digital wall are available. Asynchronous interactions are continuous tasks, which are supported *via* control version system, project workspace or bulletin board system. Synchronous distributed interaction are video conference, telephone meeting, shared whiteboard, collaborative editing tools... Asynchronous distributed interaction would be blackboards or e-mails. A comprehensive CSCW system should address all of the quadrants presented in the table 3.1.

Concurrent engineering has been the subject of many research activities [Ghodous *et al.*, 2003; Ghodous and Vandorpe, 2000; Sriram, 2002] that have resulted into different platforms embedding several concepts, including data management and data exchange. The following solutions are available for data communication during life cycle and application integration: data exchange and data sharing. In data exchange, each participant builds his model independently, which will then be exchanged thanks to standard formats and communication protocols. A well known collaborative engineering project using the data exchange schema is SHADE (SHAred Dependency

Table 3.1 – CSCW 2x2 matrix.

	Same time	Different time
Same place	Face-to-face interaction	Asynchronous interaction
Different places	Synchronous distributed interaction	Asynchronous distributed interaction

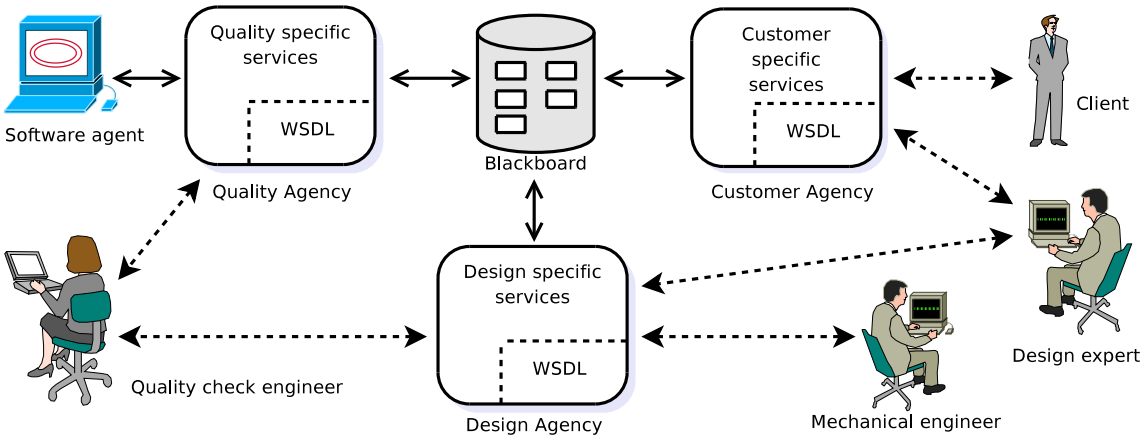


Figure 3.1 – Example of collaborative architecture based on Web Services using a blackboard as central repository [Kuhn *et al.*, 2008].

Engineering) [Olsen *et al.*, 1995; Tenenbaum *et al.*, 1993]. In the case of data sharing, a common repository is used to store and share the design solutions. This repository is divided into several areas and is accessible by to the participants. An example of a project using data sharing is DICE [Sriram, 2002], which was developed at the MIT.

The current trend heads towards data sharing with a central repository as it reduces problems such as data consistency and complexity of the design process [Ghodous *et al.*, 2003]. To allow any system to access these data, a platform independent protocol is needed. The most recent protocols are oriented towards the Web technologies for a better interoperability between heterogeneous systems [Dustdar *et al.*, 2004; Hu *et al.*, 2010; Kuhn *et al.*, 2008; Lima Dutra *et al.*, 2010].

Since the beginning of the decade, Web Services (WS) [Alonso *et al.*, 2004; Web Services Activity, 2002] are more and more used, especially by businesses, thanks to the availability of standards like the Simple Object Access Protocol (SOAP), the Web Services Description Language (WSDL) and the Universal Description Discovery and Integration (UDDI). These standards enable great interoperability as SOAP and WSDL are XML-based formats. W3C defines a Web Service as "a software system designed to support interoperable Machine to Machine interaction over a network." Web Services are especially used in Service-Oriented Architectures (SOA) where they are loosely coupled and reusable. They provide very attractive characteristics for concurrent engineering like their interoperability based on Web standards and their distributed architecture over the network. Figure 3.1 presents an architecture of a Web Services-based collaborative platform in which several stakeholders access the central repository through Web Services.

Concurrent engineering addresses all steps of product development. In the scope of this work, the decision process and how computer can support the collaboration within this process are more deeply studied.

3.2 Decision making process support

Decision support and design rationale are research fields that have been investigated for decades. They aim at supporting decision making activities while keeping the path of the decisions, their argumentation and alternatives through the design process. The most significant frameworks for this work will be presented.

3.2.1 Definition

Decision making is the process of making a choice. The evolution of a product design is composed of a succession of design choices (see figure 3.2). At each step several solutions are possible and the best of them has to be chosen according to given requirements and constraints in order to create a high quality and/or low cost product. This goes by several decisions that have an impact on the final product.

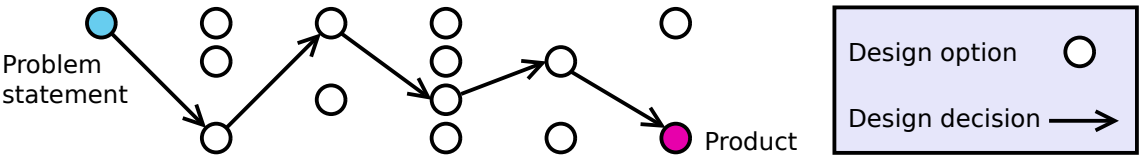


Figure 3.2 – Evolution of the design path according to the design decisions.

A decision making process can be used for many purposes like the choice of the material for a part as well as the selection of the expert that will provide an analysis report. The process of making a decision can be decomposed into several steps. Simon [1977] stated that the decision process is composed of three main stages:

1. The analysis of the problem (gather facts): intelligence stage.
2. The generation of solutions and alternatives: design stage.
3. Make decisions and implement: choice stage.

Bohanec [2001] proposed a more comprehensive list of actions composing the decision making process:

1. Assessing the problem.
2. Collecting and verifying information.
3. Identifying the alternatives.
4. Anticipating the consequences of the decisions.
5. Making the choice using sound and logical judgement based on available information.
6. Informing others of decision and rationale.
7. Evaluating the situation.

Before making the choices, the possible consequences have to be measured. Then, once the decision has been taken, it has to be communicated to concerned peoples and the new situation has to be evaluated.

The decision making process can be enhanced by using information systems. Decision support (DS) aims at helping people in making decisions by using computer-based systems [Bohanec, 2001]. DS overlaps various disciplines like:

Operations research, which aims at finding optimal solutions by exploring the solutions space. It includes methods like combinatorial optimisation, Markov analysis, linear programming, etc.

Decision analysis, which includes decision trees, influence diagrams, etc.

Decisions support systems, which help decision makers to use data and models. It includes expert systems, Online Analytical Processing (OLAP) and other business intelligence tools. Decision support systems are already used for various tasks such as to design processes [Zha *et al.*, 2008] or to plan processes [Chitta *et al.*, 2008].

Data warehouse, are repositories containing multiple heterogeneous data sources. It allows to visualise data from different angles.

Groups decision support, which gather collaborative technologies. In the scope of a collaborative environment, the interest was oriented towards group decision support systems.

3.2.2 Group decision support systems

The decision process may involve several viewpoints and thus lead to conflicts between stakeholders. To solve these conflicts the best solution is negotiation. Several groups decision methods are available [Alexander, 2002].

Nominal group technique consists in the notation of the individual ideas on a papers by each participant. Then the ideas are all gathered on a whiteboard. Once all ideas are present, the discussion can start. At the end each participant votes on each idea.

Delphi technique is like the nominal group technique but uses a survey tool. The participants do not meet. It is an adequate solution for distributed asynchronous decision process.

Arbitration resorts to an outside arbitrator who will select the alternative he deems most appropriate in case no consensus have been found.

Issue-based information system is an argumentation-based framework for problem solving, in which multiple stakeholders can participate.

The Issue-Based Information System (IBIS) [Kunz and Rittel, 1970] is one of the most interesting group decision support systems. IBIS are "meant to support coordination and planning of political decision processes. They guides the identification, structuring, and settling of issues raised by problem-solving groups, and provides information pertinent to the discourse" [Kunz and Rittel, 1970]. The IBIS was defined to find solutions to planning/organisational "wicked" problems. Rittel and Webber [1973] introduces "wicked" problems and specified ten characteristics that describe them:

1. There is no definitive formulation of a wicked problem.
2. Wicked problems have no stopping rules.
3. Solution to wicked problem are not true-false, but good-badly.
4. There is no immediate and no ultimate test of a solution to a wicked problem.
5. Every solution to a wicked problem is a "one-shot operation" because there is no opportunity to learn by trial-and-error, every attempt counts significantly.
6. Wicked problems do not have an enumerable (or an exhaustively describable) set of potential solutions, nor is there a well-described set of permissible operations that may be incorporated into the plan.
7. Every wicked problem is essentially unique.
8. Every wicked problem can be considered to be a symptom of another problem.
9. The existence of a discrepancy representing a wicked problem can be explained in numerous ways. The choice of explanation determines the nature of the problem's resolution.
10. The planner has no right to be wrong.

The search of solutions goes by a discussion between different stakeholders, who bring their expertise and point of view on the resolution of the issues. The IBIS is an argumentation framework. It supports distributed asynchronous decision processes and proposes a structure to the discourse that helps the communication. It also allows to capture the different aspects of the problem from the different participants' points of view [Ebadi *et al.*, 2009]. The structure of the IBIS is composed of three main elements:

"Issue" raises a problem in form of a question. Any element of the IBIS can raise an issue/question.

"Position" is a possible solution or a part of a solution addressing an issue.

"Argument" is a statement that supports or rejects a position.

The result of the argumentation is the choice of one or several positions that will become the solutions to the related problem. Figure 3.3 illustrates the IBIS structure with the presented elements and their connections.

IBIS served as basis for several derivatives such as the Questions, Options and Criteria (QOC) [MacLean *et al.*, 1991]. Arndt [2007] also proposed an extension that is an ontology-based

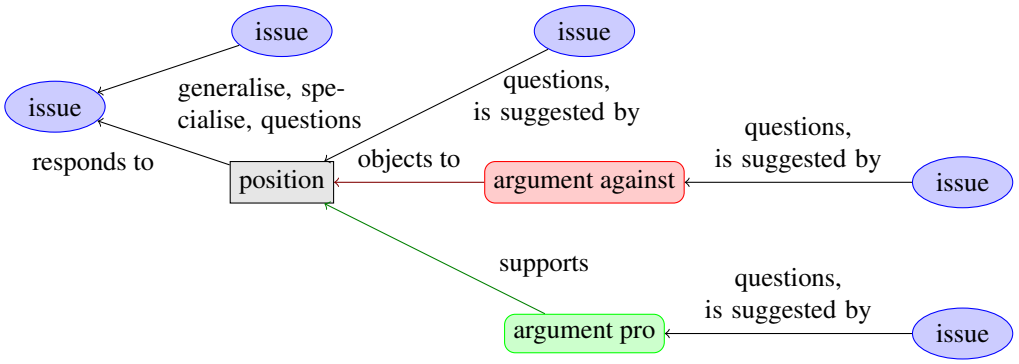


Figure 3.3 – IBIS core structure showing relations between elements.

method for decision support for product development. He proposed a knowledge model that is an extension of the IBIS framework with an ontological representation. Some applications are also presented such as for technology selection or process definition. A graphical version tool of the IBIS was proposed by Conklin and Begeman [1988]. This tool was designed according to three directives:

- provide support to computer-mediated collaborative work with remote participants.
- have a large information base that can be navigated easily.
- explore and capture the design history: decisions, rejections, trade-off analysis, etc. This can be summarised as the rationale behind the design.

3.2.3 Design rationale

The aim of design Rationale (DR) [Moran and Carroll, 1996] is to represent and store the argumentations, trade-offs, alternatives, choices (rejected or not), etc. behind design choices. It provides an explanation of the reasons that have led to a particular design. In this way it is possible to respond to what was thought when a decision has been taken. DR provides documentation on past decision processes and design intents.

3.2.3.1 Frameworks

The first DR framework was the IBIS from Kunz and Rittel [1970]. However they did not mention the "design rationale" term in their contribution. The design rationale aspects of IBIS has been brought forward in a paper from Conklin and Begeman [1988] in which they presented a graphical IBIS representation tool. The natural argumentation process proposed by the IBIS stores many information regarding the decision process and its rationale.

Another example of DR framework is the Distributed and Integrated Collaborative Engineering Design (DICE) [Sriram, 2002]. DICE is a collaborative platform project that includes design rationale aspects with the Design Recommendation-Intent Model (DRIM) [Pena-Mora *et al.*, 1993]. DRIM was also inspired by the IBIS framework. It is mainly composed of "intents" that refer to what is wanted to achieve, "recommendations" that satisfy the intent, "justifications" that explain why a recommendation satisfies the intent, and the "context," which is the information generated during the design process.

The Decision Representation Language (DRL) [Lee, 1991] is also an IBIS-based DR tool. It uses the structure of the IBIS but extends its possibilities and provides a semi-formal representation and richer vocabulary. The main additions concern the relations between the various elements. For instance it is possible to create a hierarchy by defining relations like *Is-A-Part-Of(Object, Object)* of *Is-A-Kind-Of(Object, Object)*. It also makes the distinction between issues and questions whereas

no differentiation is made between them in the IBIS. DRL is used within the SIBYL system that aims at supporting the decision making.

InfoRat [Burge and Brown, 2000] (Inference over Rationale) supports a designer by checking the consistency and completeness of the DR. The completeness checks that the structure is complete, for example, there are no existing decisions that have no arguments in their favour. It also evaluates the choices by checking the selected decision has more arguments in its favour than against it. InfoRat uses a subset of the elements present in DRL. One drawback presented by the authors of InfoRat is that the acquisition of DR has to be done manually.

3.2.3.2 Limitations

Regli *et al.* [2000] pointed out in their survey that few DR tools are used in the industry. They try to understand the reasons and the obstacles. They pointed out issues on three main aspects, which are also present in knowledge management.

The first issues are related to the capture of the rationale. DR should few interfere with engineers' main tasks, *e.g.*, it should not change its habits and the engineers should not feel having additional work to do. Otherwise they will not find the time to use it. Hence an automatic capture approach would reduce the necessity to manually add the rationale in the system. Having a human centred approach will also have a positive impact on the system adoption. Furthermore DR acquisition is application dependent, thus providing generic methods is not efficient and need to be specialised in order to fit the application.

The second issue comes from the knowledge representation of DR. In IBIS-based systems, arguments are often represented in natural language, which reduces the computer processing capabilities. The future of DR systems is the representation of data in a computer understandable way. This will allow computers to process the information and reason about the knowledge. Furthermore the systems should be also to handle formal and informal knowledge.

Finally the DR system should provide an efficient way to retrieve captured information. Rationale retrieval is a complex task when the number of arguments in the system becomes large. One can rapidly get lost in the cross-references. Users need synoptic overviews and efficient query languages to access the data.

For these reasons, there are still reluctances from industries towards DR systems even if the theoretical benefits are well stated. Many research work addressed and proposed DR system but many improvements in the knowledge acquisition, representation, and retrieval are still needed before a widespread acceptance by the industry.

3.3 Chapter summary

Collaborative design has become the standard approach for the conception of products. This approach involves several stakeholders, having various expertises and viewpoints, to cooperate. It allows to shorten time-to-market, to lower product design costs, and to enhance the product quality. For this purpose collaborative platforms have been developed to support concurrent design.

The design is a succession of choices and decisions about the product. Decision making can become complex due to the heterogeneity of the involved experts' domains. In order to facilitate the process of decision making, several methods and tools have been developed. The methods allow each participant to propose his ideas and to support or to reject others. This approach helps to reach a consensus on an issue about a product and its design. An example of approach would be the Issue-Based Information System framework.

The decision making process is rich in information. A database containing the reasons behind design choices can be build by storing the proposed solutions and arguments. Thanks to design rationale the evolution of the design and decision can be documented. However design rationale has difficulties to be accepted by the industry. It currently interferes to much with the design process.

Chapter 4

Dependencies Management

Contents

4.1 Graphs theory

The graph theory is the study of graphs [Gross and Yellen, 2003]. A graph is a mathematical structure that allows to represent a set of objects and the relations between these objects. The objects in a graph are represented by nodes, also called vertices or points. Nodes can be represented graphically by points, by circles containing a label, etc. Relations between nodes are called edges or lines. An edge connects two nodes and can be directed or undirected. A directed edge has a direction, *i.e.*, the connected nodes are ordered. It is not the case for undirected edges, *e.g.*, considering two nodes a and b, a directed edge connecting a and b is different as a directed edge connecting b and a. This type of relation is usually represented by an arrow. In the case of an undirected edge, the same edge connects a to b and b to a. This relation is represented by a line. An edge can also create a loop when a node is connected to itself. A graph composed of directed edges is called a directed graph (or digraph) whereas a graph containing undirected edges is called an undirected graph.

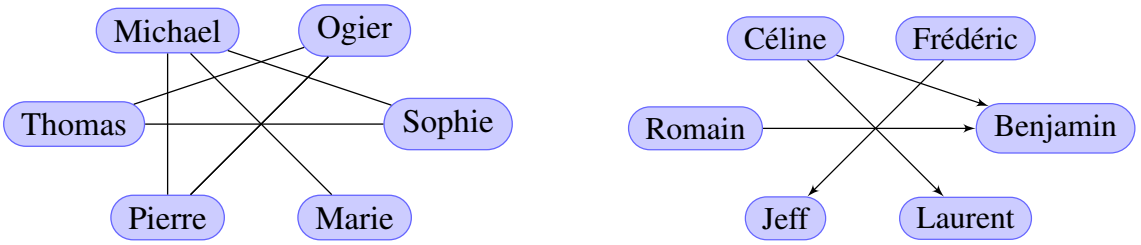
Figure 4.1 shows the representation of an undirected and a directed graph. The undirected graph of figure 4.1(a) illustrates friendship between some people. The use of undirected edges is relevant as the friendship can be considered as a symmetric relation. Figure 4.1(b) represents members of a research team, in which some persons are supervised. Here the relation is asymmetric.

A graph can be mathematical defined by $G = (V, E)$ such as:

V is a set of vertices, $V = \{v_1, \ldots, v_n\}$ with $n = |G|$, which is the *order* (number of vertices) of the graph.

E is a set of edges, $E = \{e_1, \ldots, e_m\}$ with $m = ||G||$, which is the number of edges of the graph. An edge e is defied as $e = (u, v)$ for a directed edge and $e = \{u, v\}$ for an undirected edge, where $u, v \in V$. To simplify the notation, an edge can also be represented by uv.

A path P is a non-empty graph that connects two vertices v_0 and v_k, such as $P = (V, E)$, with $V = \{v_1, v_2, \ldots, v_k\}$ and $E = \{v_0v_1, v_1v_2, \ldots, v_{k-1}v_k\}$. The length of the path is denoted k. A path P can be referred to by its sequence of vertices, *e.g.*, $P = v_0v_1 \ldots v_k$, or by calling P as the path from v_0 to v_k. Some basic but useful algorithms to determine paths in a graph are the Depth-first search (DFS) and the Breadth-first search [Knuth, 1997]. They can be used to find a path between two nodes but also to find all nodes reachable from a vertex. Furthermore they have many other applications in other graph related algorithms. The DFS algorithm (see algorithm 4.1) allows to determine if a nodes is reachable from another node in a graph. It uses a stack to store nodes that have to be visited. Visited nodes are marked in order to avoid to visit them again in the case of a cyclic graph. The algorithm finishes when there is no more nodes to visit. The Breadth-first search uses the same algorithm except that it uses a queue instead of a stack. Hence in the algorithm, as an ordered list is used as a stack, the `Queue` function of line 5, which takes the last element of the list (the most recently added) has to be replaced by the `Head` function, which takes the first element of the list.



(a) Undirected graph presenting people as node and friendship as relation.

(b) Directed graph representing members of a team with *supervisor-of* relations.

Figure 4.1 – Example of directed and undirected graphs.

Algorithm 4.1: Depth-first search

Result: True if the node is reachable from the starting node, false otherwise.

Input: A graph G (cyclic or acyclic, directed or non-directed).
Input: A node $r \in G$, which is the starting point for the search.
Input: A node f, which is searched.
Output: True or False.

```
1  begin
2      list ← ∅
3      Append (r, list )
4      while list ≠ ∅ do
5          currentNode ← Queue(list)
6          Mark (currentNode)
7          if currentNode = f then
8              return True
9          else
10             forall Unmarked child s of currentNode do
11                 Append (s, list)
12     return False
13 end
```

Graph theory addresses various aspects of graphs, such as graph computer representation, graph types, graph colouring, trees and graph dedicated algorithms [Diestel, 2005], etc. Graphs are used in numerous domains and can represent various objects, relations and configurations. A well known example of graph is the Web, in which resources are the nodes and hyperref links are directed vertices [Broder *et al.*, 2000]. In the scope of this work, graphs are used in order to represent dependencies. This kind of graphs is called dependency graph.

4.2 Dependency graphs

4.2.1 Definition

Dependency graphs (DG) are directed graphs used to represent dependencies between objects, for instance, an element A receives power from an element B. In a general context, DG are used for scheduling purposes like in:

Software compilation wherein they are used to compute the order in which the documents have to be compiled. They are also used by the compiler to optimise and improve the performance of programs by analysing instruction parallelism, pipelining, and scheduling [Heffernan and Wilken, 2005]. DG can also be used in order to remove dead code, which is code that can not be reach or that affects a dead variable, by looking at the inputs and outputs of the vertices that represent instructions.

Formula update and evaluation in spreadsheets resort to DG in order to compute the order in which the cells have to be updated.

Dependencies management within various fields like package management or visualisation [Peurriere, 2006].

An example, based on the equation system 4.1, is proposed in order to illustrate dependency graphs. This equation system generates dependencies between the different equations and variables. The dependencies between variables can be represented with a dependency graph (see figure 4.2). Each node represents a variable and each arrow has to be understood as the "source is dependent

on the target," *e.g.*, A depends on C and D. Nodes without outgoing edges do not have any dependencies.

$$\begin{cases} A = C + D \\ B = A \times F \\ C = F + \pi \\ D = E \div 3 \\ E = x + 1 \\ F = 42 \end{cases} \tag{4.1}$$

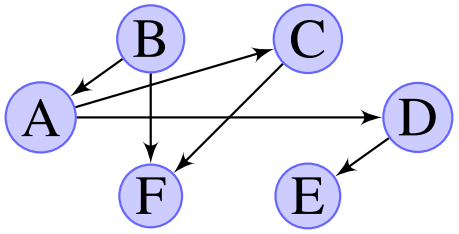


Figure 4.2 – Dependency graph between variables from equation set 4.1.

In this example the dependency graph addresses equations, but the same representation can be used for files, data, processes... This equation system and its corresponding graph will serve as example in the rest of this chapter.

4.2.2 Cycles

When addressing dependencies, one special case has to be mentioned: the cycles. There is a cycle in a graph when there exists a path that starts and ends at the same node. Graphs with at least one cycle are referred to as cyclic graph, those without cycle as acyclic graphs. To illustrate cycles, suppose that equation set 4.1 is modified by replacing the variant x by B. This results in a new dependency from E towards B. A cycle is thus created as shown by the red dashed arrows in the updated graph shown in figure 4.3. This case is problematic because in the case of an equation system, it becomes impossible to solve it without setting the value of a variable involved in the cycle because by transitivity, variables A, B, D, E are dependent on each other, *e.g.*, A needs E to be solved and E needs the value of A. Several algorithms are available to manipulate DG, such as to detect cycles.

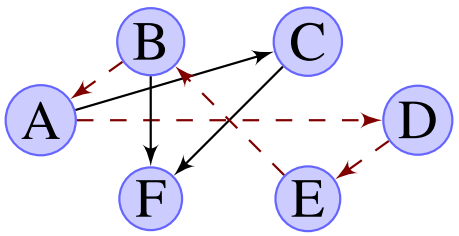


Figure 4.3 – Cycle in a graph. The dashed red arrows from a cycle.

4.3 Dependency management

Most of the existing algorithms that address directed graph can be applied to DG. Some of them, interesting for this work, will be presented.

4.3.1 Cycles management

As said previously, cycles can be problematic within DG, especially when it concerns dependencies solving. There are two main types of approaches to detect cycles, one works on the graph, the seconds works on a sequence of node that is obtained by the traversal of the graph.

In order to detect a sequence in a directed cyclic graph, a coloured DFS can be used [Knuth, 1998]. It uses three colours, usually white, grey and black. At the beginning all nodes are coloured in white. Then a DFS is successively launched on whites nodes. When a node is encountered during a DFS, it is coloured in grey. It will become black when all its descendants nodes have been visited. A cycle is detected in the graph when a grey node is reached. The algorithm stops when all nodes are coloured in black. For instance, this algorithm has been used to dynamically avoid deadlocks in computer software by finding cycles in the resource allocation graph [Jula and Candea, 2008].

The second presented approach allows to detect cycles in a sequence. The graph presented in figure 4.2 can be used to generate various sequences depending on the starting node and the taken path:

$$\begin{gathered} B, F \\ A, C, F \\ B, A, D, E, B, A, D, E, B, A \end{gathered} \tag{4.2}$$

When having a closer look at the sequence 4.2, a cycle can be identified: $B, A, D, E, \underline{B, A, D, E, B}, A$. In this case, the cycle can be exited by going from B to F.

An algorithm has been proposed by Floyd [1967] called the Floyd's cycle-finding algorithm or the "tortoise and the hare." This algorithm allows to find cycles in sequences generated by a function $f : S \mapsto S$ where S is a finite set of cardinality n. The sequence is constructed such as the element $x_{i+1} = f(x_i)$. This algorithm resorts to only two pointers on the sequence called the "tortoise" and the "hare." The tortoise and hare pointers are respectively positioned on x_i and x_{2i}, with i is the position in the sequence ($i > 0$). At the beginning $i = 1$ and at each step i is increased by one. A cycle is detected when $x_i = x_{2i}$. The space complexity is constant ($O(1)$) as only two pointers are used. The obtained sequence can be represented as a graph which looks like the Greek letter ρ.

This algorithm can be applied to other graphs when generating the sequence of visited nodes with a deterministic algorithm, which is the equivalent of the function f. In this case the sequence will loop or end at a node without outbound edges. Now suppose that the sequence 4.2 has been obtained by visiting the nodes starting from B and favouring the edges BA over BF and AD over AC. When applying Floyd's cycle-finding algorithm on the resulting sequence, a cycle is detected for $i = 4$ as $x_4 = x_8 = B$.

In many cases, the cycles in a graph have to be removed in order to have an acyclic directed graph. To remove a cycle, it is possible to remove an edge or a node involved in the cycle. In a cyclic graph, the feedback arc/edge set is a set of vertices that contains at least one edge involved in each cycle. Hence the removal of the edges present in this set results in an acyclic directed graph. Defining the minimal feedback arc set is a NP-complete problem [Alon, 2006]. The feedback vertex set is the equivalent set of feedback arc set but for vertices. It is also NP-complete [Festa *et al.*, 1999]. By removing edges of the feedback arc set or nodes from the feedback vertex set, a directed acyclic graph is obtained.

4.3.2 Scheduling

Dependency graphs can be used in order to represent and to schedule tasks, *i.e.*, provide the order in which the tasks have to be achieved to ensure that all dependencies of a tasks are completed before beginning this tasks. That is when precedence constraints are present and a legal ordering to performs tasks should be found. A task can be an update, a calculation, a project, etc.

The topological sort algorithm is a sort that linearly orders the nodes of an acyclic directed graph. The result of a topological sort of a graph G is the linear ordering of vertices in V, such as for all edges $(u, v) \in E$, u appears before v in the sequence. To illustrate this sort, it will be applied on the graph representing the dependencies in equation system 4.1 (see figure 4.2). The resulting ordered sequence, which is shown in the figure 4.4, is B, A, D, E, C, F. One can clearly see that the disposition of the nodes has changed and they are positioned linearly and that all the edges are now oriented from the left to the right. The topological sort does not guarantee the uniqueness of the solution as other sequences are also valid results. For instances B, A, C, F, D, E and B, A, C, D, E, F are both equivalent with the previous sequence.

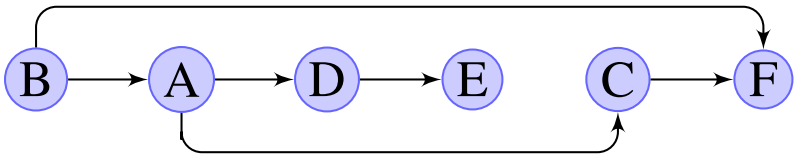


Figure 4.4 – Result of a topological sort on the dependency graph from figure 4.2.

The order, in which the equations have to be solved, is generated by the topological sort. However the result has to be reversed to obtain the right order. Equation 4.3 presents the progress of the resolution according to the reverse topological sort.

$$\begin{aligned} F &= 42 \\ C &= F + \pi \\ &= 42 + \pi \\ E &= x + 1 \\ D &= E \div 3 \\ &= (x + 1) \div 3 \\ A &= C + D \\ &= 42 + \pi + (x + 1) \div 3 \\ B &= A \times F \\ &= (42 + \pi + (x + 1) \div 3) \times 42 \end{aligned} \tag{4.3}$$

A topological sort can be obtained by using different algorithms. A first method to build the sequence is to iterate on the two steps below until there is no more vertices in the graph [Kahn, 1962]:

1. Append all vertices without inbound edges into the sequence (because they have no dependencies).
2. Remove the added vertices from the graph with their outgoing edges.

A second approach is to resort to a DFS. By appending to the sequence each completely visited node, a topologically ordered sequence is obtained by taking the reverse of the DFS result.

The topological sort is an efficient tool to oder elements having dependencies between them. Topological sort is available with a tool called “tsort” on Unix-like platforms and is also used within compilation tools like Makefile.

4.3.3 Graph visualisation

Graph visualisation and more especially graph layout is a complex problem combining flavours of graph theory and computational geometry. The objective is to show a graph in a convenient way in order to make it readable, easy to understand and to analyse. For this purpose, nodes have to be placed in a such way that they are well spaced and that the edges cross as few as possible. The interesting point is that, in some approaches, the layout is computed according to the relations between the nodes.

A first approach is based on spring forces, also called force directed techniques [Eades, 1984]. The idea consists in forming a mechanical system by replacing the nodes by steel rings and each edge by a spring. Springs have a defined natural length that will bring closer the connected nodes. With a friction, the system energy decreases until it reaches a stable state. The resulting layout from these algorithms is symmetric [Eades and Lin, 2000]. Another similar approach has been proposed by Fruchterman and Reingold [1991]. Their layout algorithm uses two principles: 1) vertices connected by an edge should be drawn near each other, and 2) vertices should not be drawn too close to each other. They used an analogy, in which vertices are atomic particles or celestial bodies. These bodies exert attractive and repulsive forces to each other, such that neighbourhood vertices are attracted and all other nodes repel each other. The layout is obtained when a minimum energy state is found by differential equations solving or a simulation of the system. Other algorithms exist that use other approaches like the one implemented in a tool called GIOTTO [Tamassia *et al.*, 1988]. GIOTTO places edges and vertices on a grid by using an algorithm composed of three phases: planarisation, orthogonalisation, and compaction. Nevertheless these algorithms address mainly undirected graphs.

A method for drawing directed graphs has been presented by Sugiyama *et al.* [1981]. Digraphs can be represented as a hierarchy in order to enhance their representation. The proposed algorithm creates a hierarchy where nodes are placed into several levels. The algorithm is composed of four phases:

1. Place the graph nodes in discrete levels according to the relations between nodes.
2. Order nodes within layers to avoid crossing edges.
3. Compute the coordinates of nodes.
4. Draw the hierarchy picture.

This method works with acyclic graphs. In the case of a cyclic graph, the nodes involved in the cycle are merged and the graph becomes condensed. Some enhancement have been made to handle cyclic graphs, such as to revere some edges to make it acyclic [Row *et al.*, 1987].

The method from Sugiyama *et al.* [1981] builds a hierarchy composed of n levels, from a directed and acyclic graph. The hierarchy is denoted $G = (V, E, n, \sigma)$, where:

- V is a set of vertices such as $V = V_1 \cup V_2 \cup \cdots \cup V_n \quad (V_i \cap V_j = \emptyset, i \neq j)$, where V_i is the set of vertices of level i and n the height of the hierarchy.
- E is a set of edges, where each edge is unique.
- σ is a set of sequence σ_i for each V_i. σ_i is the sequence of vertices within V_i such as $\sigma_i = v_1, v_2, \ldots, v_{|V_i|}$ with $|V_i|$ the number of vertices of V_i.

To create the hierarchy, each directed edge $e = (source, target)$ has to satisfy the following condition:

$$e = (v_i, v_j) \in E, v_i \in V_i \text{ and } v_j \in V_j \text{ satisfies } i < j \tag{4.4}$$

To illustrate the method, it has been applied to the graph resulting from equation set 4.1. It can certainly be agreed that the current representation presented in figure 4.2 is not easy to read. The new layout, which results from the presented algorithm from Sugiyama *et al.* [1981] is presented in figure 4.5. It results in a four levels hierarchy. The organisation of the nodes facilitates the understanding of the graph as the edges are not crossed and the vertices well positioned.

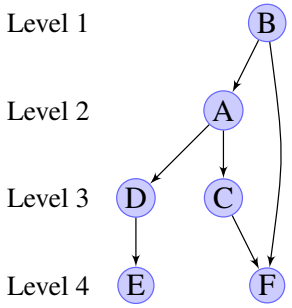


Figure 4.5 – Four-level hierarchy resulting from the application of Sugiyama *et al.* [1981]'s graph layout algorithm on the graph presented in figure 4.2.

4.4 Chapter summary

Graphs are powerful tools to represent objects and their relationships. They are used in numerous domains and have been the subject of many research works. Thus several algorithms are now available to handle graphs.

Directed graphs allow to represent dependencies between elements. These graphs, used for this purpose, are called dependency graphs. Dependency graphs allow to schedule tasks by using dedicated algorithms like the topological sort. This is particularly useful in domains wherein some elements have to be processed before others, like in compilation or for the evaluation of formulas. However to successfully schedule tasks, the graph should not contain any cycles. Cycles can be detected *via* coloured depth-first search or by the "tortoise and the hare" algorithms. Then several solutions are available to remove a cycle. For example it is possible to remove an edge involved in a cycle or to inverse its direction.

Graph visualisation techniques propose algorithms to make graphs human readable by addressing layout issues. Layout algorithms use relations between nodes to efficiently position them. For instance in force-directed techniques, a node is attracted by its connected nodes whereas it is repulsed by other nodes. Other algorithms use relations to position nodes on levels so that the edges all go towards the same direction.

Chapter 5

Knowledge-Based Engineering Templates

Contents

5.1 Template applications

Knowledge-based engineering templates were introduced in chapter 1. They aim at storing design knowledge for later reuse. In this section an overview of several template related research works is given.

Nowadays customers are in focus of the product design, for instance, regarding the ergonomics of the products or the customer's needs. Mass customisation is "the producing goods and services to meet individual customer's needs with near mass production efficiency" [Tseng and Jiao, 2007]. The objective of mass customisation is to allow customers to get involved at a stage of the design or the manufacturing in order to personalise the product. Computers play an important role to successfully merge mass production and the flexibility of customisation in order to produce custom output. However the product customisation reduces the productivity of mass production. A higher level of productivity can be achieved by the use of templates and the enhancement of product development processes [Cox *et al.*, 2003]. Within product development, CAD templates provide easily reconfigurable models. According to Cox *et al.* [2003], their instantiation time averages 3% of the time needed to create the model. Furthermore the increase of product complexity encourages the use of reconfigurable models and templates. The time spent in the design of a reconfigurable model is regained by the reduction of later designs [Cox *et al.*, 2003].

To illustrate that, Siddique and Boddu [2005] proposed a concrete example of mass customisation framework. It allows to tailor products to the customer's needs by integrating the client into the design process. This is achieved by the use of parametrised templates. The customer configures the product model *via* parameters and the result is shown in real time. They demonstrated the framework on a bicycle frame customisation example.

Kamrani and Vijayan [2006] proposed a structure for the implementation of template-based systems. The use of templates associated with computer-aided process planning shows a reduction of the development time of new products. It also gives the designers a better understanding of the costs of design changes with respect to the manufacturing.

The main goal of designing templates is to store and reuse knowledge. A key success factor is to identify and capture the relevant knowledge. Skarka [2007] developed a methodology to construct generative models, *e.g.*, CAD templates. His approach is based on the MOKA methodology [Stokes, 2001] (see section 1.1.2.4). They designed an OWL ontology for MOKA model records. The formal knowledge representation is realised within the steps "4. Formalise" and "5. Package" from MOKA, by using CATIA V5 and its KBE tools. The proposed methodology was developed in order to propose a knowledge acquisition tool for the design of generative models within CATIA V5. It was developed as at this time, there was no such tool. Regarding knowledge about templates, Alani [2007] worked on the management of template design knowledge. He proposed a template-based knowledge system and defined a knowledge model divided into four meta classes: application, function, element, and process. Within these classes or categories the templates and their elements are organised. The knowledge model is represented with OWL. It is coupled with the CAD templates through web technologies like XML and HTML. XML is used to export the template knowledge. The resulting XML file can then be imported and checked by a knowledge designer before being committed to the ontology. Knowledge can also be transferred the other way round, from the ontology to the CAD template *via* HTML pages. Thus the template can be defined first in the CAD system or in the ontology. The approach proposed by Alani [2007] requires at least two domain experts, one for the management of the ontology's aspects and one for the CAD system. Furthermore the representation between the both systems (ontology and CAD) needs to have the same structure in order to be synchronised.

Resorting to knowledge templates has proven to be especially useful when having a large range of complex products that share components having nearly the same geometry/functionalities. This is the case within the aerospace and automotive industries. La Rocca and van Tooren [2005] introduced *High Level Primitives* (HLP). HLP are parametric models that interact with the design process by modifying inputs values. The definition they gave is "generic entities with a similar functionality, shape and behaviour." HLP should be able to fulfil functionalities such as generate lift, hold a metal sheet or allow a person to reach a platform. Their definition is very close to the

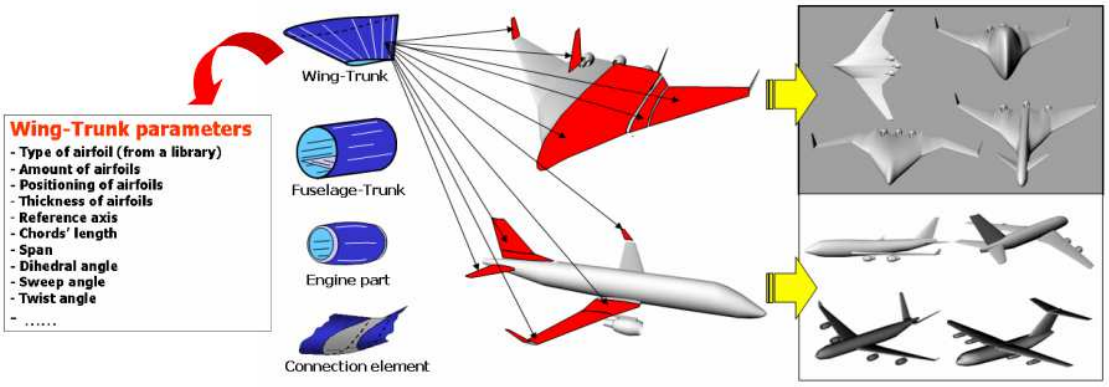


Figure 5.1 – High level primitive modelling approach that allows to generate variants of a model [La Rocca and van Tooren, 2005].

definition presented in this work. They present HLP as an efficient way to virtually manipulate ideas and create rapidly various design solutions. HLP are defined in a parametric way so they can morph themselves according to given inputs. Figure 5.1 shows their design approach with HLP. With a reduced set of HLP (wing-trunk, fuselage-trunk, engine part and connection element) they can generate a broad range of aircraft configurations. By tuning the parameters, the design can be modified, the number of internal elements (spars, ribs, etc.) adapted or the topology changed. These HLP can be generated and optimised *via* the Design and Engineering Engine (DEE) [La Rocca and van Tooren, 2009; van Tooren and Rocca, 2008]. The DEE is presented in figure 5.2. The main component is the Multi-Model Generator (MMG). It is a KBE application that allows to generate several models for various analyses and computation systems. The MMG receives its inputs from the initiator, which computes them from some requirements given by the user. The outputs of the MMG are transferred to the "converger and evaluator." These outputs are the results of the various analyses on the generated models. The results are checked to see if the generated design has converged. Otherwise new values are given to the MMG and the loop goes on till it converges. Finally the converged solution is checked to see if it meets the objectives. If this is the case, an optimised design solution according to the requirements is achieved, otherwise it starts again with new initial parameters.

5.2 Template-based design process

The use of knowledge templates influences the design of products. In the previous section, the Design and Engineering Engine that uses templates to design aircrafts bodies has been presented. In this scope templates are used as building blocks that can take various shapes due to their KBE parametric definition. These building blocks can then be combined together in order to generate different aircraft configurations [La Rocca and van Tooren, 2005]. This allows to test rapidly several design variants in the early design phases. The generated variants can be optimised *via* Multidisciplinary Design Optimisation (MDO). MDO is a promising methodology in the scope of complex product design. It aims at solving design problems involving several disciplines by resorting to optimisation techniques. However this approach lacks of high-fidelity analysis tools in order to reach a high level of automation [La Rocca and van Tooren, 2009]. Through the currently developed DEE and the use of High Level Primitives (templates) they address the automation issue with various analysis tools. The KBE templates are used to capture and to record the knowledge to process and generate the geometry.

KBE templates and template-based construction processes have been research topics at Daimler AG, an automotive company. The knowledge templates are integrated into the design process [Arndt *et al.*, 2006; Katzenbach *et al.*, 2007]. Template products as well as template features

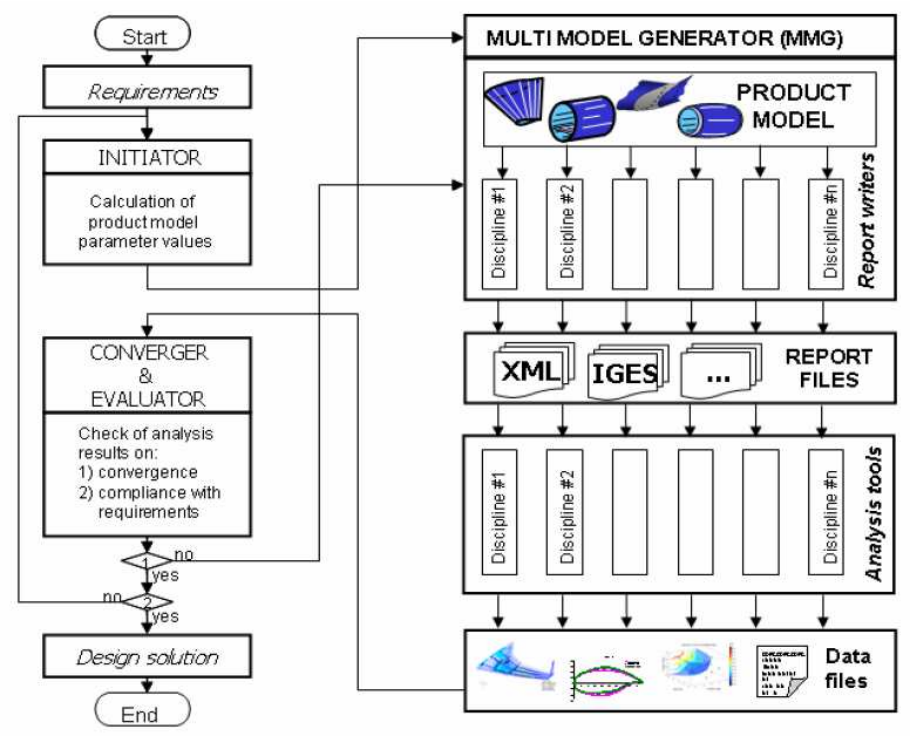


Figure 5.2 – Paradigm of the Design and Engineering Engine with the Multi-Model Generator [La Rocca and van Tooren, 2009].

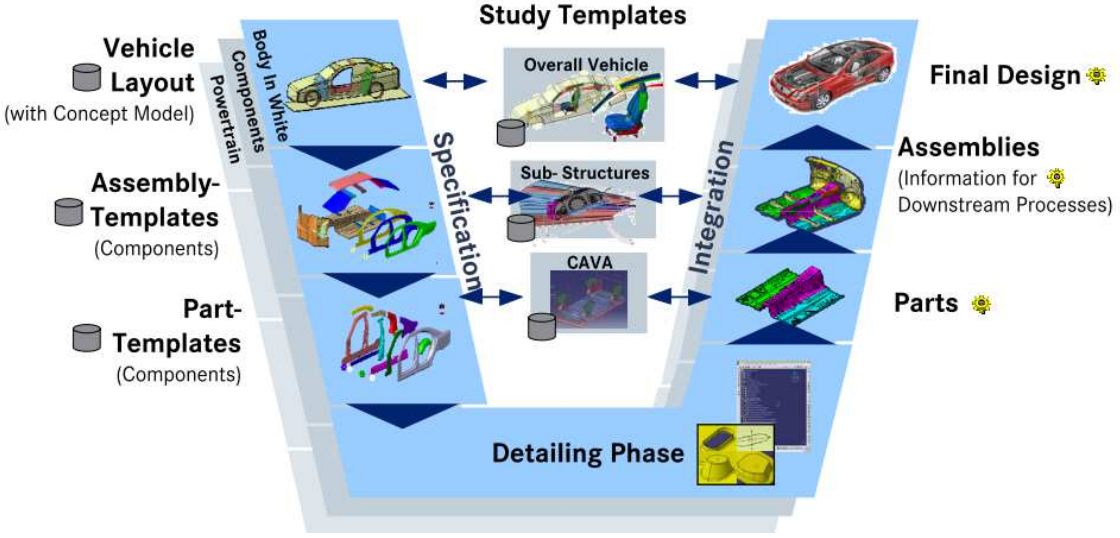


Figure 5.3 – Template-based V-model design process at Daimler AG [Katzenbach *et al.*, 2007].

are proposed to designers through libraries or catalogues [Mbang, 2008]. The design process using templates follows a V-model as presented in figure 5.3. The V-model starts from layout definition consisting of the basic structure, *e.g.*, body-in-white of a car and powertrain. It is then refined to provide details *via* the instantiation of templates, first assembly templates, then part templates and finally feature templates for the detailing phase. Then the various parts are assembled to reach the final design. During the V process, study templates are applied at several levels to evaluate the design. Templates allow to standardise the design concepts and share them between several products.

A study on the human factors regarding template use has been conducted by Katzenbach *et al.* [2007] at Daimler AG. The objective was to evaluate the human factor in the adoption of templates aiming at the standardisation of the process and to improve how templates are designed in order to facilitate its acceptance. The study consisted in evaluating and analysing two teams that had to realise two design tasks of different complexity by using templates. The outcome was that the use of templates brings benefits in the case of complex design tasks. The resulting design was better structured and was accomplished in a shorter time. The use of templates for the less complex task resulted in a longer design. The reason behind is that well structuring the model has a weaker impact and it takes more time to handle the templates than to structure the model's elements. The engineering design experience also influences the perception and the use of templates because of their complexity. The study also pointed out the need for efficient visualisation tools for relations and interdependencies between construction elements.

Mbang [2008] predicted that in the future engineers will not design products from scratch anymore. Instead they will used templates that are pre-designed elements that can adapt themselves to various contexts. Templates can have several levels of details to provide more or less flexibility. Features can then be used as construction elements to provide the details. He also proposed an approach to integrate Product, Process and Resources. This is achieved by using templates. In his approach the templates are products or features that fulfil one or several functions. The process aspects describes how the template will be manufactured. The resources aspect lists the necessary tools and materials to realise the feature.

5.3 Template management

Knowledge templates are KBE applications that typically evolve continuously. This implies to manage the versioning, the configurations of the templates, and other Product Data Management issues. Furthermore a template can be composed of many elements that are interconnected. This results in tight relationships that have also to be handled.

The use of templates results in the creation of several instances located in different contexts. When the inputs of a template instance are set, it creates new relations between the instance and elements present in the context. The management of the resulting links constitutes another highly important aspect of template management [Katzenbach *et al.*, 2007].

5.3.1 Link management

The use of KBE results in the creation of many relations between parameters, formulas, rules, etc. The complexity increases when gathering several KBE applications. For instance more than 2500 links are present in a body-in-white template structure [Katzenbach *et al.*, 2007]. Links are tightly related to the structure of the application. In the generic structure of a template presented in figure 1.5 on page 14, the link flow is well organised. The structure is clear and easy to understand. The link management allows to divide complex structures into several template-based structures, which can thereafter be reused in a more convenient way [Katzenbach *et al.*, 2007].

A result from the study of Katzenbach *et al.* [2007] on psychological human aspects is that the complexity of the problem increases when the relations between elements are not easily accessible. Lukibanov [2005] suggested the use of ontologies to explicitly represent templates and

their relations in order to facilitate the management of a large number of templates. In this way engineers can visualise relations and input parameters of templates without resorting to a CAD system. Katzenbach *et al.* [2007] pointed out that in the CATIA V5 CAD system, all links are not directly visible and that there is a lack of link overview tools.

As previously stated, links are related to the structure of the templates and assemblies. The creation of a link often corresponds to a design intent and thus has a meaning. Arndt [2007] proposed a method using an ontology-based knowledge model in order to support product decision making. In one of his applications, he specialised the knowledge model in order to integrate the template link flow aspects, which are relations between templates and other documents. These relations can also be documented with the design justifications. The aim is to provide information to CAD template designers as well as to CAD template users and thus to allow to faster reach a high degree of products maturity.

From the several works and studies presented above, it can be asserted that the focus on link management aspects is strong. The number of links can quickly reach several thousands in complex scenarios involving several knowledge templates and KBE elements. It results in a relationship network that is complex to understand without the help of suitable tools. The relations represent the structure of the model. Links visualisation is helpful to rapidly analyse the structure of a template or the context of an instance. The relations generate dependencies between the involved elements. When a template is modified, the data flow resulting from the modifications is transmitted through the links. Hence the changes are likely to impact other elements. This is why the update of templates has to be managed to avoid errors.

5.3.2 Update management

Templates are modified during their life cycle (see figure 1.7 at page 15). These modifications will impact linked documents. However the update of the linked documents does not always consist in the update of parameter values, but also in the modification of the geometry.

In some cases, the new template version resulting from the update has to be distributed, *i.e.*, its instances should be updated too in order to benefit from the changes. The distribution of the latest template versions would also reduce the number of different versions of the template that are in use, which would facilitate the maintenance. However the update of a template instance may impact the context wherein it is used, through the modification of the template inputs/outputs. This aspect should also be taken into account when updating a template instance.

The update management addresses many issues. As mentioned previously, the number of parameters and links within large KBE assemblies or templates can easily reach several thousands. Template instances are subject to the same behaviour. Thus the maintenance of the whole is a challenging task due to this large amount of data.

Few research work study template update. Lukibanov [2005] addressed the problem of template management and more especially the update. He focused his work on the relations and links between templates. His objective was to provide a tool that would assist engineers in the template management. The approach he proposed was based on ontologies and focussed on the CATIA V5 CAD system. The ontologies allow to represent various kinds of relations that can exist within and between templates. Thanks to the ontologies, relations can be managed outside the CAD system for more flexibility, *e.g.*, integrated within other tools, for example, in order to analyse them. For that purpose he developed a tool that analyses CATIA V5 templates and maps them to the knowledge model defined in the ontology. The input parameters and relations can then be visualised and analysed in order to validate the template changes. He also proposed a business process for template update, which is presented in figure 5.4. The first step of the process consists in the development of the template. Then a loop starts, in which the engineer in charge of template updates checks the dependencies by visualising the data in the ontology and then modifies dependent templates as necessary. Once all the updates are completed, the ontology has to be updated in order to include the recent modifications. The dependencies are then computed within the ontology. This is a semi-automated task in which the engineers can resort to OntoWorks, an

internal tool from Daimler AG based on Protégé 2000, in order to fix the mapping of parameters in case of changed parameter name.

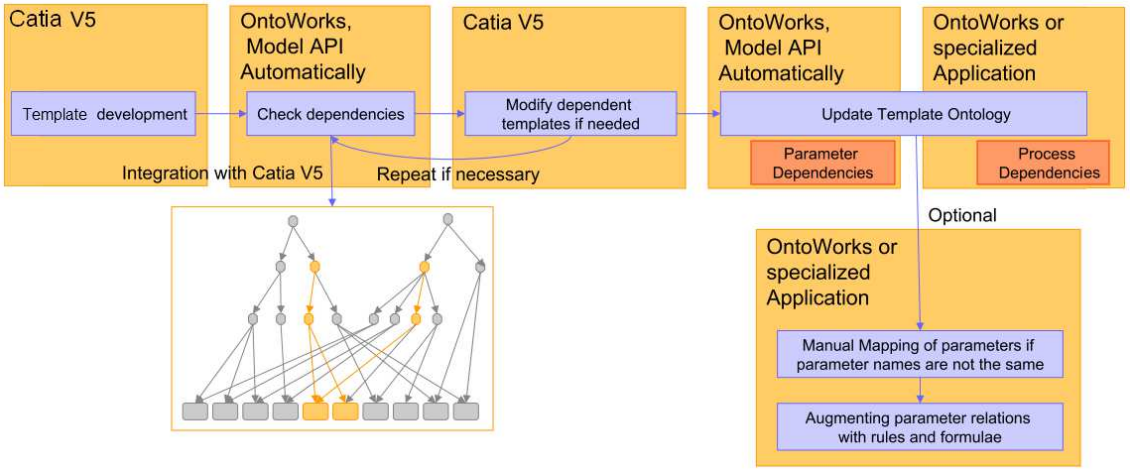


Figure 5.4 – Business process for template update proposed by LUKIBANOV [2005].

The approach proposed by LUKIBANOV [2005] allows to facilitate the management of templates in a complex design environment, wherein a large number of templates are interconnected. However template instances are not mentioned in his work. Thus there may be a limitation, which is that only templates defined within another templates are addressed. His work seems to focus on the definition of templates.

Regarding the ontological representation, the information extracted from the CAD models are stored as classes within the ontology. However concrete objects, like the extracted data, should be represented as instances of the defined ontology classes. Furthermore, this approach does not take advantage of inference mechanisms, which is one of the main benefit of using ontologies.

5.4 Chapter summary

Knowledge templates technologies slowly start being used within companies. First companies were the automotive and the aerospace industries, in which the complexity of the models requires the reuse of knowledge to speed up the design. However the design process has to be adapted to take advantages of the template technologies. In new design processes, templates are used as building blocks, from the main concept models till the detailed products. This results in a variety of template types according to their level of details.

The template-based design raises some issues, especially concerning the management of templates. The use of templates generates many dependencies towards others templates and documents. This results in the creation of links between documents. These links are fundamental to understand the structure of KBE assemblies. Hence some approaches propose to gather, to represent and to document them. The dependencies generated by the links are also a key factor for the template update. The modifications applied to a template have to be propagated and it has to be ensued that no model related to an instance gets broken due to the update. For this purpose, an efficient visualisation of the links is a mean to supervise the updates.

The investigations showed that current template management approaches are often narrowed to the template definitions and do not take into account the instances. However template instances are a current concern of template maintainers. Furthermore better support tools could be provided to engineers in order to save time when doing tasks related to the template update, especially when including the management of instances. The tasks before the update of a template are currently

not addressed, for example, the collaborative process resulting to a design choice to upgrade the template.

PART III

Case study

Chapter 6

Study of CATIA V5

Contents

6.1 Dassault Systèmes CATIA V5

The purpose of this chapter is to present CATIA V5 CAD system. However it is not meant to be a comprehensive description and not all components of CATIA V5 nor the design with CATIA V5 will be introduced. The focus is put on templates and links related aspects. In the scope of this work the Release 19 of CATIA V5 was analysed.

6.1.1 CATIA presentation

CATIA is a 3D computer-aided design software initially designed by Dassault Aviation for its own needs during the 1970's. Its initial name was CATI. In 1981 CATI became CATIA and started being maintained by an independent unit from Dassault Aviation called Dassault Systèmes (DS). Since 1981 IBM is in charge of marketing and distribution of CATIA. In 1982 the first customers of CATIA were automotive and aerospace industries like BMW, Dassault Aviation, Mercedes-Benz or Snecma. Two years later the second version of CATIA came out: CATIA V2 that had a new architecture and provided colour graphics. In 1988 CATIA V3 marked the migration of CATIA from mainframe computers to UNIX. Following the acquisition of CADAM in 1992 by DS, CATIA V4 was released in 1993. It provides an increased openness and an innovative approach to mechanical design. In 1999 the version 5 of CATIA was released, which is a complete rewrite of the software, introduced a new user interface, feature-based design and Microsoft Windows support. The latest version, CATIA V6 was released in 2008. With this version the support for Unix platforms was discontinued and a tight integration with Product Lifecycle Management is provided.

CATIA is used in numerous domains like automotive, aerospace, ship building, architecture, electricity or consumer goods design [Guennuni, 2008]. In the scope of this work, the version 5 of CATIA was studied.

6.1.2 CATIA V5

The version 5 of CATIA is a Computer-Aided Design/Manufacturing/Engineering system. It is composed of various workbenches, which are dedicated to specific tasks like part and assembly design, drafting, finite elements analysis, digital mock-up, machining or knowledgeware. An important aspect of CATIA V5 is the features-based design. At the contrary to CATIA V4 in which the geometry is defined *via* boolean operations, the geometric design in CATIA V5 is based on parametric elements called features that are high level operations with functionalities. Brass [2005] defined features as composed of (i) associative information: features can have parent/child relations towards other features, *e.g.*, a hole has a surface as parent element, (ii) mathematical information: a feature has geometric and topology information, and (iii) attributes: a feature is also defined by attributes like its colour, the length or the type of line. CATIA V5 allows parametric and associative design by the use of features. That means features can be easily modified through the change of parameters' value. The modification of the feature implies the update of child features due to the association. This process is managed by the system and thus frees the designer of managing dependencies. Hence he can focus on the design. Feature are an efficient way to present the design intents of the designers. The feature based design results in a specification tree that presents the evolution of the design (see figure 6.3(a)).

In the continuity of parametric design, CATIA V5 proposes a Knowledge-Based Engineering (KBE) workbench. This workbench allows to store knowledge within models. It provides user parameters, formulas that allow to dynamically compute the value of parameters, rules that contain specific actions triggered according to defined conditions, and checks that provide a feedback to the designer if the current configuration is valid according the given requirements. The KBE workbench also provides the possibility to optimise parameters or to define knowledge templates.

CATIA V5 has his own proprietary file formats, which are, for instance, CATPart for a single model or CATProduct for an assembly. However it can export and import models into other formats, such as STEP or 3DXML. However the specification tree, and thus the design intents and history, are lost during the export.

6.1.3 Reasons for selecting the CATIA V5 system

Today CATIA V5 has two main competitor systems that are NX (Siemens PLM software) and Pro/ENGINEER from Parametric Technology Corporation. Furthermore a new version of CATIA has been released recently.

Pro/ENGINEER is also a CAD/CAM/CAE solution. It was the first CAD software to provide feature, parametric and associative-based design in the market. It has a modular structure that allows to easily add new functionalities. For instance, the Pro/ENGINEER Expert Framework provides KBE capabilities. NX is a suite of integrated CAD, CAM and CAE applications. It also provides knowledge-based automation tools. However many automotive and aerospace industries are using mainly CATIA V5. Thus it seemed the most relevant CAD system to investigate.

CATIA V6 is the latest CAx tool from Dassault Systèmes and the successor of CATA V5. However CATIA V6 was not available at the beginning of this work. In the automotive and aerospace industries, projects are complex and can last many years. It is rare that a company switches or does a major upgrade of a software during a projet to prevent compatibility issues. Thus CATIA V5 will most likely be used for another few years. Some companies have only migrated recently from CATIA V4 to CATIA V5, about 10 years after its first release.

For both reasons CATIA software was selected instead of Pro/ENGINEER or NX for the primary analysis.

6.2 Templates in CATIA V5

6.2.1 Technologies

Two main technical solutions to define and use templates with CATIA V5 have been identified. The first is to resort to the knowledge workbenches. This solution was defined as the “Dassault technology” to clearly differentiate it from the second approach that is more generic.

6.2.1.1 Dassault technology

The most convenient way to create and use templates within CATIA V5 is to resort to the dedicated CATIA workbenches: “Product Knowledge Template” and “Business Process Knowledge Template.” There are three types of template that can be created within these workbenches:

The feature template is a construction element defined by the user. It allows to define advanced features that are composed of other features, and geometrical and KBE elements. Two types of feature templates are disposable: the PowerCopy (PwC) and the User-Defined Feature (UDF). The difference between them is to be found in the instantiation result. The result of the instantiation of a PwC is a white box whereas it is a black box for a UDF. More details are given in section 6.2.2.

The document template is a complete model, which can be a single part or an assembly, that is meant to be reused. Document templates can only be defined within geometrical models that are stored in CATPart or CATProduct files. However a document template can contain other documents that are external to the model like spreadsheets or drawings.

Business process template present the capability to design customised workbenches. In this way specific process can be implemented and deployed within the company. It also enables the creation of design and engineering task sequences that can be automated.

In the scope of this work only the feature and document templates have been investigated. Business templates have been considered as out of the scope for several reasons. First the focus is put on the design itself and the models, and not on the design process. Second according to received feedback, business process knowledge templates are currently little used within companies. Thus the “Product Knowledge Template” (PKT) workbench has been investigated.

A benefit from the PKT workbench is that it provides a wizard user interface to define and use templates. To define a template, the user has to select the content of the template, and define its inputs and possible outputs (called "publications" in CATIA V5). The definition of a template creates a specific feature in the document. This feature represents and contains the template definition. Hence a document template is a feature provided in a document, and not the document itself.

Templates can be referenced and organised within libraries in order to facilitate the use of templates. In CATIA V5 libraries are called "catalog."

6.2.1.2 Document as template

Templates can also be defined and used without resorting to the template workbench from CATIA V5. However this approach implies no automation in the process of template definition and use. This approach has been referred as the use of a "document as template." This corresponds to the use of a CATPart or a CATProduct as a template, *i.e.*, reuse this document *via* copy-paste for example. An interesting point is that this concept can be extended to other types of documents if they fit the template schema of figure 1.5. Unlike templates defined *via* the dedicated workbench, here the template definition is the document and not a feature. That means that the organisation of the document content is crucial to facilitate its instantiation because there is no explicit definition and no wizard to guide the template user. Thus it is more practical to group all inputs in order to identify them easily.

As the definition of a template is not explicit, its definition relies on the user who will, for example, store the document in a folder gathering the templates or in a specific category of a catalogue. This lack of explicit definition makes difficult the use of this approach to define feature templates. For this reason, only documents are considered for the "document as template" concept.

6.2.2 Instantiation of knowledge templates

6.2.2.1 Differences between templates' definitions

In the previous section, it has been seen that there are two ways of using templates. Either with the dedicated knowledge template workbench or by using a standard document as a template.

With the knowledge template workbench, the user wishing to instantiate a template can do it through an user interface dedicated for this purpose. The inputs of the template are then presented and the user only needs to select the corresponding elements in the future context. The template designer can define that all inputs are mandatory. A preview of the result of the instantiation is also available to the user. When he validates, the process is complete, *i.e.*, a copy of the template has been put in the context (as well as on the file system) and the inputs are linked.

Without this workbench, all actions have to be done manually. First a copy of the template document has to be made on the file system. The user has to take care with the UUID for the files. UUID are unique identifiers given by CATIA when it creates a file. CATIA can not load two files with the same UUID at the same time. The copied file has also to be renamed in the case of multiple instantiations. Then the designer can load the new document in the context. Again some conflicts can arise with the names. After that he has to find and to set the inputs of the template manually. No information are given if all inputs are mandatory or not. Finally, in the case an input of the template is published, the designer has to update the publication because it gets broken when replacing the input.

6.2.2.2 Remarks about template instances

Both presented approaches create a template instance. A template instance has a separate life cycle from its definition. Hence templates and instances can be modified independently.

An important noticed behaviour is that the way the template is defined (knowledge workbench or document as template) has no impact on the resulting instance. That means that the way the template has been defined and instantiated can not be distinguished by looking at the instance. Actually it is also not possible to differentiate an instance from a standard CATIA document. The instantiation process generates a standard document or standard features for PwC. After the instantiation no relationship is maintained between the template and its instances. This is quite problematic regarding template management. CATIA V5 provides no solution to track templates' instances.

Two types of instances have been identified: the direct and the indirect instances. They are resulting of different instantiation procedures. The direct instances are the instances described so far. They result from the instantiation of a template. An indirect instances corresponds to the result of the copy of a template instance included in another template. This leads to template interweavings that are common but have to be addressed in order to avoid redundant updates. Figure 6.1 illustrates both types of instances.

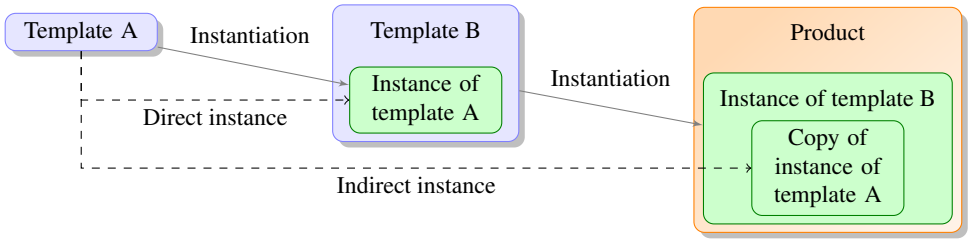


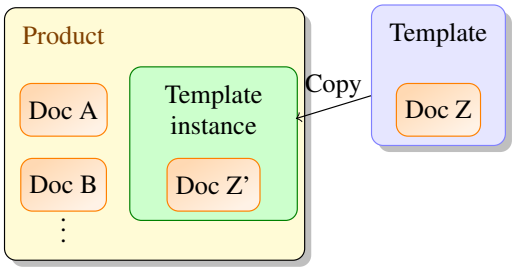
Figure 6.1 – Schema of direct and indirect instances of the template A.

6.2.2.3 Instance's content

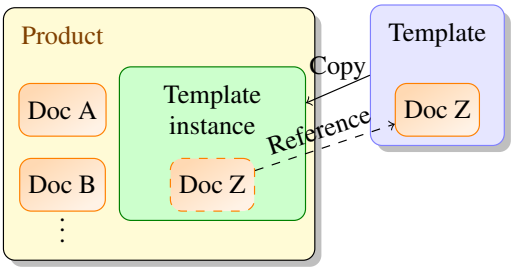
In section 6.2.1.1, two types of feature templates have been presented: the PowerCopies being white boxes and the User-Defined Functions being black boxes. The difference is in the instantiation result. The instantiation of a PwC adds the content and the structure of the feature template (elements defining the template) to the design as well as to the specification tree. The PwC user can thus see and modify the result. This is not possible with UDFs. UDF instantiation results in a modification in the design like the PwC, but the content of the instance is hidden and not accessible by the template user. The specification tree only presents one unique feature. Hence UDFs can be used for intellectual property protection. Regarding document templates, there is no such protection. Thus an external solution has to be used. A workaround would be to encapsulate the knowledge within an UDF and instantiate it in the document template.

CATIA proposes two instantiation approaches for the documents contained in a product template. Figure 6.2 illustrates them. The first one is like the one for template instances: the content of the product template is also copied into the context. It results in the creation of a new document that will be a copy (Doc Z and Z' in figure 6.2(a)). The second one consists in a reference from the template instance to the original document (Doc Z in figure 6.2(b)). Both approaches present their pros and cons. The former, by making copies, allows to apply local modifications to the instances. Nevertheless this diversity raises management issues. The latter facilitates the management of the template and its instances as it avoids multiple copies of the document contained in the template. But you can not make any modifications to any of the instances because they would also appear in other template instances.

The first method, which creates a copy of the template, presents the best characteristics for multiple instantiations.



(a) The instantiation creates a copy of Doc Z, which is the content of the template.



(b) The Doc Z' is not copied and refers the original.

Figure 6.2 – Presentation of the two approaches for the instantiation of a KBE product template within CATIA V5.

6.2.3 Templates update

CATIA V5 provides an update mechanism between linked documents. For instance, when the value of an user parameter is modified, all related elements in or outside the document wherein the parameter is defined are updated, *i.e.*, the update is propagated through relations. In the case of a major modification, this mechanism is not sufficient and the engineer will have to manually update the related elements. Such a change could be a modification of the geometry or of the type of an element. In this case, the engineer will have to analyse the relations. But CATIA V5 does not provide a practical visualisation tool for relations. Only one assembly can be visualised at a time with the "Desk..." tool. The link types are shown in another location (Edit/Links) and only the links from one document are available at a time.

However, as said previously, there is no explicit relation/link between a template and its instances. Hence there is no synchronisation between them, they are independent from each other. Furthermore the engineer can not easily find an instance of a template in the case he wants to update it.

6.3 Links and relations

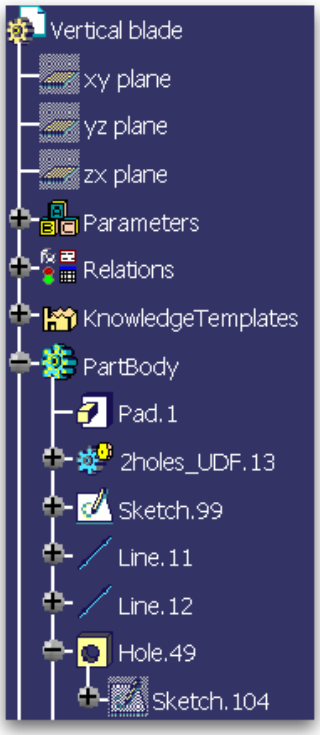
Links and relations play in general an important role for the update of models in an assembly. For this reasons this area has been investigated in order to reveal what links and types of links are present in CATIA V5 and what the individual impacts on the update of a model are. Guennuni [2008] identified three types of relation within CATIA V5: parent-child relations, aggregations and Multi-Models Links (MML). Thus all of them had to be investigated.

6.3.1 Relations within documents

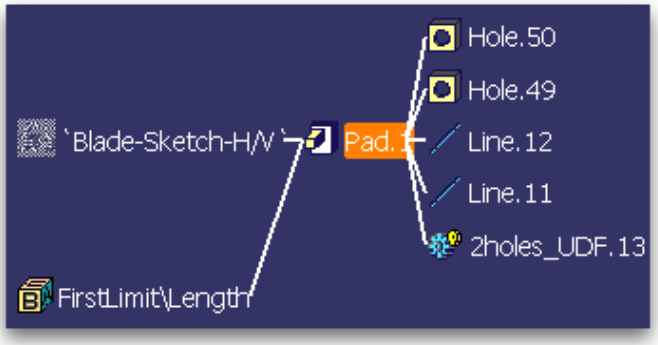
Within a CATIA V5 document, there are two types of relations: parent-child and aggregation. It concerns principally CATParts and CATProducts, which are the documents that contain the design information.

An aggregation is a relation allowing to structure the design. It is a relation between two objects meaning *has* or *is composed of*. Hence an element is composed of aggregates. However an aggregate can only be aggregated to one object. For this reason the resulting structure is a hierarchy that can be represented by a tree. Figure 6.3(a) shows the specification tree of a CATPart, which typically presents the aggregation tree. The `Vertical Blade` part is composed of planes, a set of parameters and a set of relations, knowledge template definition and the part body set. The part body is composed of various features that can also be composed of features and so on. In the same way, there is an aggregation between an assembly and the parts composing it.

A parent-child relation is typically a dependency relation between two elements, in the way that the child is dependent on the parent. So it is a directed relation. A parent can have several children, as well as a child can have several parents. The resulting structure is a directed graph. Figure 6.3(b) presents the parent-child view available in CATIA V5. Depicted are the parent/child relations of the Pad.1 feature. Parents are located on the left of the analysed feature, children on the right. The relations are represented by a line. One can see that the pad has two parents that are its sketch and a user parameter. The modification of one of them will result in an update of the pad. Thereafter the children of the pad have to be updated too.



(a) CATIA V5's specification tree of the Vertical Blade part, presenting its aggregated elements.



(b) Parent-child view for Pad.1 in CATIA V5. Parents are on the left, children on the right of Pad.1.

Figure 6.3 – CATIA V5 aggregation and parent-child relations.

6.3.2 Multi-Model Links

The term Multi-Model Link (MML) is specific to CATIA V5. A MML represents a relation or a link between two documents, which represents a dependency. The dependency can apply on many objects like geometry, parameters or documents. The design when using elements from another document in the scope of an assembly is referred to as contextual design or design in context. The use of a model in an assembly creates what is called an "instance" in CATIA V5, which is different of the template instance concept. A document can thus have several instances in an assembly. For example, a screw could be defined in one document and have ten instances in an assembly.

In CATIA V5 all links are directed and unidirectional. The source of the link is the dependent document and the target the dependency. As the links are unidirectional, the target document, in which the dependency is located, is not aware of the existence of the link.

The precision "Multi-Model" has been added because in the previous version of CATIA (*i.e.*, CATIA V4) there is only one document type. In CATIA V5 the design can be decomposed within various documents, each of them having its purpose.

Table 6.1 – CATIA V5 Multi-Model Links.

MML name	Document type		Description
	Source	Target	
Context	CATPart	CATProduct	This link designates the context element for the design in context. There can be only one context link going out from a part.
Import	CATPart	CATPart	Link that references a geometrical element located in another document. The creation of this link implies the creation of a context link if no context is currently defined.
KWE_CONTEXTUAL	CATPart or CATProduct	CATPart or CATProduct	Like an import link but addresses a published parameter in the current context.
KWE_REFERENCE	CATPart or CATProduct	CATPart or CATProduct	Created when a reference to an unpublished parameter is made, or when referencing a published parameter out of the context.
Instance	CATProduct	CATPart or CATProduct	Link targeting a document included in the assembly. There is a link for each instance of the document in the assembly.
CCP	CATPart	CATPart	This kind of link is created when some geometry is copied with link, out of the scope of a context.
ViewLink	CATDrawing	CATPart or CATProduct	Link from a view in a drawing to the document containing the drawn element.
Attribute	CATDrawing	CATPart or CATProduct or CATDrawing	Link a text contained in a view of a CATDrawing to the described geometry in the 3D model. It can also link two text in two CATDrawing.
Feature component	catalog	CATPart or CATProduct	Reference a template defined with the specific workbench (feature of document) or a RuleBase, which is a set of rules.
File component	catalog	CATPart or CATProduct	Reference a document from the catalogue.
Sub-Catalog	catalog	catalog	Created when a catalogue refers another catalogue or a part of it.
Validation	CATPart	CATPart	Link from a RuleBase, created with link from a catalogue, toward the document containing the original RuleBase.

Continued on next page...

Continued from previous page

	Document type		
MML name	Source	Target	Description
Material	CATPart	CATMaterial	Link to the material definition in the catalogue.
User Design Table	CATPart or CATProduct	external spreadsheet	Link to a spreadsheet containing a set of possible configurations for an object in a model.
Applicative Design Table	CATPart	external spreadsheet	This link is created when an UDF using a spreadsheet is instantiated. The link goes from the document wherein the UDF is instantiated to the original spreadsheet.
Document Template Link	CATProduct	CATPart or CATProduct	This link concerns Dassault document templates for product/assembly templates. It designates a document included in the template, from which a copy will be created (see figure 6.2(a)). However all documents from a document template will be "included" in the new context.
Doc	CATAnalysis	CATPart or CATAnalysis-Results or CATAnalysis-Computations	Link from an analysis to the documents analysed or resulting from the analysis.
Assembly Feature Input	CATProduct	CATPart	An assembly feature is a positioning feature within an assembly that allows to place models according to some layout/pattern defined in another document, for example, the RectPattern feature. A link is created from the document containing the assembly feature towards the document in which the pattern is defined.
Shape	CATProduct	Geometry document (V4 model, *.cgr...)	Link to the imported document.

Table 6.2 – CATIA V5 documents types.

File extension	Description
CATPart	Contains a model description composed of the geometry, KBE elements, etc.
CATProduct	Contains an assembly description. It can group other CATProducts but also CATParts and can contain KBE elements.
CATDrawing	Contains 2D views on a part or a product.
catalog	Allows to reference and organise parts, products, templates...
model	CATIA V4 document.
CATProcess	Corresponds to the machining workbench storage format for process description.
CATMaterial	Catalogue dedicated to material specifications.
CATAnalysis	Contains an analysis, such as a structure analysis.
CATAnalysisResults CATAnalysisComputations	Both are used to store the results of CATAnalysis.
CATScript	File containing a script that can be executed by CATIA V5.

6.3.2.1 CATIA V5 document types

Ten document types related to CATIA V5 have been identified. They are listed by their file extension in table 6.2.

6.3.2.2 Multi-Model Link types

In CATIA V5 there are 19 types of links that allow to create relations between documents. The list is available in table 6.1. The table presents the links' name as they are shown in CATIA V5, the type of the source and target document of the corresponding MML link, and finally a description.

It can be noticed that most of the links have at least a CATPart or a CATProduct as source or target. It seems obvious as the information, geometry as well as knowledge, is store in these resources. Only one link that concerns templates has been identified (*Document Template Link*), however it addresses only its definition. There is no link between templates and their instances.

Some classifications of MMLs have been proposed in the literature. However they do not address all the identified link types, because all the types may not have been identified or new links were introduced in more recent versions of CATIA V5. Brass [2005] classified MMLs according to three points of view: the structure, the assembly, and the document viewpoint. The structure point of view gathers links between documents involved in the product structure. It contains *instance* and *shape* links. It corresponds to what they called "instance-to-instance" MML because it is in the scope of a product context. The second is the assembly viewpoint that gathers links present in the scope of relational design, such as *import* and *KWE_CONTEXTUAL* links. They are referred to as "reference-to-instance" MML. Finally the document viewpoint gathers links between documents out of the scope of a product or external non-CAD documents, and is called "reference-to-reference." Guennuni [2008] reduced the classification to two types of links: the "reference-to-instance" and the "reference-to-reference" links because MMLs are dependencies between documents. The information provided by CATIA V5 about existing links indicate either an instance or a document as target. However, the source of any link is always a document because it is the document that contains the information about the links, not the instance.

Thus the proposed classification presented in table 6.3 summarises the definition of the literature merged with the realised analysis. The "reference-to-instance" category contains the links that refer to an instance in the information given by CATIA V5. The others are categorised as "reference-to-reference." The three viewpoints are those from Brass [2005]. This classification provides a first overview of the type of relations according to some of their specificities.

Table 6.3 – Multi-Model Link classification.

Link category	Viewpoint	MMLs
Reference-to-instance	Assembly	Import, KWE_CONTEXTUAL
Reference-to-reference	Structure	Instance, Shape, Sub-Catalog
	Document	Context, KWE_REFERENCE, CCP, ViewLink, Attribute, Feature component, File component, Validation, Material, User Design Table, Applicative Design Table, Document Template Link, Doc, Assembly Features Input

Table 6.4 – CATIA V5 link statuses.

Status	Description
Ok status	Link working and used, document found and data synchronised.
Ghost link	Link that remains in a document even if the linked data is not available any more. However there is not enough information to find the problem.
Synchronized	Reference is found and synchronized but the link is not used, *e.g.*, the source is disabled.
Not synchronized	An update of the document is needed because the target of the link has been modified.
Document not found	The target document is not found at the specified location given by the link.
Document not loaded	Document existing and found but not loaded in the current session.
Reference not found	The document is found but the referred element in the document of the link is not.
No status	No status can be given on the link. For example, it is the case when the context document is not loaded.

6.3.2.3 Link statuses

Links can be in several states called *status*. The status of a link depends on several aspects related to the target. For example, the target document or feature can not be found or the synchronisation following an update has not been done yet. The status of a link is available with the link information of a document, *i.e.*, in the "Edit/Links..." menu. Table 6.4 presents the various statuses that have been identified. The listing of link statuses provides an overview of some constraints that have to be handled when relations between documents are treated.

6.3.3 Influences on models update

6.3.3.1 Impact

The influence of each link on the update process has also been analysed. The objective is to have all documents up-to-date after a modification. As said previously, the links represent the relations between documents and thus can generate dependencies. The dependencies have to be resolved before propagating the update. This is well handled for the aggregation and parent-child relations within models. Here the interest is focused on the MMLs regarding the update propagation.

All MMLs were analysed in order to see the behaviour when making a modification at the source, then at the target of the link. Several behaviours can be pointed out. For links that target the content of the document, a change to this content implies an update of the link source in order to be up-to-date. This behaviour present an exception for *Doc* links as they also target the

result file from an analysis. Thus any modifications in the results have no impact on the analysis itself. When a link targets only the document, like for an *instance* link, the modification of the content does not impact the source. The update of the source becomes necessary when the target document is not available anymore (removed, renamed or moved to another location).

The influence of the Multi-Model Links in the update process does not present any unexpected behaviour. As said previously, the target of the link is not aware of the presence of a document referencing it. Thus only the source can be impacted by a change.

6.3.3.2 Cycles

In section 4.2.2 has been exposed that cycles are problematic with reference to dependencies and updates. Hence the CATIA V5 behaviour concerning the treatment of cycles has been investigated.

CATIA V5 prevents any cycle in the geometry by detecting cycles . A cycle would cause the geometry of a model to be impossible to compute because the update will not be able to determine from which element to start. This behaviour is located at the document level, which means that even if there are no dependencies between the geometry, the import link creation is forbidden between documents when generating a cycle.

However, CATIA V5 allows cycles between parameters and formulas. Three parts were created in an assembly with a parameter p_x in each of them, where x is the number of the part. Then a cycle has been created between these parameters *via* formulas, in order to make them all equal, $p_1 = p_2 = p_3$. Now all parameters share the same value and it is impossible to modify any of them because they are fixed with the formula.

This behaviour does not present any problems so it was modified in order to create an evolving cycle in order to test CATIA V5. One millimetre was added in the formula of part 3 such as $p_3 = p_2 + 1\text{mm}$. Once the formula is validated, the value of p_3 is updated. Then the first part has to be updated because $p_1 = p_3$. Two behaviours have been noticed from CATIA V5 in this situation depending on how the updates of the parts in the assembly is realised. The first approach consists is asking CATIA V5 to update part 1, then part 2 as the value of p_1 has changed and $p_2 = p_1$ and so on with no end. The second one is to update the assembly containing the 3 parts and see how CATIA V5 handles the cycle. Starting from a value $p_1 = p_2 = p_3 = 10\text{mm}$ then adding 1 millimetre as describe above, the following result is obtained: $p_1 = p_2 = 4\text{mm}$ and $p_3 = 5\text{mm}$ and the first document shows a sign telling that it is not up-to-date. When updating the assembly one more time, the value of all parameters is incremented by three, and the first document needs to be updated. From that can be deduced that CATIA V5 implements a cycle counter on the updates in order to prevent infinite loops, but it does not warn the user of the presence of the cycle between parameters.

There are two types of elements that can be imported from a CAD document by another: geometry and parameters. The behaviour of the system was analysed when faced to imports that create a cycle, because cycles cause major problems when dealing with updates. It has been noticed that, on the one hand, CATIA V5 prevents cycles between the geometry by blocking the link creation and warns the user. On the other hand, there is no such mechanism for parameter referencing. It is possible to create cycles between several parts on parameters with formulas. According to the resulting behaviour, CATIA V5 handles cycles with a counter, but it doesn't solve the problem. When the update stops, there is still at least one part that is not up-to-date. Furthermore the user is not warned of the presence of a cycle.

6.3.3.3 Links and templates

Templates and their instances have been studied regarding the links. First a template has no restriction concerning the link. All MMLs can be used in the design of the template or can reference it depending of the type of the document. Thus they are subject to the same update rules and procedures as with other documents. Then the fact that there is no link between a template and its instances has to be emphasised. As an instance is a standard document, there is

no solution provided by CATIA V5 to find the instances of a template. Hence there is no update procedure between a template and its instances: they have a separate life cycle.

6.4 CATIA V5 programming

Dassault Systèmes provides the facilities to program CATIA V5 with the help of two Application Programming Interfaces (API). With these APIs it is possible to access, modify, delete and create objects within CATIA V5.

6.4.1 Application Programming Interfaces

Two APIs are available: the automation API and the Component Application Architecture (CAA V5) API.

6.4.1.1 Component Application Architecture

CAA V5[1] is a framework part of the foundations of CATIA V5. It allows to access and to extend CATIA objects, as well as to create user interfaces within CATIA V5. This API is accessible *via* C++ and Java languages. CAA allows to extends the functionalities of CATIA V5 and also to create batch applications that do not need a CATIA V5 interactive session. However a specific licence is required to use CAA V5.

6.4.1.2 Automation

The automation API is a subset of the CAA V5 API. It also provides a higher level API composed of more complex functions. Hence it is lighter and simpler to use than the CAA API. The typical use of the automation API is the creation of scripts. Scripts can be created within CATScript or external applications with the Visual Basic language.

6.4.1.3 Comparison

CAA V5 is the most powerful API available for CATIA V5. It is also the most complex API to develop with. It requires advanced programming skills. CAA suits well for complex applications that require a comprehensive control of CATIA.

The automation API has been designed to easily and quickly develop small applications. Hence it avoids the overhead of CAA when the task to fulfil is simple enough. This API allows to automate certain process quickly and efficiently.

In the scope of this work, the CAA V5 API is used in order to manipulate CATIA V5 document content because it provides a more comprehensive access to CATIA V5 objects than the automation API.

6.4.2 Limitations

As presented in the previous section, CAA V5 is the foundation API of CATIA V5. Nevertheless some restrictions are present and not all librairies of CATIA are accessible. In this section is outlined the current status regarding the main concerns that are the MMLs and the templates.

[1] `http://www.3ds.com/fr/products/caav5/welcome/`

Table 6.5 – Presentation of MMLs retrieval status *via* the CAA API.

MML	Retrieval status
Context	OK. Needs the PX1 licence.
Import	OK. Needs the PX1 licence. The target feature is not directly provided. However it can be found by analysing the specification tree.
KWE_CONTEXTUAL	Idem as Import.
KWE_REFERENCE	Can be retrieved if the source and target product are loaded in the session. Needs the PX1 licence.
Instance	OK.
CCP	Idem as KWE_REFERENCE
ViewLink	OK.
Attribute	Not found.
Feature component	OK.
File component	OK.
Sub-Catalog	OK.
Validation	Not found.
Material	Not found.
User Design Table	OK.
Applicative Design Table	OK. Needs PX1 licence.
Document Template Link	Not found
Doc	OK.
Assembly Feature Input	Not found.
Shape	OK.

6.4.2.1 Multi-Model Links

To be able to automatically retrieve the dependencies between documents, MMLs have to be accessed *via* the CAA API. There is no central API that gives access to links present in a document like provided by the interactive menu function "Edit/links..." from CATIA V5. Thus each link has been investigated on its own in order to retrieve them. However not all of them were found. The table 6.5 presents the results of this investigations.

The majority of the MMLs have been found. Without solution are only three of them. An important issue that has to be raised is that in the available links, the *Import* and the *KWE_CONTEXTUAL* link can not be differentiated. Both are merged in the return value of the CAA function. Furthermore the *Document Template Link* has not be successfully retrieved. However this information is not crucial as it is only used by the instantiation process to determine if a document in the template has to be copied or just referred.

6.4.2.2 Templates recognition

The information that can be gathered about templates and their instances has also been analysed. Three aspects required to manage templates have been focused: (i) if a template can be identified, (ii) if an instance can be distinguished from a normal document, and (iii) if an instance can be matched with its template. Table 6.6 present the results of the investigation.

One can see that only the UDF fulfils the three aspects. For other template types, some workarounds had to be elaborated.

Table 6.6 – Template and instance related information retrieval status with CAA.

Template type	Template recognition	Instance recognition	Instance tracking
PowerCopy	Yes	No	No
User Defined Feature	Yes	Yes	Yes
Document template	Yes	No	No
Document as template	No	No	No

6.5 Chapter summary

6.5.1 Study results

During the study of the CATIA V5 system two ways to use templates were identified: the dedicated workbench of CATIA V5 and the use of a conventional CATIA V5 CATPart or CATProduct document as a template. The former provides a comprehensive wizard to defined and instantiate templates. It also allows the definition and the use of document templates as well as feature templates, which are small construction elements. The latter does not provide any support and thus results in more complicated tasks. For this reason, it is complex to use this approach to design feature templates. When instantiating a document *via* both approaches, the resulting instances can not be distinguished from each other because the result is the same. Direct and indirect instances were also defined. Direct instance are resulting from the instantiation process. Indirect instances are the result of the copy of a template instance contained within another template.

The parametric and associative design within CATIA V5 results in the creation of relations between objects. The are the aggregations and parent-child relations inside a document and Multi-Model Links between documents. These relations imply dependencies between the objects and thus are a main concern for the update process. The propagation of the modifications is accomplished on the basis of relations. Nineteen types of MMLs were identified. There is only one link that concerns templates: the *Document Template Link*. It is involved in the definition of document templates. So there is no link between a template and its instances. The behaviour of CATIA V5 when faced to link cycles has also been tested. CATIA V5 prevents any cycle in the geometry. However it is possible to create cycles between parameters. CATIA V5 avoids infinite computation by detecting the cycles and limiting the number of iteration to three. However the whole model can not be updated as there will always be an element not up-to-date.

CATIA V5 provides two APIs to enable the automation *via* programming: the automation API and the CAA V5 API. The former enables the fast development of scripts whereas the latter it more powerful and complete but also more complex. The CAA API can be used to retrieve a majority of the identified link types. Then the investigations addressed the identification of templates, their instances and if there is a way to make the link between a template and its instances. Only the UDF fulfils all of these three aspects, others are missing the instance identification and tracking.

6.5.2 Raised issues

In addition to the template issues, the limitations and the restrictions of the CAD system used to implement templates have also to be faced.

First there is no comprehensive instance tracking, *i.e.*, document template instances can not be identified and the relation between a template and its instances can not be made. For a document as template, there is even not a standardised definition for the template, in order to identify it.

Then there are the indirect instances that have to be handled in order to prevent redundant or non-necessary updates (see appendix C).

Finally possible loops have to be managed as CATIA V5 does not prevent them, except in the geometry (*Import* link). Furthermore the distinction between *Import* and *KWE_CONTEXTUAL* links can not be made, which can be problematic.

6.5.3 Conclusion

The objective of this chapter was to analyse a CAD system in order to provide concrete solutions to template issues that can be applied in real cases. The implementation of templates, the link types and the programming capabilities raise new issues. They have to be taken into account before designing the solution.

PART IV

Contributions

Chapter 7

Template Update Process

Contents

7.1 Introduction

This chapter provides an overview on the template update process defined in this work, and introduces the realised contributions, which is presented in the following chapters.

7.1.1 Addressed issues reminder

Template management is a huge problematic wherein it is very hard to address at the same time all its aspects comprehensively. In the scope of this work, two main issues that concern the update of knowledge templates were retained.

Knowledge templates may be complex applications that can involve several stakeholders and viewpoints. The concurrent aspect of today's product design can lead to possible conflicting argumentations. This is also the case in the scope of template updates, when a design solution has to be found to solve an issue. For this purpose the decision process is taken into account by the proposed approach.

The update of template instances is the second retained point. The modifications made to a template need to be reflected to its instances in order to avoid having to manage numerous desynchronised versions and fix possible bugs in instances. However when a template is used in several assemblies, the high number of template instances generates difficulties concerning their update. The order in which instances are updated impacts the result as well as the time spent on this task. For this reason, an update strategy has to be elaborated to enhance the instance updates.

7.1.2 Results of the state of the art analysis

Template management is a topic that has been little addressed in the past. However it is a brake to the adoption of template technologies at a large scale by the industries.

Lukibanov [2005] addressed template management and proposed the use of ontologies to represent templates and their interrelations. Information about the templates can then be visualised in order to validate the changes. Ontologies also allow to manage templates outside of the CAD system, in this case CATIA V5. He also proposed a business process for template update.

Regarding the decision process, the Issue-Based Information System framework has been selected because it presents interesting capabilities to address decision support. It allows a distributed and asynchronous decision making process between several stakeholders. Furthermore it can be used as a design rationale tool to store the argumentation that has led to a decision. However, despite its recognised advantages, design rationale tools have difficulties to be accepted by companies due to the difficulties to capture, represent and retrieve the knowledge efficiently.

Arndt [2007] proposed an ontology-based method for decision support with an application to template management. His method allows to document the link flow from templates in order to facilitate their use by engineers.

7.1.3 Approach

In order to solve the presented problems, theoretical aspects as well as concrete aspects through CATIA V5 concerning knowledge templates were analysed first. Then existing related works in the literature have been investigated. Based on these elements, a methodology has been designed that embraces the template update and provides an approach to enhance the decision process and the template instances update tasks. The methodology comes along with corresponding tools to apply it.

The proposed methodology aims at supporting template maintainers in their task. The core elements of the methodology is the template update process that provides the guidelines to efficiently approach the template update issues.

7.2 Template update process

7.2.1 Presentation of the process

The defined template update process is presented in figure 7.1. It is composed of nine steps. Two of them are located upstream from the template update phase represented in ovale and correspond to the decision making process, the rest are downstream and address instances update. The defined process covers template update tasks from the definition of the issue or the requirement implying the template update, up to the update of the instances. The whole process can be divided into three main parts identified (a), (b) and (c) in the figure 7.1. These parts will be described in the following sections.

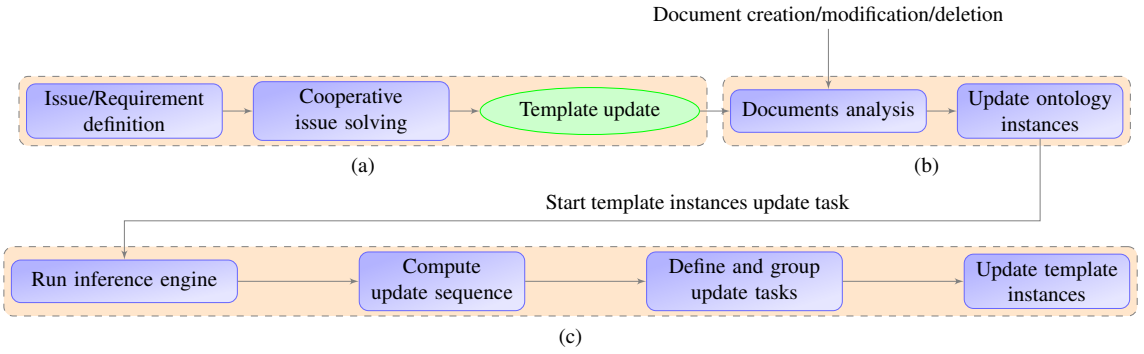


Figure 7.1 – Proposed global process for template update.

7.2.2 Collaborative issue solving

This section presents the part (a) from figure 7.1. This part is composed of three steps that are the definition of the issue or requirement, the issue solving, and the update of the template. The aim of this part is first to define the issue or requirement that would require a modification in a template. Then a discussion starts between involved persons in order to find the best solution to the defined issue. In this proposal, this is achieved by using a customised Issue-Based Information System. The solution resulting from the consensus between stakeholders can then be implemented. No tools or recommendations concerning the modification of the template are provided. The proposed approach is meant to interfere as few as possible with the existing company procedures. The details of this part are given in chapter 8.

7.2.3 Document analysis and ontological representation

This section presents the part (b) from figure 7.1. This part is composed of two steps: the analysis of documents and the update of the instances present in an ontology that has been defined. The documents that have to be analysed are the updated templates, but also other documents like CAD models. In CATIA V5, CATProducts are the “master of the information” [Lukibanov, 2005]. The objective is to extract and gather information from documents in order to have an up-to-date computer representation of them and also the relations between them. The data extracted from the documents is used to instantiate an ontology. The ontology provides a formal and computer understandable representation of the domain. Furthermore it allows reasoning in order to infer new knowledge. With these information a sequence can be computed, representing the order in which template instances have to be updated. The details of this part are given in chapter 9.

7.2.4 Update sequence computation

This section presents the part (c) from figure 7.1. It is composed of four steps: the use of an inference engine on the ontology, the computation of an update sequence, the definition of update tasks and finally the update of the template instances. This part of the process is triggered when the instances have to be updated, according to company criteria. The use of an inference engine on the ontology allows to classify concepts, instances and relations. This allows to deduce implicit knowledge according to the domain definition. All this knowledge, explicit and implicit, about the documents is then used by an algorithm to compute an update sequence. This update sequence is designated to support engineers in the template update task by providing the order in which the updates have to be processed in order to avoid problems like redundant updates or missing an update. The updates are then grouped into tasks in order to enhance the check-out/in of models from the Product Data Management system. Finally, the update can be carried out following the given sequence. The details of this part are given in chapter 10.

7.2.5 Comparison with the existing update approaches

Lukibanov [2005] proposed a process to update templates, which is depicted in figure 5.4 page 67. Its first step is the modification of templates. Then the dependencies in the ontology are checked and the dependent templates are then updated. Once the update is performed, the template ontology is updated with the modification made in the templates.

If a comparison is made between the presented process with the update process from Lukibanov [2005], many differences can be noticed. First a broader view on the template update process is proposed as the decision making is addressed in addition. Second the ontology takes advantage of an inference engine in order to deduce new knowledge, which allows to address easier template instances. Finally an update sequence can be generated to support engineers and thus reduce the time needed for the whole process.

7.3 Chapter summary

The purpose of this chapter was to introduce the proposed approach. The update templates process is presented. The process acts upstream and downstream of the template update. It is designed to infer as little as possible with the existing procedures within companies.

The process is composed of three main parts. The first part, which is located upstream of the template update, concerns the search of solutions to a template issue, within a collaborative environment. This part is the topic of chapter 8. In the presented process, an ontology is maintained to represent templates, documents, and their relations. This corresponds to the second part and is presented in chapter 9. The last part addresses the update of the template instances. In this part, a sequence to follow in order to efficiently apply the update is built. Chapter 10 details this part.

Chapter 8

Collaborative Template Issues Solving

Contents

8.1 Introduction

This chapter presents the first part of the template update methodology (see process depicted in figure 7.1). Templates are updated in order to solve issues in their design or behaviour, or to address new requirements. This implies to find a feasible solution for the evolution of the template. However templates are designed in a collaborative context and thus involve several engineers with different expertises. These engineers will also be involved during the decision making process that leads to the selection of a solution for the template update. The objective is to support the cooperative decision making process regarding template update in order to reduce its duration while enhancing the collaboration and solution quality.

8.2 Definition of the need

Bugs as well as new requirements are the two major reasons for updating templates. Their definition is an important step as the decision process will be based on this definition. However the definition may evolve during time by the arrival of new information.

In all cases the definition has to clear and easily understandable in order to thereafter focus on its solving. Once the definition of the need is made, it can be written into the decision support system in order to start looking for solutions.

8.3 Decision support system

8.3.1 Selection of the framework

Decision support systems have been investigated in chapter 3. The IBIS framework aims at supporting the solving of "wicked" problems. This type of problem is defined in section 3.2.2. According to the definition of "wicked" problems, it can be stated that the solving of template issues can be considered as a "wicked" problem because it matches the characteristics described by Rittel and Webber [1973]. For this reason, the Issue-Based Information System (IBIS) has been selected in the scope of this work. The IBIS framework provides the necessary concepts to raise issues and to allow stakeholders to propose their positions and opinions. In this way the discussions to find solutions to a problem are structured, which supports a clear communication. Furthermore, the framework also endorses the role of a design rationale solution by storing the argumentation and the choices that have led to a design solution (see section 3.2.3).

The IBIS framework is a generic framework and thus is not dependent on a specific implementation. Hence a Web Ontology Language (OWL) ontology has been used to represent the IBIS concepts. An IBIS concept is represented by an OWL concept. The instances of defined OWL concepts will be created by the discussion's participants. A major criteria for the selection of an ontology to represent the IBIS concepts are the capabilities to easily extend it with new concepts and to integrate it with other ontologies. For example, at Daimler AG, several ontologies representing car sub-subsystems have been developed, which were consolidated into a master vehicle ontology [Lukibanov, 2005]. The aim is use and link the extended IBIS ontology with the ontology defined in the next chapter, which contains concepts related to CAD and KBE.

In section 3.2.3.2 some limitations of design rationale and indirectly the IBIS framework have been presented. However only the issue related to the capture of the rationale will be addressed in the scope of this work. It would be a complex research work of its own to address and propose solutions for all of them. The IBIS framework is rather extended to make it more specific to the problem while trying to facilitate its adoption by users. For this purpose the IBIS framework has been extended with new concepts related to the problem in order to store information specific to the template update.

8.3.2 Extension of the IBIS

8.3.2.1 New concepts

Figure 8.1 presents the proposed extended IBIS model, which was defined in the scope of this work. The core elements from the original IBIS framework are represented by dashed boxes. The figure focuses on new concepts and relations for more clarity (see figure 3.3 on page 50 for the original structure of the IBIS framework). These elements allow stakeholders to propose and debate about a template related issue or a new requirement. This model was extended to focus on the problematic.

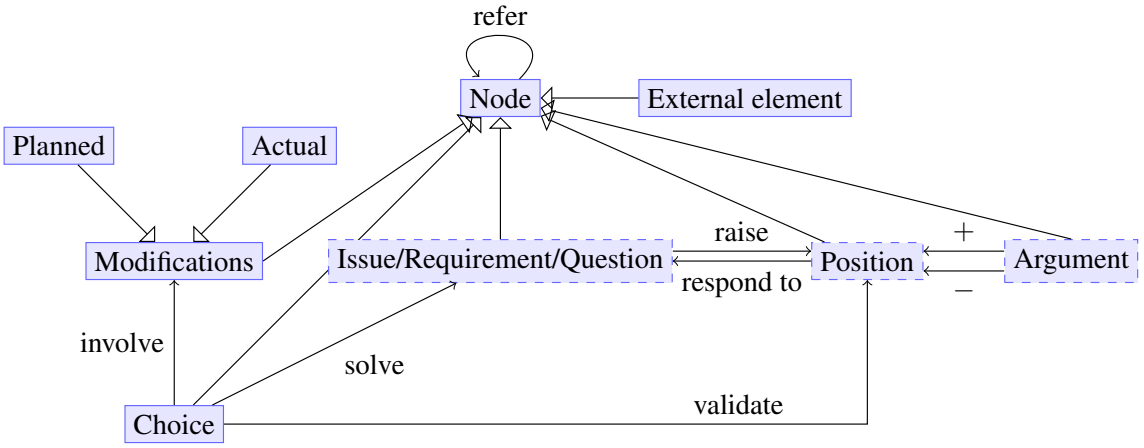


Figure 8.1 – IBIS-based model for template update support.

First the subsumption capabilities of ontologies have been used to define a parent concept called *Node*, which is an abstract concept, *i.e.*, not meant to be instantiated. A relation has been defined so references can be created from one node to another. In this way existing issues, ideas or arguments can be referenced to support a discussion. It is a first step to reuse past experiences explicitly and take advantage of design rationale. References can also be made on a new concept called *external element*. It is a generic concept that represents what is not explicitly defined in the ontology, such as documents that are an attachment for a given argument.

The second main addition concerns the choice of a solution and its impacts. A choice validates one or more positions. This results in modifications, either in the template or in a related document or context. After the validation, the planned modifications can be entered into the system in order to guide the template maintainer by providing, for example, a summary. With the help of these information, the template update can be accomplished without necessarily browsing the decision making process. When the update is over, the actually applied changes should also be stored for documentation purposes. A documentation of the encountered difficulties would be useful for further modifications (lessons learned). The actual modifications are also intended to support engineers in charge of modifying the template instances. Otherwise, without documentation, they would have to analyse the modified template, which would require additional time. Again, being able to reference external elements can be useful in this case.

8.3.2.2 Benefits

The benefits of the use of a customised IBIS-based system for the template update decision support are numerous.

First an IBIS provides an argumentation tool for distributed and asynchronous collaboration on decision making, as classified by the Computer Supported Concurrent Work matrix presented in table 3.1 page 46. Engineers are provided with an adequate tool to work together in order to rapidly solve template issues and find a consensus about solutions. As the decision process is structured, the understanding of the various propositions and debates is facilitated.

The IBIS is also a design rationale tool. Thus the decision process is documented and stored so that future issue solving can rely on past experiences. The reuse of past experiences is a key factor to avoid mistakes and speed up the process.

The IBIS framework is self sufficient to support the decision process. However, in order to extend its functionalities, concepts to support and to document the modifications have been added to the system. The update of template instances is enhanced by adding a description of the planned modifications resulting from the reached consensus. Engineers in charge of the instance updates would be able to quickly start the update without having to analyse the decision process in order to extract the relevant information. So it endorses also the role of a communication solution. Then, after the modifications are made, they can be documented in the system. In this way the changes are linked with the reasons and the arguments that have led to them. So modifications applied to a template are justified.

Another benefit comes from the references to external documents, instances or concepts from another ontology. The references allow the decision process to remain close to the actual facts by referencing actual objects. So the questions, positions and arguments can be supported by facts or resources.

The choice of OWL as representation language will allow further enhancements by using a reasoner to search into the database as well as to classify the various instances. However this is not proposed in this work but is presented as a perspective.

8.4 Template update

The template update is the step around which this work is articulated. However few interactions are realised with this task in order to avoid having to modify existing processes or workflows within companies.

The proposed approach aims at giving necessary descriptions to the template update phase as an input. These description are resulting from the collaborative decision and will guide the update task. Then company's processes are used for the template update. Once the update task is completed, the changes have to be reported in the system. This is meant to document the template evolution, as well as to provide information for the people who will update the instances.

To summarise, the information contained in the system about an issue would be the issue definition, its resolution process, the retained solution, and the planned and actual changes. Then possible remarks or new issues resulting from the actual modifications can be added and the solving process continues.

8.5 Chapter summary

In this chapter, an approach to support the decision process with the purpose of finding solutions to template issues was presented. An Issue-Based Information System was used and extended to allow template maintainers and engineers to collaborate in a distributed and asynchronous way. It provides a structure to the dialog and thus eases and accelerates the decision process.

The framework has been extended with concepts related to the template update problematic. However no design domain specific concepts were proposed in order to provide a high degree of freedom to the companies. Thus they can use their own ontologies. By integrating them within the decision process, the adoption of the tool by the users should be easier as the functionalities are present in the same location and linked together. In this way the design rationale aspects of the template update would be integrated into the system without requiring an additional task. The design rationale stored by the Issue-Based Information System combined with the description of the modifications is used to document the template evolution.

The concepts of the framework are represented within an ontology. It can also be further extended in order to cover new needs and to facilitate its integration with other ontologies. It also present interesting reasoning capabilities for future work.

Chapter 9

Ontology Definition for Knowledge Representation

Contents

9.1 Introduction

In the previous chapter a decision support system for collaborative template issues solving was presented. A short process composed of three steps was also introduced, from which the last step is the update of the concerned template. Now the focus is put towards the updates of the instances of modified templates in order to make them benefit of the latest modifications brought to their corresponding template. The objective is to provide engineers in charge of the update of template instances with a strategy for the update task, with the aim to facilitate and speed up this task. To be able to define this strategy, an overview of the current status of the documents and their relations is needed. Based on this overview an update sequence can be automatically computed to guide the engineers.

In order to be able to support the instances update, several information are needed, *i.e.*, a computer understandable representation of templates, CAD models and their relations is required. It will result in an overview that can be used by the engineers and processed by computers. The use of such an overview is more convenient and faster than working directly with the documents. For this purpose the templates and the CAD models have to be analysed in order to extract the raw data. Raw data only represent explicit knowledge. Parts of the knowledge, called implicit knowledge, are not directly available for computer use and must thus be inferred.

An ontology has been defined in order to represent a part of the knowledge template domain, with the purpose to share its definition and support the template instances update problem solving. The choice of an ontology has been made because it allows a formal representation of concepts, which is well suited for computer processing such as for reasoning. The reasoning mechanisms are able to infer implicit knowledge from the domain definition. Implicit knowledge could not be extracted directly from documents' raw data. For example during the analysis of CATIA V5, some lacks were identified because of missing or non-accessible information like template instances as presented in table 6.6 page 85. When all the raw data extracted from the documents is combined and processed by ontology inference capabilities, it gets enriched and becomes knowledge about the relationships.

The hierarchical nature of ontologies provides interesting characteristics for abstraction and specialisation of concepts. Thus generic concepts can be defined that abstract CAD system specific concepts. Instances of the CAD system specific concepts can thereafter be classified within these generic concepts. In this way, data from heterogeneous systems can be integrated and manipulated *via* the generic concepts.

Another advantage of using ontologies is that they can be managed independently from any application. This reason has also been pointed out by Lukibanov [2005] for his modelling choice in his approach for template update, which is presented in section 5.3.2. Ontologies are extensible and scalable and can grow independently from an application [Andersen and Vasilakis, 2007]. Thus if new needs or requirements are discovered, they can be integrated into the ontology without interfering with existing applications. The extensibility of ontologies is a key property to facilitate the reuse/sharing of an ontology between applications.

According to the classification of ontologies available in section 2.2.1.3, the designed ontology can be classified as an application ontology. It is designed for one specific application and in order to address a specific problem. Thus it contains concepts that are little purpose to be used within other contexts or domains. Nevertheless the ontology includes some concepts from the CAD and KBE domain. Thus the ontology could be extended to address both of these domains and then could be reused directly by other applications related to these domains.

The design of an ontology is a complex process that also impacts the application using the ontology. For this purpose a methodology, from the ones listed in section 2.2.2, has been followed in order to successfully achieve this crucial task.

9.2 Design of the ontology

9.2.1 Design approach

Andersen and Vasilakis [2007] stated that "it requires very good knowledge of the domain to be modelled, the ability to conceptualize the underlying ideas and a good knowledge of the syntax of the ontology language" in order to correctly express the knowledge model. Furthermore to design an ontology, the use of an adequate ontology engineering methodology is recommended. In this way the design can rely on researches and experiences in the ontology domain.

The Ontology Development 101 methodology is defined by their creators as a simple knowledge engineering methodology. The engineering method includes the following seven steps:

1. Define the domain and the scope of the ontology.
2. Consider reusing exiting ontologies.
3. Enumerate important terms in the ontology.
4. Define the classes and the class hierarchy.
5. Define the properties of classes (slots).
6. Define the facets of the slots.
7. Create instances.

It is an iterative methodology. The ontology has to be maintained and evolves depending on new needs, better understanding of the domain or with the feedback of its use. So these steps will be repeated, in order to extend, revise or enhance the ontology according to the new information.

In section 2.2.2 methodologies to design ontologies have been presented. A quick comparison between these methodologies based on the reviews in the literature is presented in section 2.2.2.3. From these information, the Ontology Development 101 methodology from Noy and McGuinness [2001] has been chosen for the reasons named below.

This methodology was chosen because of its lightness and the simplicity to use it. It is also independent from the representation language so it provides the liberty to choose the adequate representation language according to the needs or experience. The Ontology Development 101 was also ranked first in the methodology comparison from Hakkarainen *et al.* [2005] and that has also consolidated the choice of this methodology.

In order to know the representation and reasoning capabilities of the language and thus to correctly express the model, the representation language for the ontology has been chosen before starting the design of the ontology [Andersen and Vasilakis, 2007]. Bylander and Chandrasekaran [1988] stated that "representing knowledge for the purpose of solving some problem is strongly affected by the nature of the problem and the inference strategy to be applied to the problem." The Web Ontology Language (OWL), which is presented in section 2.3.3, has been selected. OWL is a standard, which is still evolving to provide better reasoning capabilities. Within all OWL version, the OWL Lite and OWL DL are interesting because they guarantee computational completeness, which is not the case of OWL Full. They are also based on a subset of Description Logics and many tools are available to work with OWL. The latest version is OWL 2. However OWL 2 was not standardised when the design of the ontology began, thus it was not in competition for the representation language selection.

So the Ontology Development 101 methodology [Noy and McGuinness, 2001] has been used to design the OWL ontology used in the scope of this work. It is composed of seven iterative steps, with each of them addressing a particular aspect of the ontology. The following section will go by all the seven steps in order to present the process that has led to the ontology that is used to compute update sequences.

9.2.2 Definition of the domain and scope

The definition of the domain and the scope of the ontology is a crucial step as it narrows the knowledge to be represented to the essential. Otherwise the concepts defined in the ontology could

spread within domains. The analysis and the definition of these domains would require time, which would be lost in the case these concepts are not used.

According to the methodology, the definition of the scope and domain consists in answering the following questions. These questions are used to limit the scope of the model. Nevertheless the answers are subject to change during the evolution of the ontology.

What is the domain that the ontology will cover? The ontology is meant to represent KBE templates, template instances and information about their relationships in the context of CAD assemblies. However in the scope of this work, the KBE templates are limited to the product design domain.

For what are we going to use the ontology? The ontology will be used by computer systems to compute a sequence of updates for template instances. This sequence should support engineers in their template instances update task, after some modifications have been made to a template.

For what type of questions the information in the ontology should provide answers? These kinds of questions are also called "competency questions" [Grüninger and Fox, 1995]. Here is a non-comprehensive list of questions the ontology, used as a knowledge base, should be able to answer in order to help solving the studied problem.

- What documents refer and are referred by a given document?
- What are the instances of a given template and where are they located?
- What are the inputs and outputs of a given template?
- What is the type of a given relation between two documents?
- What are the type, location and name of a document?

Who will use and maintain the ontology? The ontology is not meant to be used by a human user directly. The objective is to provide a computer understandable and processable representation to a computer software in charge of calculating an update sequence, which will then be used by a human user. Thus computer scientists would be indirect users of the ontology, but direct users will be software or engineers if they need to retrieve specific information.

Its maintenance will require some knowledge about ontology design. Hence it can be achieved by design engineers with this competency, in order to extend the ontology with their domain or application expertise.

Summary Through these questions a preliminary draft of the ontology and its usage can be made. The first two questions pointed out the domain and scope of the ontology that are KBE templates and the creation of an update sequence for template instances update. The content of the ontology starts to be defined by giving answers to the competency questions. Finally the definition of the users and maintainers provides an idea on some indirect requirements to facilitate the manipulation of the ontology. In this case it has a weak impact as the persons accessing the content of the ontology, users and maintainers, are from the same domain.

9.2.3 Reuse of existing ontologies

Existing ontologies can be either reused or can be considered during the design of a new ontology. It allows to mainly save time even if a formalism translation is needed between the existing ontology and the formalism chosen for the new ontology.

In the scope of this work only one work dealing with KBE templates by using an ontology has been found. It has been undertaken by Lukibanov [2005] wherein an ontology is used for template management. This work is presented in section 5.3.2. Unfortunately no publication of

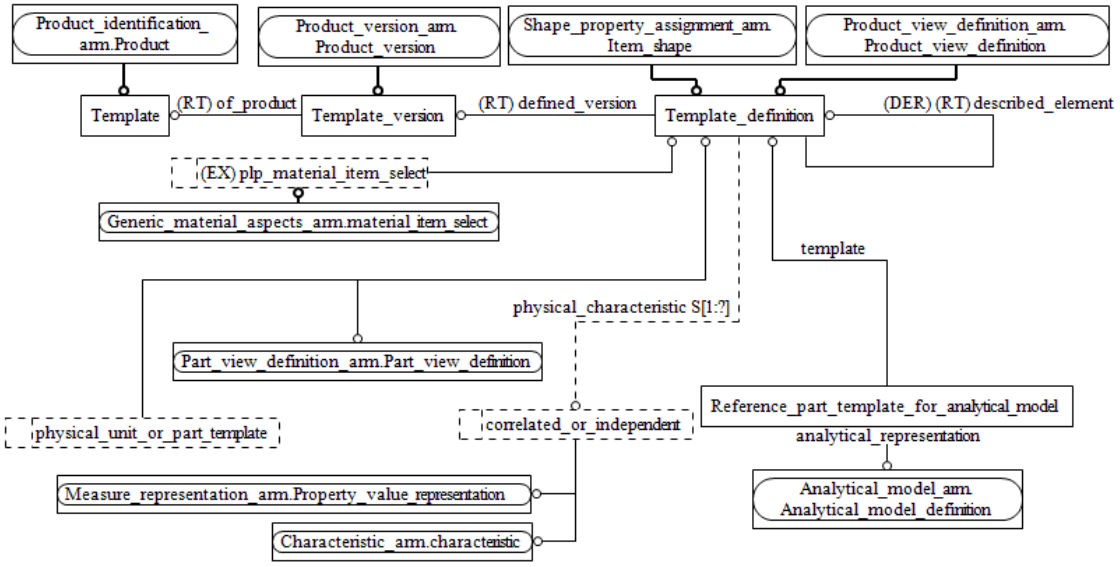


Figure 9.1 – EXPRESS-G schema representing the definition of a template in STEP.

the ontology has been available, certainly due to intellectual property issues as it was the result of industrial research work. Thus this work was complex to take into account due to the presented circumstances and thus few elements have been reused.

There are few defined ontologies that are available within the CAD and KBE domains. However many standards are available and can be used to define some parts of the ontology. One of the most famous and used standard in the CAD domain is the Standard for the Exchange of Product Model Data or STEP (see section 2.4). The focus has been put on this standard as it can be considered as the larger and most complete in this domain. The aim of STEP is to describe product data through the life cycle of a product and is used for product data exchange. However STEP does not address KBE or KBE templates. So it is not suitable to use STEP in the scope of this work. Nevertheless it can provide generic concepts about documents.

In the scope of this work, aspects related to documents and template identification were studied. For this purpose diverse STEP Applications Modules (AM) were investigated. The AM 1722, which is called "Part template," aims at the identification of templates, its versions and definitions. Figure 9.1 shows the entity level defined in this AM with an EXPRESS-G diagram. In the scope of this work the top part of the schema is particularly interesting. It can be seen that a template is a specialisation of a STEP product. Thus it has versions and definitions. These concepts will be reused for the definition of the ontology.

Other AMs are related to templates such as the 1717 or 1720. However they do not provide relevant concept definitions for this work.

9.2.4 Enumerate important terms

Noy and McGuinness [2001] advised that concepts in an ontology should be very close to the objects as an ontology is a model of the world, thus its concepts have to reflect the reality. For this reason CATIA V5 has been analysed in chapter 6 as well as the template concepts that are in the scope, in order to point out existing terms.

Terms that have been enumerated come from 4 locations: the original needs that have been defined to update template instances, the state-of-the-art about KBE templates, the CATIA V5 analysis, and the STEP standard. Collected terms are presented in the following non-comprehensive

list: (i) Already identified from the problem definition were terms like template, instance, dependence, relation, input/output or update. (ii) Concerning the literature, terms like maintenance, link, link management, assembly, or template type have been raised. (iii) CATIA V5's terms related to documents types, MML and templates are also present in the list. (iv) Finally from the STEP standard, terms to identify and version templates were included.

9.2.5 Define classes and hierarchy

According to the methodology the definition of the classes and the relations are two very close steps because users usually define relations after having defined few concepts. In order to provide a clear overview and to facilitate the understanding both steps have been treated separately.

Uschold and Gruninge [1996] identified three possibilities to develop a class hierarchy: the top down, the bottom up and a combination of both approaches. The top down approach consists in defining the more generic concepts first and then continuously refine them. The bottom up starts from the most specific classes that are then grouped together under more general concepts. The mixed approach consists in defining the most salient concepts and the generalising or specifying them appropriately.

The third approach has been used. The listed terms in the previous section have to be filtered in order to select the ones that will take place in the hierarchy. The construction of the hierarchy was articulated around the *Template* and *Instance* concepts because they are the centre of interest. At the same time foundation elements that are the CATIA V5 template concepts are present. The objective was to link them together by specialisation and generalisation.

The resulting taxonomy is presented in figure 9.2 on page 105. It shows the retained classes and their organisation.

9.2.6 Define classes relations and properties

9.2.6.1 Relations

Regarding the class relations (object properties), the focus has been put on the representation of relations between documents. Each link identified within CATIA V5 was added as a relation.

Then like for concepts, relations have been classified. Table 6.1 on page 78 presents the various links that have been identified within CATIA V5. As said previously each link does not have the same impact or behaviour. For this purpose all links that are involved in the information flow and the update propagation have been grouped under a parent relation called *DependenceLink*. In this way the CAD system links are abstracted and there is only one link that has to be managed for the dependencies.

In the scope of the addressed problem, a link has to be created in order to relate a template to its instances. To be able to do this, a relation called *InstanceLocation_link* has been added and will link a template definition to one of its corresponding instances.

More generally many links have been defined that are not available in CATIA V5. For instance in CATIA V5 a *Doc* link can create a relation between an analysis and the analysed document as well as a results file. Two new links have been introduced to distinguish them because the first one is the input of the analysis, and the second is the output. The objective is to infer these links by rules in order to add knowledge that is not explicitly available by the analysis of documents. In this way the algorithm will be able to take advantage of this enriched representation to compute update sequences.

Furthermore all inverse links were also defined in order to be able to navigate between documents in both directions. This is necessary because CATIA V5 MML are unidirectional.

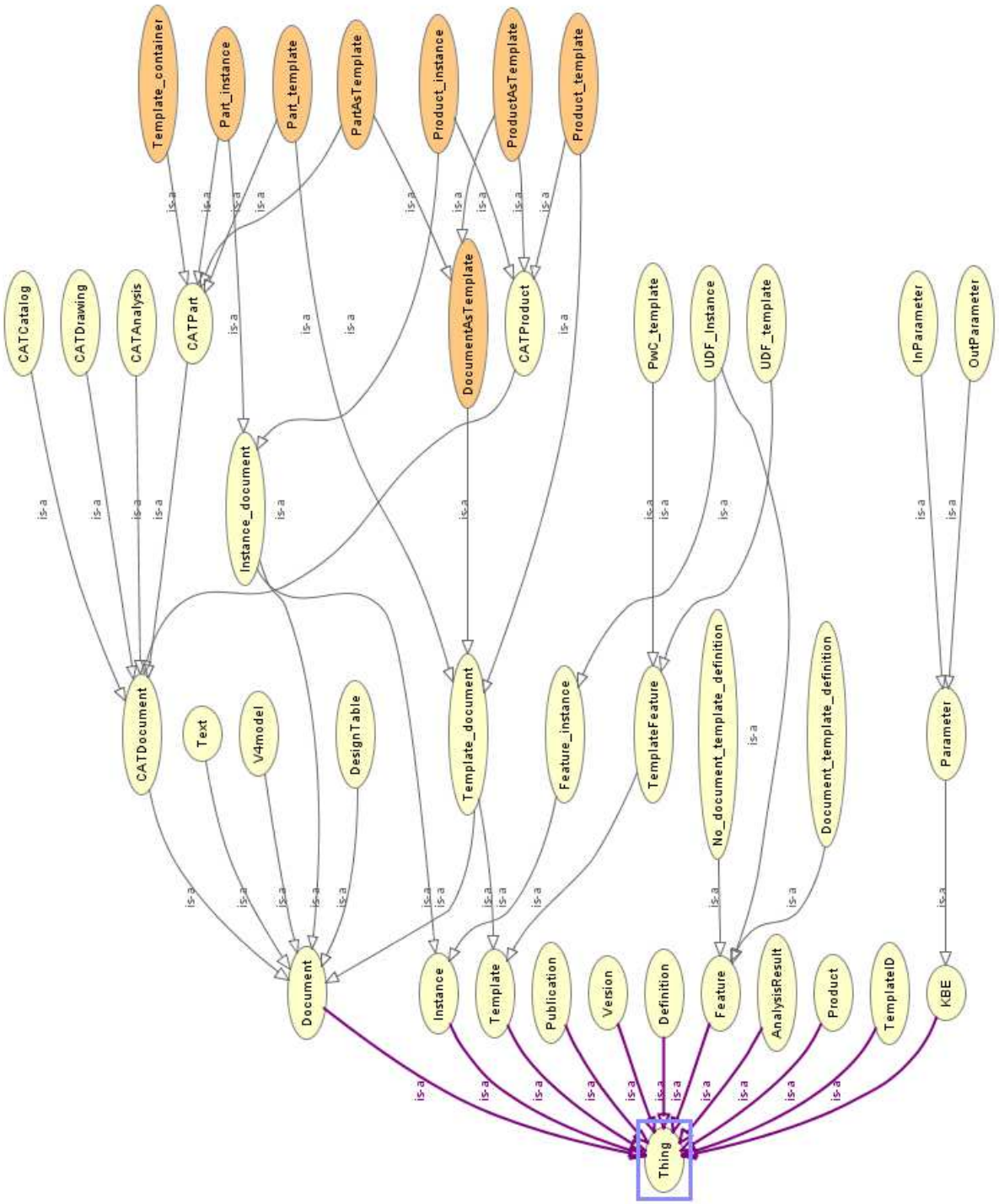


Figure 9.2 – Taxonomy of the designed ontology.

9.2.6.2 Data properties

Data properties can be considered as attributes of classes in object-oriented programming. They are used to describe concepts like the colour of a car. In the scope of this work they were mainly used to store document related attributes like the location or the name.

9.2.7 Define slots' facets

The facet of a slot is the type of value it can take. For instance a data property that represents the name of a document would be a string.

The facets are also used for object properties. An object property is a relation that has a source that is called the "domain" and a target that is the "range." The range and the domain are a class or a class expression, *e.g.*, the intersection or union of classes. The facets of the MML correspond to the classes representing the source and target documents defined in the table 6.1.

The definition of the facets is important when defining a domain. It can be used by reasoner engines to check the consistency of the definition as well as to classify instances.

9.2.8 Instantiation of the ontology

The instantiation of the ontology has been automated as it would be a huge and time consuming task for a human. Thus it has been entrusted to a computer system that is in charge of the analysis of the documents in order to extract the relevant information. Documents that are related to templates, CAD models or documents that are source or links are analysed. Then all found relations as well as the CAD system specific concepts are instantiated. It converts the raw data extracted from documents into instances of corresponding classes and relations. The other concepts, which are more generic, are meant to be inferred.

9.3 Presentation of the defined ontology

9.3.1 Concepts overview

The previous sections presented the approach followed to define the ontology. It was based on the Ontology Development 101 methodology.

The resulting ontology has been implemented with the Web Ontology Language. It is composed of three levels. Figure 9.3 illustrates these levels by showing a simplified extract of the ontology

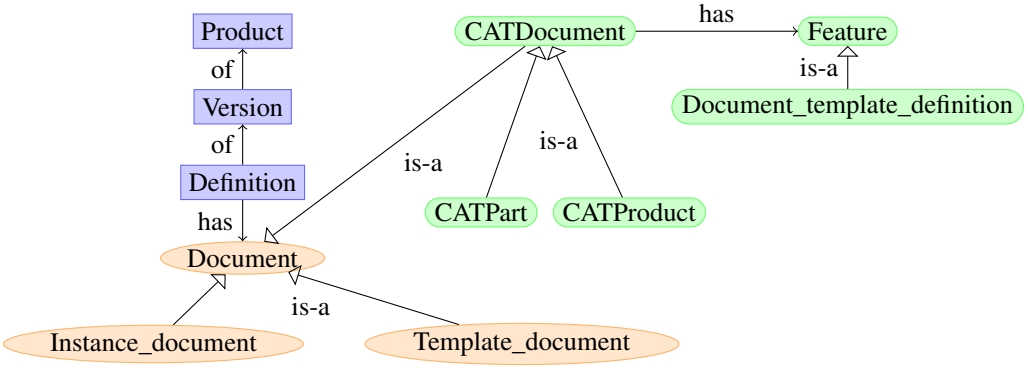


Figure 9.3 – Extract of the ontology with the abstraction level and the CAD system concepts (here CATIA V5).

for a better understanding. The rectangles (blue) represent the highest level, which contains the most generic concepts, here related to documents and were taken from the STEP standard. It can be seen that a product is composed of versions that have definitions like presented in figure 9.1. In the middle are the application related concepts, which are presented as ovales (orange). They represent concepts necessary to solve the problem and provide the CAD system abstraction. Finally the bottom level is the CAD system specific level, represented by rounded rectangles (green) in figure 9.3. Here concepts from CATIA V5 are represented. A *CATDocument* can contain features, such as a *Document_template_definition* feature. The concepts of the ontology correspond to those presented in the taxonomy available in figure 9.2.

In section 9.2.8 has been said that only the CAD system specific concepts are instantiated and thereafter the instance will be classified within the other concepts. Thus these other concepts need to be defined according to the CAD system specific concepts. For instance, the concept called *Template_document* has been defined (see figure 9.3). In CATIA V5 the definition of a template is realised by a feature located inside a document, even for document templates. However it would be more convenient to classify a document that contains such a definition as a *Template_document*, because this piece of information is not explicit without accessing the document content. So the *Template_document* concept has been defined as follows:

$$Template\_document \equiv (CATPart \cup CATProduct) \cap \exists has.Document\_template\_definition$$

Other concepts were defined the same way, like the *Template_container*, which is a document containing a template feature.

As said previously, there is no template instance concept in CATIA V5. However it is an important concept in the scope of the problem solving. Thus it has to be inferred. Instances have been defined as follows:

$$\begin{aligned} Instance\_document \equiv\ & (CATPart \cup CATProduct) \cap \exists has.No\_template\_definition \\ & \cap \exists hasAttribute.TemplateID \end{aligned}$$

This defined concept allows to classify an instance when it has a *TemplateID*, which is an unique identifier placed in the template and that will be copied during the instantiation, and it does not contain a template definition, in order to differentiate the instance from the template. In this defined concept can be noticed that the negation present in the definition given in the previous sentence is not present ("does *not* contain a template definition"). It has been expressed with the *No_template_definition* concept. This is due to the open-world assumption of ontologies, which was introduced in sections 2.3.3.4 and 2.3.3.6, wherein everything that is not explicitly stated is considered as unknown and thus nothing can be inferred.

9.3.2 Relations

Besides concepts, relations are the second core elements of an ontology. Figure 9.4 presents a part of the relations that have been defined. The relations have been classified and form a hierarchy. At the top of the figure is the *DependenceLink* that gathers the CAD system specific links implying a dependence between documents. This relation will be used by the algorithm in order to rank the documents. The *Contains* property is a transitive property that allows to know all elements contained in another, even recursively. Furthermore there are the links that are meant to represent CATIA V5 links but also more generally to represent a relation between two documents. The inverse links have been defined to enable the navigation in both direction between documents. Then the template related links were also grouped together.

Inference engines take the object properties into account. However relations can not be defined like concepts. Thus in order to enhance the inference possibility concerning relations, rules have to be defined.

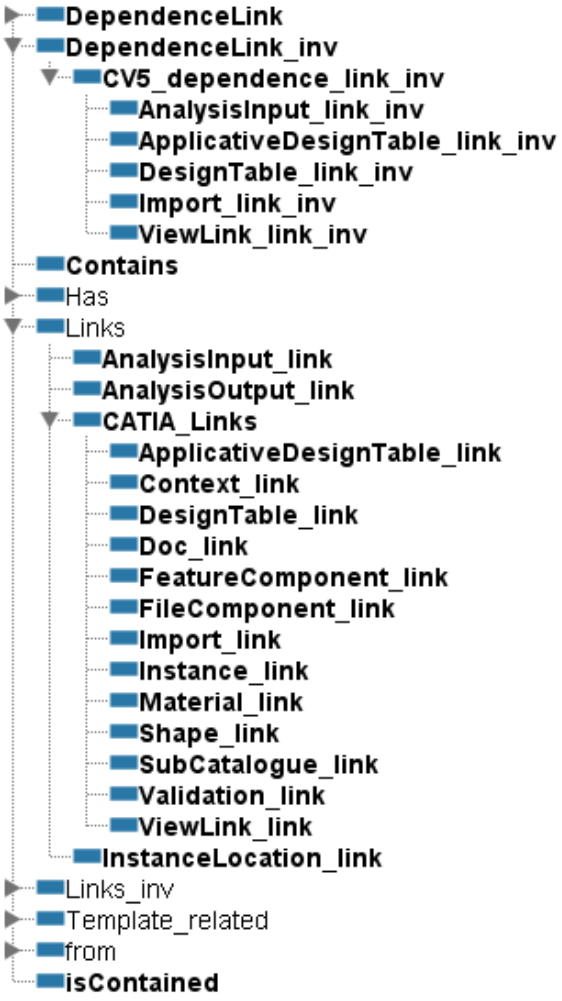


Figure 9.4 – Classification of the object properties within the ontology.

9.3.3 Rules

Rules have been added within the definition of the ontology in order to support advanced inferences with instances and relations. They are powerful tools that allow to increase the expressiveness of OWL. Rules enable to draw conclusions from given facts on the schema antecedent-consequent.

In section 2.3.3.6 the Semantic Web Rule Language has been presented. It allows to use rules in conjunction with OWL and thus allows to express more complex relations and definitions of concepts and relations by involving several instances in the reasoning.

SWRL rules are used to enrich relations. The relation between a template and its instances (*InstanceLocation_link*) is created between a template and its instances, which share the same ID as the template, via the following rule:

```
Template_document(?t) ∧ hasTemplateID(?t,?id) ∧ TemplateID(?id)
∧ Instance_document(?i) ∧ hasTemplateID(?i,?id)
    ⇒ InstanceLocation_link(?t, ?i)
```

It was said in section 9.2.6.1 that two new links were defined to distinguish the input from the output of a CATIA V5 analysis. This is necessary because a modification in the input of an analysis would require to redo the work in order to update the results. This is realised by the two following rules, based on the target document type:

```
CATAnalysis(?a) ∧ Doc_link(?a,?t) ∧ Text(?t)
    ⇒ AnalysisOutput_link(?a,?t) ∧ AnalysisResult(?t)

CATAnalysis(?a) ∧ Doc_link(?a,?t) ∧ CATPart(?t)
    ⇒ AnalysisInput_link(?a,?t) ∧ AnalysisInput(?t)
```

9.4 Chapter summary

This chapter introduced an ontology that has been defined with the purpose to represent and infer knowledge about documents. This knowledge will thereafter be used to support the generation of update sequences.

The selection of an ontology rather than another representation has been made due to the formal definition and reasoning capabilities provided by ontologies. Ontologies are used to represent a domain or a part of a domain in a formalised way. The Web Ontology Language has been chosen to develop the ontology. It is a W3C standard language for the ontology representation, which was developed for the Semantic Web. It provides a formal representation based on Description Logics and is supported by several reasoners. Furthermore the SWRL language can be used to define rules, which will be applied on the ontology by the reasoning engine.

OWL reasoning capabilities provide several benefits. First it allows to check the consistency of the definition of the represented domain. In this way the domain definition is guaranteed without contradictions. Ontologies can also be used to infer new knowledge. For instance reasoners are able to classify concepts but also instances of the ontology according to the domain definition.

In the instances update application, reasoning is used on the ontology in order to infer implicit knowledge. The implicit knowledge can not be directly extracted from the CAD documents. So reasoning allows to classify documents or to create new relations between documents. The document classification is also used to abstract the CAD system and generalise this approach.

A methodology has been followed in order to design the ontology. The different steps have been described as well as the undertaken actions and results. This has led to the definition of concepts, relations and also rules that define the ontology.

There is a major difference between databases and ontologies that has to be taken into account when defining an ontology. Ontologies are based on the open world assumption that means that something that is not present in the knowledge base is not considered as false (see section 2.3.3.4). This means that it has to be explicitly stated that an element does not exist. This was, for example, the case for the identification of template instances.

Table 6.6 presented the different template implementation types identified in CATIA V5 with their retrieval status. There were major issues concerning the identification and tracking of instances as well as the identification of documents used as templates without an explicit template definition from the dedicated CATIA V5 workbench. The concepts present in the ontology combined with reasoning and the addition of information in the CAD models allow to classify these documents and thus to solve these issues. Only the PowerCopies instances could not be addressed due to the complexity to identify their instances because there are features and geometry that can not be distinguished from the others. A solution would be to resort to pattern matching but this was not investigated in the scope of this work.

The ontology contains about 40 classes, more than 60 relations and 5 rules. The expressiveness of the designed ontology is equivalent to the $\mathcal{SHI}(D)$ Description Logic, which is a subset of OWL Lite. The functional properties have not been used. According to table 2.1 (page 41) the reasoning complexity time of the designed ontology is `ExpTime`.

The designed ontology is an application ontology, which has been developed for the specific purpose of solving the template instances update issues. However it can be reused to develop a visualisation tool, for example, the visualisation of all dependencies between documents. The resulting ontology is of a reasonable size and thus would be simple to maintain and to evolve for further applications.

According to Noy and McGuinness [2001] the assessment of the ontology quality is realised by using the ontology in the application it has been designed for. Next chapter will present the algorithm that uses the ontology in order to compute sequences to guide engineers during the template instances update task.

Chapter 10

Update Sequence Computation

Contents

10.1 Problem definition

10.1.1 Introduction

An ontology that represents the knowledge about documents has been presented in the previous chapter. It allows the inference of implicit knowledge from the explicit information extracted from CAD models and templates. Documents can thus be classified as well as relations between them. This is used as a solution to abstract the CAD system. New knowledge can also be discovered according to the definition of the domain present in the ontology.

In the scope of this work the issue of KBE template instances update is addressed. The update of instances is a complex and time consuming task as it requires a deep analysis of document relationships to be able to update needed documents as well as not forgetting any of them. The complexity of the network created by the documents and their relations make this task difficult for engineers without supporting tools. For this reason an approach to support this task is proposed. It is based on an algorithm that generates a sequence of documents that have to be processed in the scope of the update. The knowledge present within the ontology is used by the update sequence computation algorithm.

10.1.2 Objectives

The aim of the algorithm is to produce a sequence that avoids engineers the effort of finding which documents have to be updated. Within large CAD assemblies, the number of documents can reach several thousands. Furthermore templates can be used in an arbitrary number of assemblies. In a such context the update of template instances is a challenging task.

The resulting sequence should provide a strategy to guide engineers during the update by providing the order in which the documents have to be addressed in an automated way. The computation of the sequence has to take into account several aspects and issues. For example the direct and indirect instances presented in figure 6.1 (page 75), as well as to prevent redundant instantiations, overwriting by wrong replacement order or to update an instance that must not be updated because it is in a released status.

Template instances do not share the life cycle of their template as they become independent documents through the instantiation. Hence the life cycle of template instances has also to be taken into account during the sequence computation. Figure 10.1 presents the life cycle of a template instance as its has been defined in this work. The first step, “In context,” represents the status of an instance that has been put in the target context before the inputs are set. That means the instance is present but has not adapted itself to the context. The “Instantiated” status represents the instance adapted to the context and providing the functionality. It is the standard status of a template. The instance can also be modified and customised by an engineer for many purposes, such as to take into account specificities of the context. The figure 1.10(b) (page 18) illustrates such

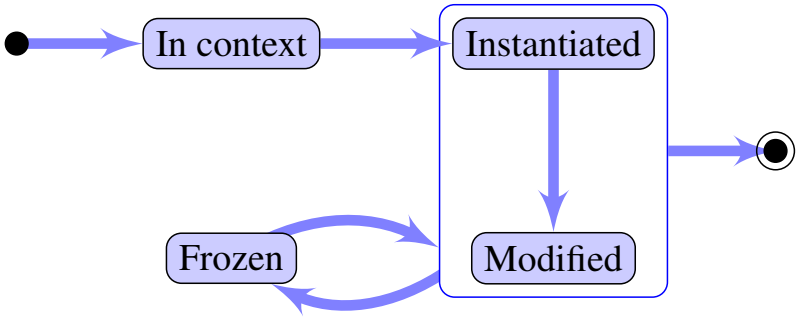


Figure 10.1 – Template instance status evolution during its life cycle.

a customisation wherein the instance on the right has been modified in order to avoid a collision with the metal sheet. When a product is ready for manufacturing, its geometrical representation must not be changed anymore. For this reason an instance can also be in a "Frozen" state, so it should not be updated if the template definition changes.

10.1.3 Problem representation

The template instances update issue has been studied as a dependency management problem (see section 4.3). Documents and relations form a graph that will be used as a data structure to solve the problem. Within this graph, a node represents a document and an edge represents a link between two documents. This is achieved through the ontology wherein the instances represent documents and object properties the links. The instances can be classified under several concepts, hence the nodes are typed.

The particularity of this approach is that nodes and edges are typed according to their classification made in the ontology by using inference engines. The algorithm will use generic concepts for the node and edge types in order to not be dependent on the CAD system.

In order to compute a sequence according to the graph and ontology knowledge, two approaches were tested based on dependency graphs, which have been presented in section 4.3.

10.2 Tested approaches

10.2.1 Introduction

In order to generate a sequence, the following principle has been applied: a template instance or document can only be updated if and only if all documents it depends on are up-to-date. To do that two algorithms were tested to generate what is called an update sequence: a topological sort and a ranking algorithm. Both tested approaches work on dependency graphs and construct sequences iteratively. However they have been adapted to take advantage of the knowledge about documents that is stored in the ontology.

Both algorithms are interesting because they can work locally. They do not need to have and work on the whole graph but rather explore it iteratively. This presents huge benefits when having several thousands nodes that are not all subject to be involved in the sequence construction.

In section 4.2.2, it has been said that cycles are problematic when working with dependencies. Thus a mechanism is necessary in order to prevent infinite loops. The detection of cycles in realised during the construction of the sequence. The following section will first present the algorithms used to compute update sequences, and thereafter the cycle handling approach. In this way the focus is put on the update sequence construction.

10.2.2 Topological sort

10.2.2.1 Concept and scope

The topological sort has been introduced in section 4.3.2. It is a basic approach for job scheduling. It allows to linearly order the jobs according to their dependencies by working on a directed acyclic graph. This algorithm seemed to be a good starting point to solve the update issues.

The algorithm was studied on a subset of the template instances update problem. The aim was to evaluate this approach on a smaller problem, which consists in the generation of an update sequence for update between documents, without templates. When a document in an assembly is modified, the changes may impact related documents, because of an external reference on a parameter or geometry. Hence the related documents will require an update to take the changes in the referred document into account.

10.2.2.2 Algorithm

The adapted topological sort algorithm is presented in algorithm 10.1. It creates a sequence of documents that have to be updated following some modifications in a document.

Algorithm 10.1: Adapted iterative DFS-based topological sort for document update scheduling.

Result: Linear sequence of documents presenting the order in which they have to be updated.

Input: A list $modifiedElements$ of initially modified documents.
Input: The OWL ontology $onto$.
Output: An ordered list of documents res.

```
1  begin
2      res ← ∅
3      dict ← {} ;                          // Dictionary mapping a value to a list
4      list ← modifiedElements ;                          // List used as a stack
5      while list ≠ ∅ do
6          elem ← ReadTail(list)
7          if not Contains(dict, elem) then
               // Get all dependent documents
8              children ← GetLinkTargets(onto, elem, DependenceLink_inv)
9          else
               // Retrieve the stored list of children
10             children ← GetList(dict, elem)
11         if children ≠ ∅ then
12             child ← ExtractTail(children)
13             if not Contains(res, child) then
14                 Append(child, list) ;        // Becomes the next element to be visited
15                 Update(dict, elem, children) ;          // Removes the child from the
                   dictionary
16         else
               // If all children have been visited, the current node can be added
                  to the sequence, and removed from the stack
17             Append(ExtractTail(list), res)
18     return res
19 end
```

The main differences between the proposed algorithm and the topological sort is that only a part of the graph is taken into account. Not the whole graph created by documents and their links is processed, only necessary elements based on the given modified documents, which are starting nodes. The topological sort only takes a graph as input. In the adapted version, the list of modified documents has been added to the inputs list. Hence the construction of the sequence starts from defined nodes. The algorithm works also well if the input documents are located in different connected components in the case of the graph is non-connected.

A topological sort can be obtained *via* different methods. The topological sort approach chosen in this investigation is using of a customised DFS. The approach from Kahn [1962], wherein the algorithm starts with the nodes without inbound edges, can not be applied because the algorithm does not work on the whole graph. Thus there may be inbound edges that come from any documents in the graph. Additional computation would be required in order to check if these inbound edges are impacting the computation or if they can be removed. The most convenient implementation of DFS is a recursive algorithm because it is easy to write and to understand. However in this case an iterative version has been developed. It uses a stack to store the nodes that have to be visited. This choice has been made to avoid performance issues [Lantzman, 2007]. The recursive version

would necessary be less competitive as an object-oriented programming language has been used to implement it. Hence the numerous calls to the recursive function may cause a call stack overflow due to the numerous nodes to visit in depth. Furthermore the computational performance of the iterative algorithm is better than the recursive one.

The post-order traversal needs to be used to create a topological sorted sequence. Thus documents are only added to the sequence once all their children have been visited. In order to fulfil this requirement a dictionary, which maps an element to another one, has been used to store remaining nodes' children that have to be visited. This would be implicitly achieved with the recursive version. The obtained sequence has to be inversed in order to obtain the update sequence that can be used. Like the DFS, this algorithm presents a linear complexity of $O(V + E)$, where V is then number of vertices and E then number of edges.

Another particularity is that the algorithm resorts to the ontology defined in the previous chapter (line 8). Hence a reference to the ontology has also been added to the inputs list. The ontology can be navigated or queried through the reasoner in order to formulate more complicated queries.

In this specific case, only the *DependenceLink_inv* are used. Thus only the documents that have links that are classified as *DependenceLink_inv* are added to the sequence. Here the classification capabilities of ontology with a reasoner are again useful in order to obtain inverse links as the links are unidirectional. Thanks to the ontology combined with a reasoner, this task is achieved before the sequence computation, thus the information can be directly used by the algorithm. It results in a simple and easy to maintain algorithm that can handle various types of documents.

10.2.2.3 Application and results

As said previously, several solutions can be valid topological sort results. It is also the case in this application because the ordering is based on the dependencies.

This approach has been tested on one assembly, which is the clamp presented in figure 1.10(a). The clamp is an assembly composed of several documents and has been design with CATIA V5. Figure 10.2 represents a dependency graph of the clamp assembly. In this graph there are three types of document: (i) assembly documents that are in green, (ii) part documents that are in orange, and (iii) spreadsheets in blue. The represented relations are *DependenceLink*. Thus the document at the origin of the link refers to an element in the target document or in the document itself.

Let us suppose that two documents were modified, the `Adapters` and the `Skeleton 11`. Hence these documents are the inputs of the algorithm with the ontology instantiated from the automated analysis of the assembly documents. Figure 10.3 presents the result obtained by using the algorithm 10.1. The solution is a sequence of five documents. Only the necessary documents are kept in the solution. This sequence has to be read starting from the vertex without inbound edges (`Skeleton 11`) and then by following the arrows.

10.2.2.4 Solution review

The obtained solution is not satisfying if compared with a solution given by an expert about the same problem (see figure 10.4).

In the solution provided by the expert two sequences are visible. These two sequences are independent from each other because they are based on the analysis of two non-connected components of the graph. Thus this solution is better as the two sequences could be processed concurrently and is easier to understand than the previous solution.

The solution to generate such a sequence would be to detect non-connected components containing an input document and apply the algorithm on them separately. However this approach also shows some limitations. For example if the `main skeleton` document is modified, the sequence presented in figure 10.5 would be obtained. As you can see there is only one sequence because it can not be decomposed into connected components. However this sequence, although

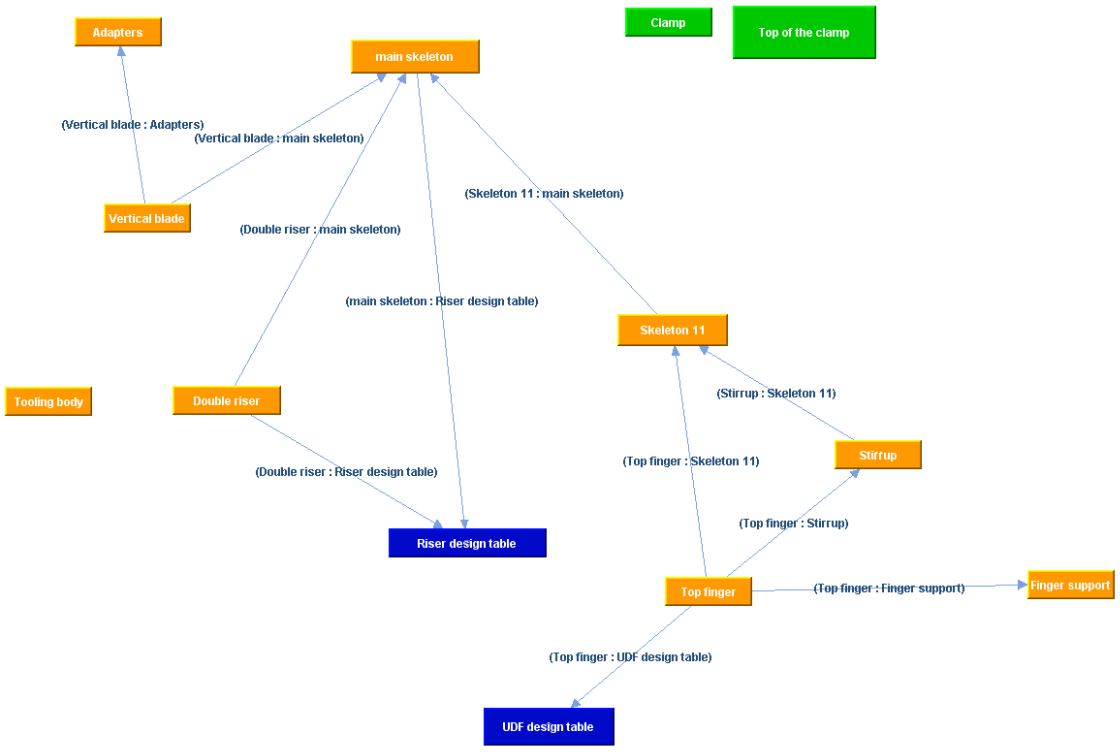


Figure 10.2 – Documents and their *DependenceLink* corresponding to the clamp assembly shown in figure 1.10(a). The CAD models were realised with CATIA V5. Green boxes are CATProducts, orange boxes represent CATParts and blue boxes represent external spreadsheets.

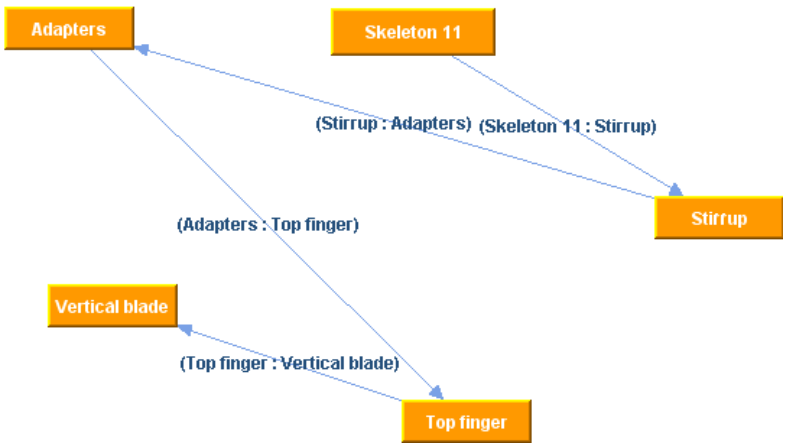


Figure 10.3 – Sequence resulting from the algorithm 10.1 on the graph presented in figure 10.2 with `Skeleton 11` and `Adapters` as modified documents.

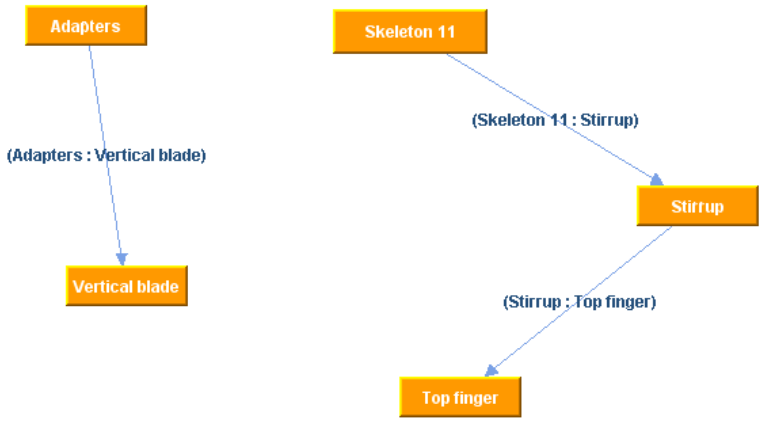


Figure 10.4 – Update solution provided by an expert.

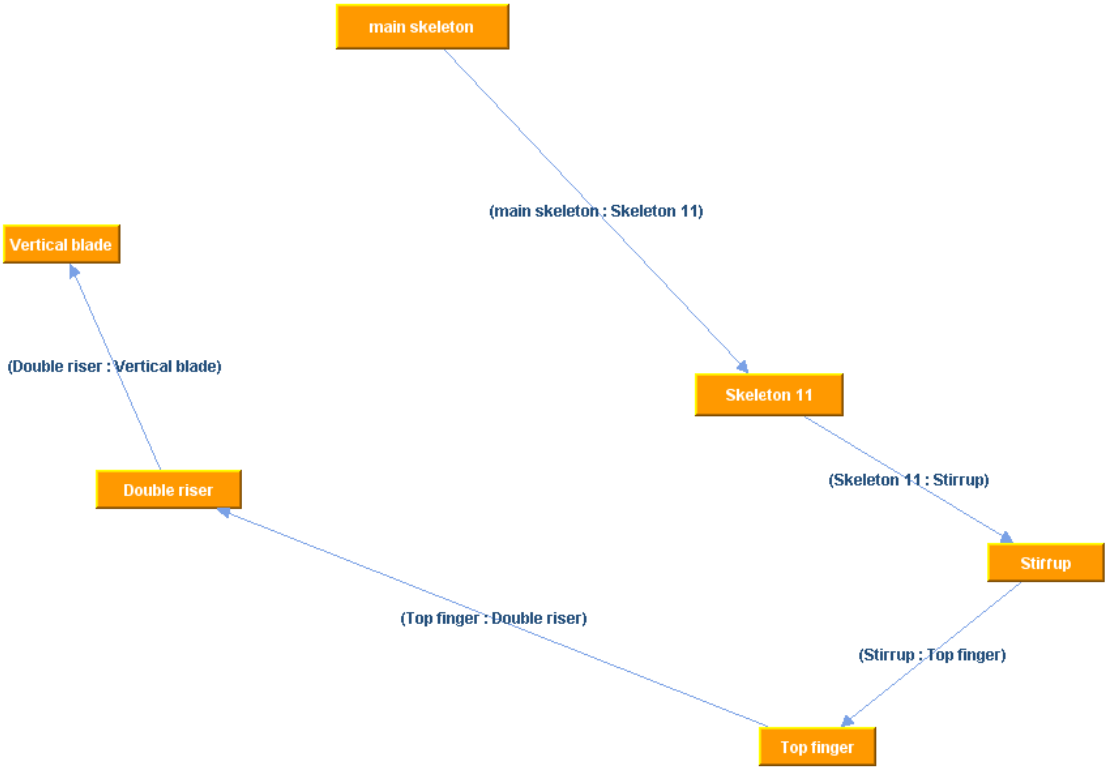


Figure 10.5 – Algorithm 10.1 applied on non-connected graphs, with the `main skeleton` as modified document.

being correct, is still not satisfying because some documents can still be processed at the same time. For instance, on the left of the picture are located the `Vertical blade` and the `Double riser` that have no relations with the three documents on the right. Nevertheless they are present in the same sequence.

Thus even on connected graphs, the algorithm based on the topological sort does not suit to the needs. The linear arrangement of the sequence is a limitation in todays concurrent environment. For this reason another approach has been tested in order to solve this issue.

10.2.3 Ranking

The evaluation of the topological sort showed that constructing update sequences based on dependencies is a relevant approach. However it presents some limitations about the concurrency of the updates as it only provides linear sequences. For this purpose another algorithm has been tested which provides a ranking-based approach.

In addition to the previously tested approach, this approach takes both templates and instances into account. Thus more complex mechanisms are involved.

10.2.3.1 Algorithm

The principle is to attribute an absolute rank r_k, where k is the number of the rank ($k \in \mathbb{N}$), to each document involved by the update. The update of documents is meant to start from documents at r_0, to continue with r_1, then r_2 and so on. Documents located at the same rank have no dependencies against each other, and thus can be processed concurrently. In order to update documents located at a rank r_i, all the documents they depend on have to be located in a lower rank r_j, such as $j < i$. This approach was adapted from the graph layout algorithm from Sugiyama *et al.* [1981].

Algorithm 10.2 presents the developed ranking algorithm. The sequence is built from a list of modified documents and the knowledge present in the ontology. The modified documents are placed at r_0. Modified documents are not necessarily templates, the algorithm also allows to generate update sequences for normal CAD assemblies like the previously presented algorithm does. Furthermore if a document contained in a template is updated, the procedure will consider the template as modified and thus includes it in the computation in order to update its instances.

The approach is different from the topological sort. Here the aim is to build the sequence rank after rank, *i.e.*, breadthwise, instead of going into depth like with the DFS. The algorithm tries to emulate the expert thinking by constructing the sequence iteratively. To achieve that, documents having the current document as dependency are added in the next rank (lines 10-12). Adding them into the next rank keeps them close to their dependency, and thus its presence at this rank is easily understandable when reading the sequence. Once a rank is built, no documents can be added into it. However some documents can be moved to a higher rank if another dependency is found later. So only local operations and moves are realised.

When a template is updated, all its instances are added at a rank located after the rank of the last element contained in the template (algorithm 10.2, line 26). In this way the content of document templates are up-to-date and can thus safely be forwarded to the instances. If an instance is in a "Frozen" state, it will not be updated, *i.e.*, attributed a rank and not added to the queue. Then as the instances have changed, related documents have to be checked, links rebuilt, etc.

The `ManageParentDocuments` function adds necessary container documents (algorithm 10.2, line 16). For example when a document template instance is present at a rank, the assembly containing it should be located before the instance. So it will be loaded before doing a re-instantiation of the template.

When a document template instance is analysed, a check is realised for feature template instances within this document (algorithm 10.2, lines 18-19). If a template feature instance is present,

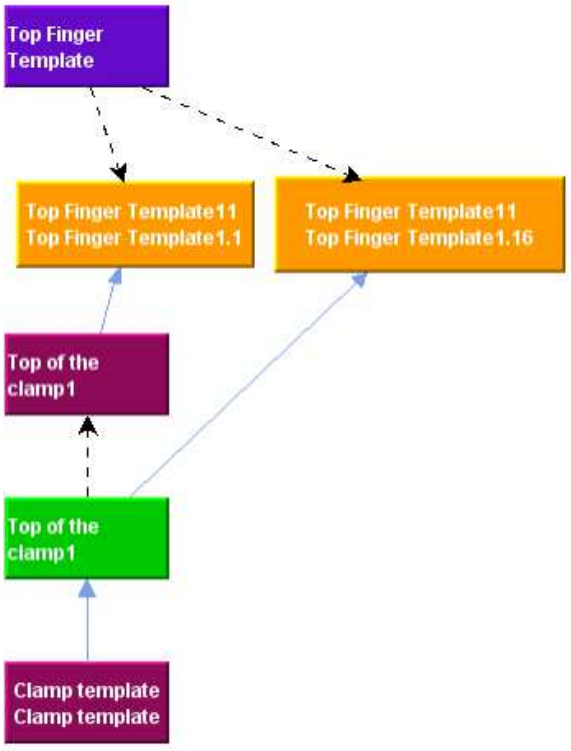


Figure 10.6 – Example of an indirect instance that has to be removed from the sequence as it will be overwritten by the containing template. Example taken from CATIA V5. Blue arrows show *Instance* links (see table 6.1, page 78) and black arrows *InstanceLocation* links (see section 9.2.6.1).

it has to be re-instantiated when it is not present in the document template. This means the template feature has been instantiated in the document template instance, and thus that it will be overwritten by the re-instantiation of the document template. To avoid loosing the feature instance, it has to be re-instantiated in the new document template instance.

The `CleanSequence` function looks for indirect instances and remove them if necessary (algorithm 10.2, line 30). They are removed if one of the documents that includes them is an instance of a document template being updated. Figure 10.6 illustrates this situation. The `Top Finger Template` has two instances, which are targeted by a dashed black arrow. These two instances are located in two different assemblies, the blue arrows represent *Instance* links from CATIA V5. However one of the two assemblies is a template and the other is one of its instances. Thus the second instance from the `Top Finger Template` would be overwritten during the re-instnatiation of the `Top of the clamp1` document. So the second instance can be removed from the sequence as it would result in an unnecessary update. The `CleanSequence` also modifies the sequence in order to remove all empty ranks that may have appeared by the move of documents due to the iterative approach.

10.2.3.2 Results

Figure 10.7 shows the result of the ranking algorithm with the `skeleton` as input, which is the same as for figure 10.5. A sequence composed of four ranks is obtained. The three documents located at rank 1 have no dependencies against each other and can thus be processed concurrently. This is a valuable benefit from this approach. In addition, when comparing both sequences, the second sequence is easier to understand due to the hierarchical representation.

Algorithm 10.2: Update sequence ranking algorithm

Result: An ordered sequence of ranks. Each rank contains a list of documents to be updated.

Input: A list $modifiedElements$ of initially modified documents.
Input: The OWL ontology $onto$.
Output: A list containing lists of documents.

```
1  begin
2      res ← ∅ ;                          // List of documents with a corresponding rank
3      modifiedTemplates ← ∅
4      list ← modifiedElements
5      foreach elem ∈ modifiedElements do
6          SetRank(elem, res, 0)
7      repeat
8          while list ≠ ∅ do
9              elem ← ExtractHead(list) ;                    // List used as a queue
10             curRank ← GetRank(elem)
11             children ← GetLinkTargets(onto, elem, "DependenceLink_inv")
12             SetRank(children, res, curRank + 1)
13             Append(children, list)
14             if elem ∈ (Template ∪ Template_container) then
15                 modifiedTemplates ← elem
16             ManageParentDocuments(elem, res)
17             if IsInstanceDocument(elem, onto) then
18                 featureTemplateInstances ←
                   MoveFeatureTemplateInstance(onto, elem, res)
19                 Append(featureTemplateInstances, list)
               // Look for template documents in which the current element could be
                  contained
20             parentTemplates ← GetParentTemplates(onto, elem)
21             Append(parentTemplates, modifiedTemplates)
22         if modifiedTemplates ≠ ∅ then
               // Process next modified template
23             template ← ExtractHead(modifiedTemplates)
24             if IsNotFrozen(template) then
25                 instances ← GetTemplateInstances(onto, template)
26                 SetInstancesRank(res, template, instances, onto) // Add the instance
                       after the rank of the last document contained in template
27                 Append(instances, list)
28     until modifiedTemplates = ∅
       // Transform the list of document with associated rank into a list of
          ranks associated with a list of documents
29     ranks ← OrderPerRanks(res)
30     CleanSequence(onto, ranks)
31     return ranks
32 end
```

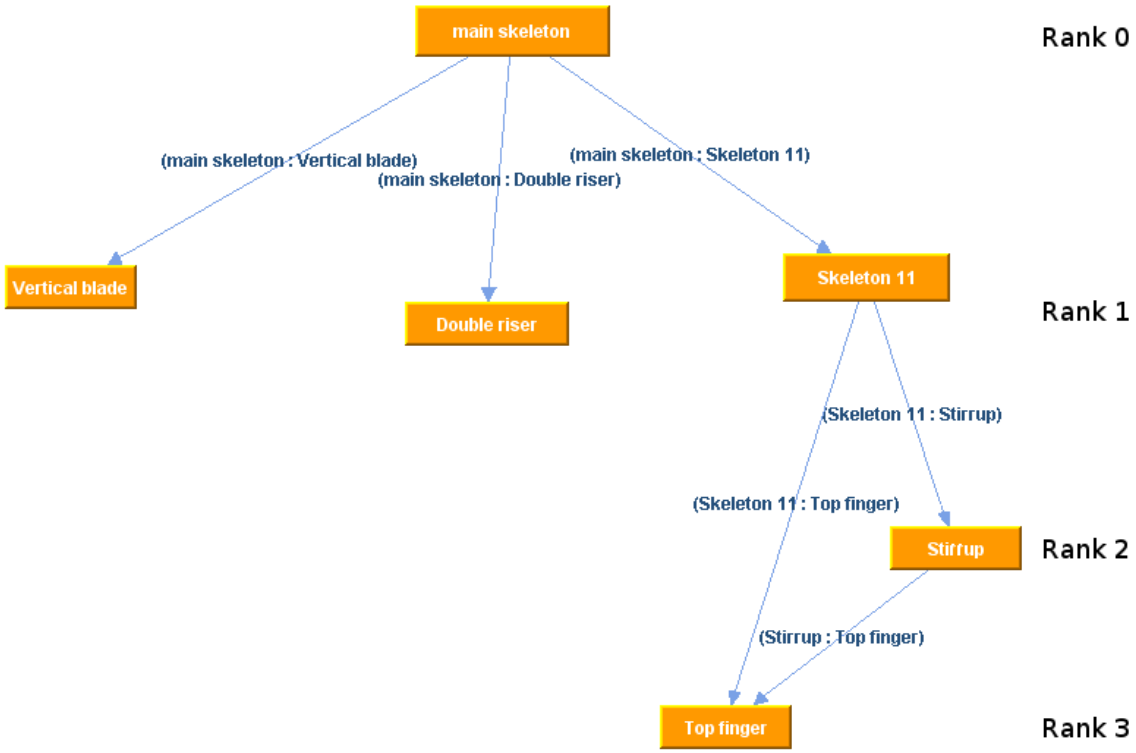


Figure 10.7 – Sequence resulting from the algorithm 10.2 on the graph presented in figure 10.2 with `main skeleton` as modified document. Arrows target the dependent documents.

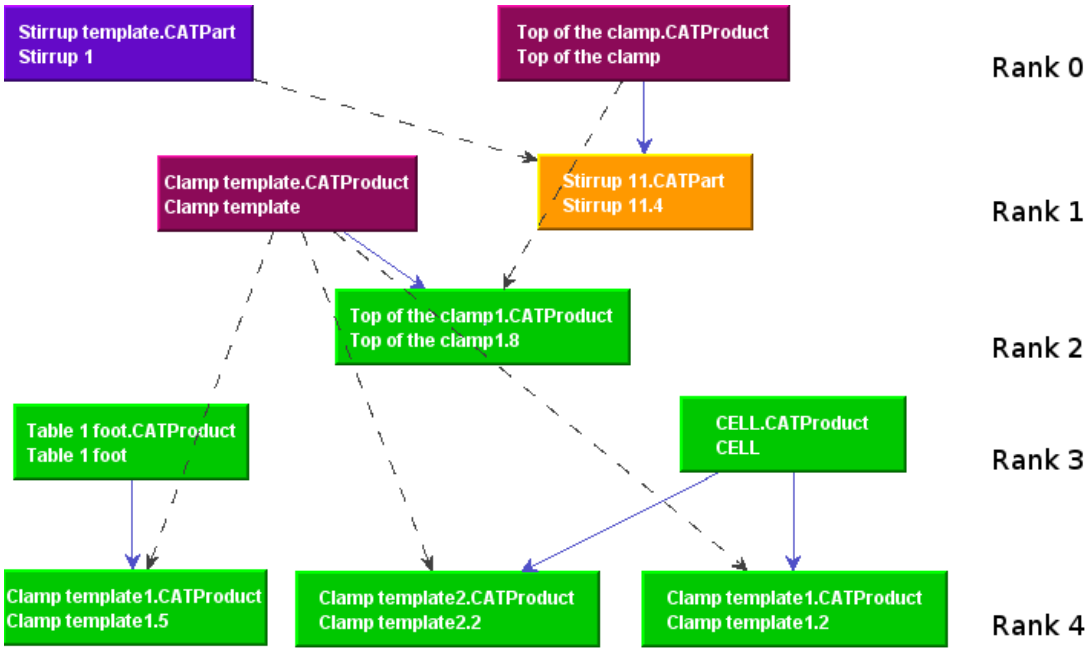


Figure 10.8 – Example of resulting sequence obtained with the ranking algorithm on a problem with templates and template instances. Dashed blue arrows represent *Instance* links (see table 6.1, page 78), dashed black arrows are *InstanceLocation* links (see section 9.2.6.1).

Figure 10.8 gives another example of a rank-based sequence obtained with this algorithm, now including templates and template instances in the set of documents. The set is composed of 96 documents, including 11 templates that are used in three different assemblies. The resulting sequence is composed of five ranks, each rank containing one or more documents. At rank 0 are located two documents, the `Stirrup template.CATPart` that is the originally modified document and the `Top of the clamp.CATProduct`. The latter has been added at this rank because it contains an instance of the modified document, so it has to be loaded in order to be able to re-instantiate the template. However the `Top of the clamp.CATProduct` is also a template and its content has been modified by the update of the `Stirrup` instance. Thus its instances have to be updated too. There is only one instance that is present in the `Clamp template.CATProduct`. This document, being a template too, has its instances added in the sequence in order to update them.

The algorithm was tested on several configurations with different modified documents as input. The results were satisfying when tested, *i.e.*, used to update template instances after the modification of one or more templates and documents. The ranking approach gives a more concise and structured view on documents to process than the topological sort. The documents are less spread, which allows to easily understand which actions have to be undertaken. The notion of sequence to designate the result of the algorithm is kept, however it concerns ranks instead of just documents. As documents located at the same rank have no dependencies against each other, they can be processed at the same time if the necessary resources are available.

10.2.3.3 Complexity analysis

In order to evaluate the computational performance of the ranking algorithm, it was analysed in order to estimate its time complexity. The inputs are the ontology representing d documents and their relations, and the list of m modified documents.

An analysis of the time needed by the algorithm has been performed, according to different inputs. The evolution of the computation time according to the increase of the number of modified documents given as input was analysed. The tests were realised on a Pentium M 1.8 GHz with 2 GB of memory.

Figure 10.9 shows the time evolution with an ontology representing 96 documents including 11 templates. A second ontology instantiated with 25 documents including 7 templates has been used in figure 10.10. The modified documents were randomly selected at each run, in a first time within all documents, then only from the set of templates.

In the first figure, the selection within all documents went until 30 documents, which represents 31.25% of the documents. It is already a high rate of modified documents. The selection within the set of templates started from one template until eleven, which corresponds to all templates available in this example. In the second figure both selection went from one to the highest number of documents available.

When having only templates as input the algorithm takes more time in comparison to the selection of any document. This is the result of the impact of added template instances. The update of an instance implies the update of its dependent documents, which can be located in different assemblies. Thus another region of the graph formed by the documents is analysed. When picking documents randomly, several parts of the graph formed by the documents are also analysed. However no other parts are added to the analysis.

It can also be seen that the standard deviation is reduced when the number of input documents increases when using only templates as input. This means that the algorithm time becomes more regular. The reason behind this behaviour is that when a certain amount of documents is updated, nearly the whole graph will be explored. So the number of analysed documents stabilises. Thus less computations are required as documents are already present in the sequence. The time variation comes from the order in which documents are explored.

The larger test ontology that has been used is composed of less than 100 documents. This is few compared to complex assemblies like cars. No larger testing ontologies were available. Thus it would require the design of several assemblies and templates. However, from the two figures,

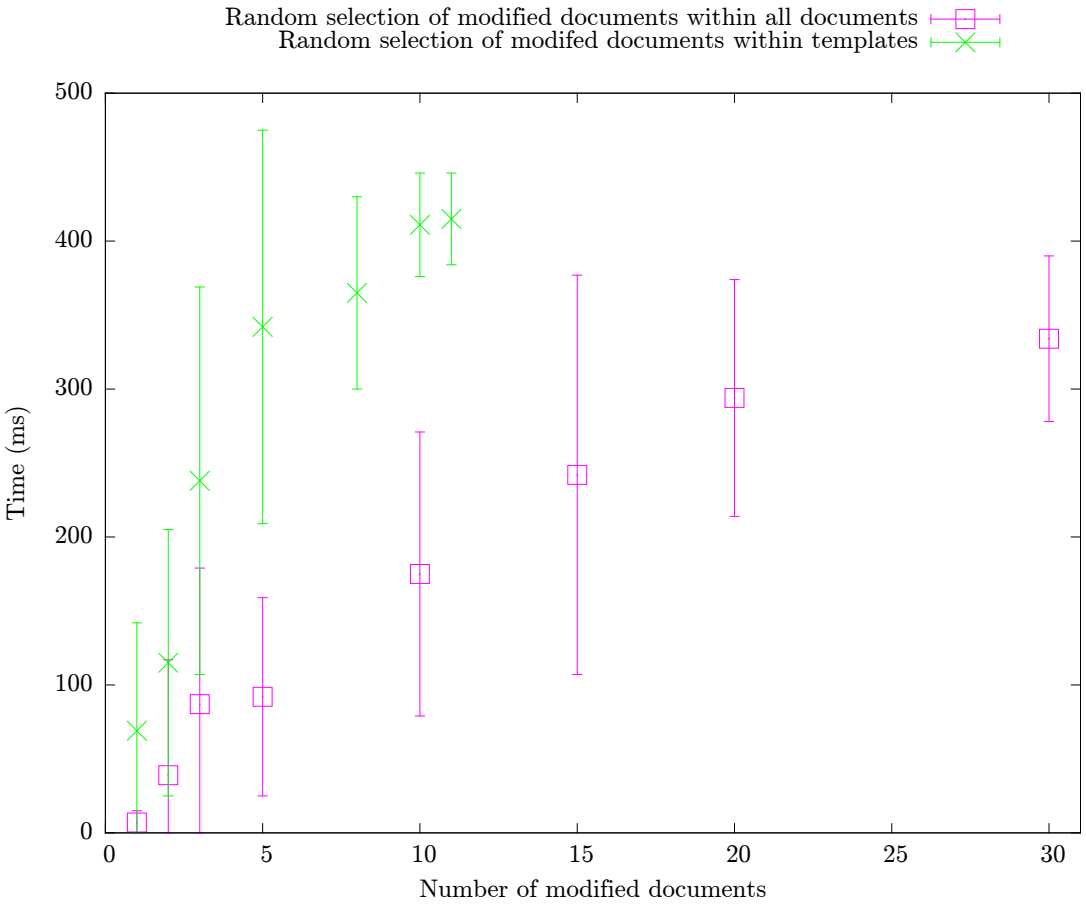


Figure 10.9 – Evolution of the time needed by the algorithm according to the number of modified documents within 96 documents. Values represent the mean and the standard deviation on 10 runs.

a trend can be seen concerning the time complexity of the algorithm according the number of modified documents. The curves raise quickly at the beginning and thereafter slow down The number of modified documents still has an impact on the time. The curves present a logarithmic shape. Thus a complexity of $O(d \times ln(m))$ could be expected from the evolution of the curves as the time does not raise proportionally to the number of input documents and that the time raises slower towards the end.

10.2.4 Cycle handling

The presented algorithms are inapplicable if the graph formed by documents and their dependencies contains cycles. In section 6.3.3.2 CATIA V5 has been tested against cycle creation. It has resulted that no cycles can be created within the geometry but that cycles are allowed between documents through parameters. However it ends up with at least one documents not up-to-date. Nevertheless it has to be taken into account that *Import* links and *KWE_CONTEXTUAL* links can not be differentiated due to CAA API restrictions (see section 6.4.2.1). Thus a combination of both link types can result in a loop between documents. So the approach needs to prevent infinite loops as they are possible in the representation even if there is, in a practical way, no loop inside CATIA V5 models. However this case is rare as the design leads to a hierarchy of documents with the more detailed documents referencing to the ones from a higher level. Nevertheless a cycle detection mechanism has been added for the case that this situation appears.

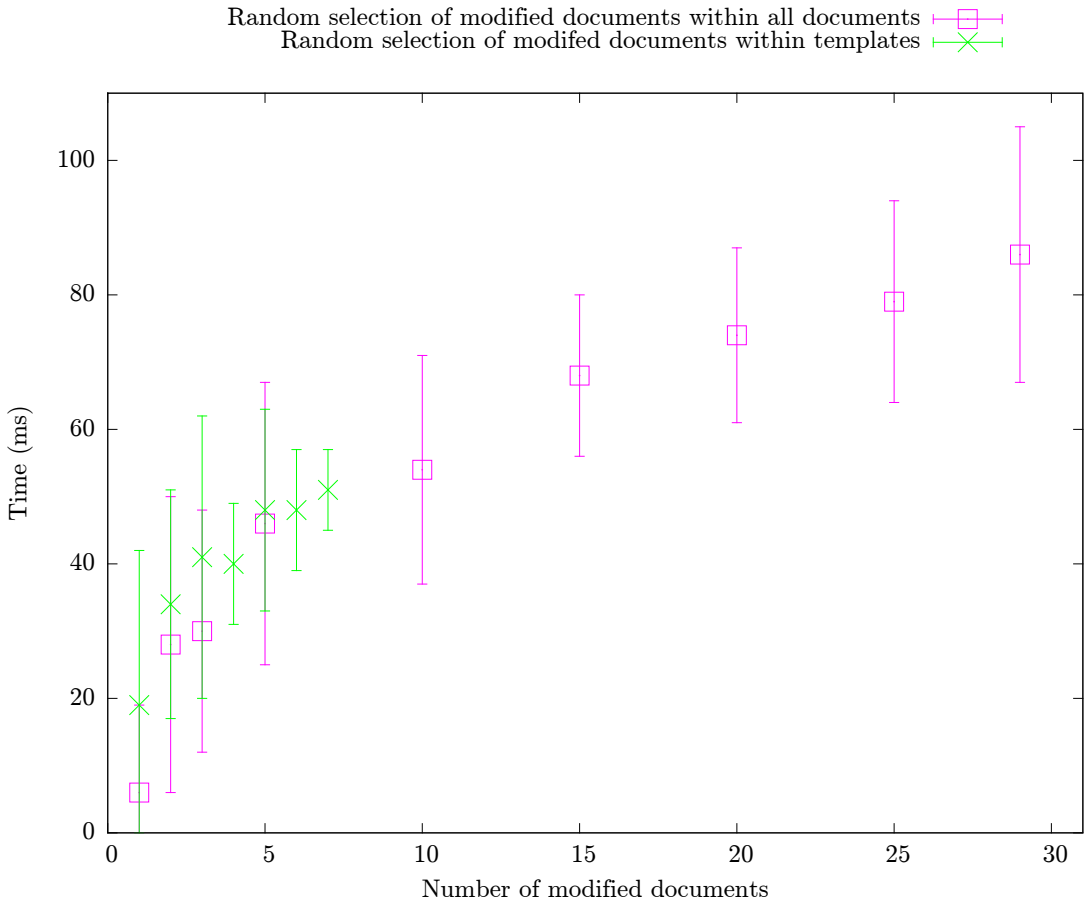


Figure 10.10 – Evolution of the time needed by the algorithm according to the number of modified documents within 25 documents. Values represent the mean and the standard deviation on 10 runs.

In section 4.3.1 some approaches to detect and remove cycles on graph were presented. In the scope of this work cycles have been detected by using an approach close to the one from Floyd [1967], which is presented in section 4.3.1. The cycle detection can not be done in a static manner, *i.e.*, by analysing the whole ontology instances. This would be time consuming and would not tell which relations should be removed in order to delete a cycle. Furthermore the definition of the feedback edge set is a NP-hard problem to solve.

The detection of cycle is realised during the construction of the sequence. In this way the last relation involved in the cycle could be removed in order to delete the cycle. If a document is moved from a previous rank to the rank after which the current document is, a check is realised to see if there is a cycle. That is, a path from the child to the current document is looked for. If such a path is found, the depending document is not moved in order to break the cycle, which corresponds to ignoring the dependency. The document that is not moved will thus not be up-to-date. Each such case has to be solved by hand.

10.3 PDM check-out aware sequence

In order to update instances and other documents, they have to be retrieved from the Product Data Management system (PDM). However the check-out of documents leads to a transfer of several Gigabytes of data. Then the transfer has to be done back in the PDM system. In the previous

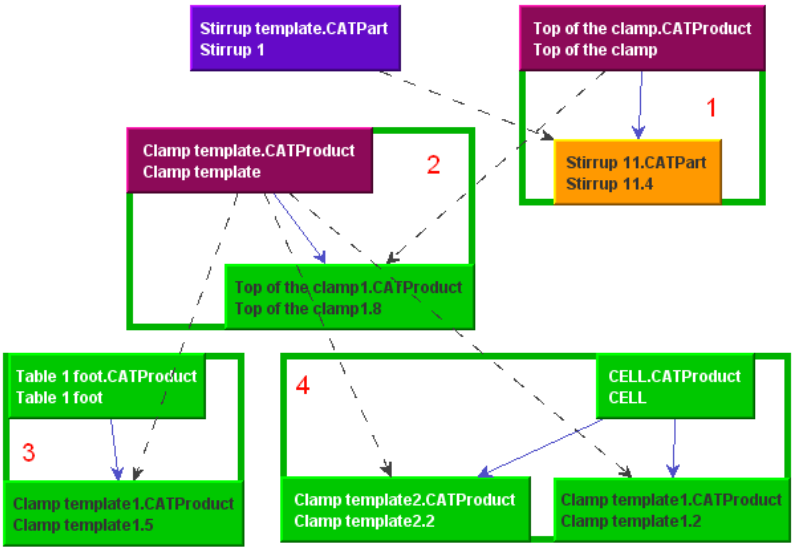


Figure 10.11 – Same result as presented in figure 10.8, but with smart the check-out information. Documents that should be retrieved together are grouped within green rectangles.

section the resulting sequences from the ranking algorithm showed that documents present at the same rank can be processed at the same time because they have no dependencies between them. In certain cases, for example, when replacing several template instances in the same assembly, it can be problematic to do the update of template instances at the same time. In order to update them, the context has to be loaded. Or in this case they all share the same context, which thus has to be loaded for each template instance. If done concurrently, the result of the instances update would lead to a time consuming merge that should be avoided.

For this reason some documents should be retrieved together, which is called the smart PDM check-out. It consists in grouping documents that should be checked-out together. In order to provide this information to the engineers, documents that should be checked-out together are marked in the sequence. When looking closer, a solution consists in checking out the minimal assembly wherein documents are subject to be updated. The corresponding groups for the result presented in figure 10.8 are visible in figure 10.11. Four groups were created corresponding to the four assemblies present in the sequence. Documents located at the same rank and within the same group can not be updated at the same time on remote computers. They have to be processed on the computer were the group has been checked-out.

Once the last document from a group has been updated, the whole group can be checked-in into the PDM system again. In the sequence presented in figure 10.11, the group 1 has to be retrieved first in order to update the template instance. At the same time the group 2 can be checked-out on another computer, however the instance of the `Top of the clamp` can only be replaced when the group 1 has been checked-in in order to access the updated version of the template. Finally, groups 3 and 4 can be processed concurrently on two separate computers.

10.4 Proposed approaches for template instances update

In the previous sections the approaches that allow to create sequences for the update of template instances were introduced. Now based on the generated sequence, the instances have to be updated. This section presents three solutions to perform this task.

10.4.1 Re-instantiation

The first way to update an instance is to remove the old one and to instantiate the new template definition in order to replace the old instance. This approach allows to replace an instance relatively easily. No specific knowledge except the instantiation of a KBE template is required.

However this approach has several disadvantages. First the current configuration of the template will be lost, *e.g.*, user parameters that drive the design. Furthermore if some modifications to the geometry have been done, they will also be lost. Links from and to this instance may also be broken and have to be rebuilt.

A solution to address the loss of configuration would be to store the configuration and put them back in the new instance. The storage of parameters and other KBE data is already achieved by the ontology. The data are extracted during the analysis of documents.

10.4.2 Apply local changes

Another solution, which presents its advantages in the case of small modifications, is to directly apply the changes made to the template on all its instances. In this way the instance document remains the same. However this approach is highly dependent on the modifications made and would be revealed as inappropriate in most of the cases.

The main advantages of this approach are that the local changes and configurations of the instances are not lost. Obviously problems can arise when the modifications are conflicting with the local changes. Another advantage would be that the links with and from other documents are not broken as they are not necessarily modified. Furthermore in some CAD systems like CATIA V5, documents contain an unique identifier (UUID). In some circumstances, this UUID must not be changed due to PDM related issues, and the creation of a new document will lead to a new UUID. With this solution the identifier would remain unchanged as only the content of the document is changed and no new document is created. The problems of this approach are its complexity in case of conflicts and its limitation to small changes.

10.4.3 Rebuild the template instance content

The third approach for the update of a template instance is a combination of the two previous ones. It consists in keeping the same document but replacing all its content by the content of the template. In this way this modification is seamless to the PDM system as the UUID is not changed. Furthermore, as all the content is replaced, there are no conflicts to be expected. Nevertheless like with the re-instantiation of the template, the configurations and local changes are lost. The larger drawback is the complexity of this approach. It is not realisable by a human in reasonable time. Thus this solution requires to be automated.

10.4.4 Comparison of the proposed methods

The three presented approaches are possible solutions to update a template instance. Engineers in charge of the update will have to choose the most suitable solution. For this purpose they can access the decision support system in order to see the actual modifications that were reported by the template maintainer (see section 8.4). From these information he will be able to evaluate the situation and choose the appropriate method.

Table 10.1 compares the three approaches according to the complexity of updating an instance, if the unique identifier contained within an instance is kept, an estimation of the time needed to update the instance, the estimation of the complexity to automate the approach, and if the modifications made on the old instance are kept or are lost. The re-instantiation of the template is the easiest and a fast way to update the instance. It could be relatively easy to automate according to the CAD system automation capabilities. The second approach, which consists in updating the instance by applying the modifications applied to the template, can be easy when

Table 10.1 – Template instances replacement approaches comparison.

	Complexity	UUID	Update time	Automation complexity	Changes
Re-instantiation	Low	New	Fast	Low	Lost
Local changes	Low to High	Kept	Fast to slow	High	Kept
Content copy	High	Kept	Fast	Medium	Lost

there are few, non conflicting changes, but it can also reach a high degree of complexity when the changes are numerous and conflicting with local modifications in the instance. Its automation requires to identify the differences between the template and the instance, which can be complex. Furthermore it requires a human intervention in case of conflicts. The update of the instance content by reconstructing the content of the template seems to be a unsuitable solution without automation. However it is a fast approach that allows to keep the UUID.

In practice, the re-instantiation is the recommended solution to update the template by hand. However, when all these approaches are automated, the application of the modifications to the instance would be recommended as both UUID and local changes are kept.

The automation of the update approaches is necessary when the number of instances is high. However this aspects have not been addressed yet.

10.5 Chapter summary

This chapter has presented the tested approaches and algorithms developed in order to generate a sequence of update for template instances. This sequence is aimed to support engineers in charge of the template instances update. By following this sequence, the analysis of assemblies and the location of the updated templates' instances, which is a complex and time consuming task, is avoided. This sequence also includes information for smart check-out of assemblies and CAD models from Product Data Management systems. Finally three instance replacement approaches were presented.

Two algorithms were tested to compute the update sequence. The first one is based on the topological sort and allows to create a linear and ordered sequence of documents, so that all dependencies of each document are located before this document. Thus the dependencies will be up-to-date when updating each document. However this approach entails the drawback that the resulting sequence is linear, and thus not suitable for today's concurrent engineering. For this reason a second algorithm has been developed based on ranks. An ordered sequence of ranks is computed, where documents are placed within these ranks. The principle remains the same as with the topological sort, *i.e.*, the dependencies of a document have to be placed in former ranks. Documents with the same rank have no dependencies between them so they can be updated concurrently if the resources are available. In order to avoid conflicts and time consuming merges, assemblies have to be checked-out together. This aspect has been taken into account in the sequence by grouping documents that should be retrieved together.

These algorithms work with the instances from the ontology presented in chapter 9. This ontology represents the documents and relations between these documents, the whole forming a graph. The vertices and edges from this graph are typed according to the concepts of the ontology in which the instance/relation they represent is classified. This allows to define generic concepts and base the algorithms on these concepts. Thus the developed algorithms are CAD system independent due to the use of an ontology.

Three approaches to update a template instance have been proposed. They all present various advantages and drawbacks and the choice of the most suitable approach is case depending.

The following chapters will present the tools that implement the main parts of the whole approach for the template instances update. A scenario is also presented to validate and evaluate the approach based on the developed tools.

PART V

Application

Chapter 11

Methodology's Tools

Contents

11.1 Introduction

The previous chapters have presented the propositions that address the template update related issues. A process has been defined that includes: (i) the collaborative search of design solutions for template issues or new requirements, (ii) an ontological representation of documents and their relationships and (iii) an update sequence computation algorithm, using the ontology, in order to support engineers during the update of template instances.

This chapter describes the different developed tools that support the methodology. The developments were focused on the update sequence computation. An overview of the current architecture of the system is depicted in figure 11.1.

Two main systems were developed. The first system is presented on the left side of figure 11.1 and concerns the CAD system (here CATIA V5). The second system is composed of several modules that allow: (i) the computation of update sequences, (ii) to visualise documents, their relations and the update sequences, (iii) and to instantiate the Web Ontology Language (OWL) ontology with the XML description of documents. The decision support tool is not depicted in the architecture diagram because it has not been implemented yet.

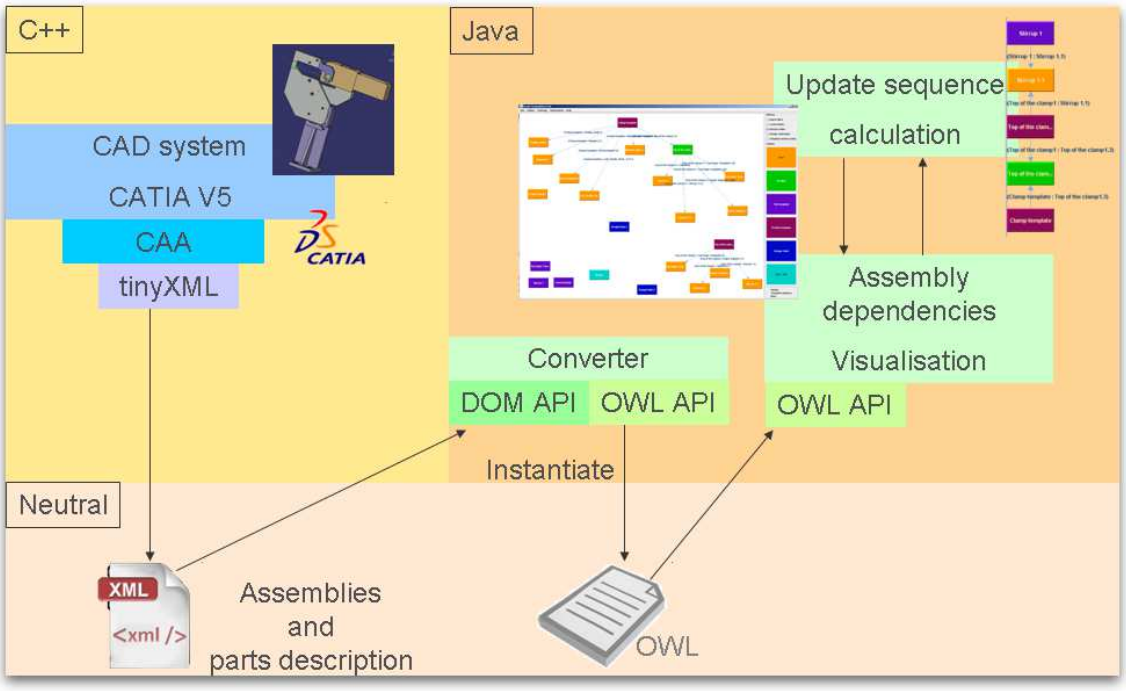


Figure 11.1 – Architecture of the developed software related to the data extraction, dependencies visualisation tool and update sequence computation.

11.2 Collaborative issue solving

An extension of the Issue-Based Information System, which is used to allow the collaborative solving of template issues was presented in chapter 8. The concepts that were presented have been defined within an OWL ontology. This ontology was created using Protégé 4 ontology editor [Protégé 4].

The other parts of the decision support system are not implemented yet. The objective is to design a three-tier architecture (see appendix D) with the ontology as data layer and Web Services for the business logic layer (see figure 3.1 page 47). The presentation layer, which corresponds to

the user interface, can be implemented within heterogeneous applications and platforms thanks to the Web Services. Thus the ontology could be integrated into existing systems, in order to ease the adoption of the collaborative tool.

11.3 CAD models and templates analysis

In order to be able to instantiate the ontology presented in chapter 9, the data has to be extracted from the CAD models. In the scope of this work the focus has been put on the CATIA V5 CAD system. The C++ programming language and the CATIA V5 CAA API (see section 6.4.1.1 on page 83) has been used to develop a batch application. The CATIA V5 CAA API is a commercial product from Dassault Systèmes.

The developed application takes a list of documents to analyse as input and generates an XML file describing the relevant content of each documents (see figure 11.1). An XML file has been chosen as an intermediate file between the CAD models and the ontology for two reasons. First the best APIs to manipulate an OWL ontology are available in JAVA and the document analysis application is developed with C++. Thus a bridge is built between these two programming languages with the XML file. Second is XML as a standard very present in the industry, whereas OWL requires new expertises. Thus an intermediate file is present to facilitate the use of extracted information by legacy systems. To create the XML file, the TinyXML parser has been used [Thomason]. It is a small open source C++ XML parser available under the zlib/libpng license.

Currently all documents have to be analysed each time to update the ontology instances. However an incremental update would reduce the ontology update time.

An example of an XML description of a CAD model is provided in appendix E. The XML-Schema corresponding to the output of the software is available in appendix F.

11.4 Dependencies visualisation and sequence computation

A visualisation software has been developed in order to provide engineers with an overview of the ontology content, *i.e.*, the represented documents and links (see figure 11.2). According to Katzenbach *et al.* [2007] the visualisation of relationships is an asset during the design of a product. The current version is specific to CATIA V5 and allows to visualise the various document types and link types, which are both represented by different colours according to the type. In order to facilitate the understanding, links can be filtered in order to be able to set the focus on specific relations.

The update sequence computation algorithms have been included in the visualisation software. The user selects the modified documents and then starts the computation. The sequence is thereafter shown in the same way as in figure 10.11 (page 125).

The visualisation software also includes the "Converter" that instantiates the OWL ontology from the data present in the XML files.

This whole application has been developed with the JAVA programming language. The manipulation of the ontology is realised with the OWL API 2.2 [OWL API], which is available under the GNU Lesser General Public License (LGPL). The latest version of the OWL API is used by Protégé 4 in order to manipulate ontologies.

In order to infer knowledge in the ontology, two reasoners were tested: FaCT++ 1.2.3 [FaCT++] available under the LGPL and Pellet [Pellet; Sirin *et al.*, 2007] with a dual licensing: AGPL for open source application, and an arranged alternative license for closed source and commercial applications. One of the main differences between them, is that Pellet provides the support for SWRL rules, which is not the case for FaCT++. Thus Pellet suits more in this application because SWRL rules are used.

The graph visualisation is realised *via* JGraph 5.12.3.2, also available under the LGPL [JGraph 5].

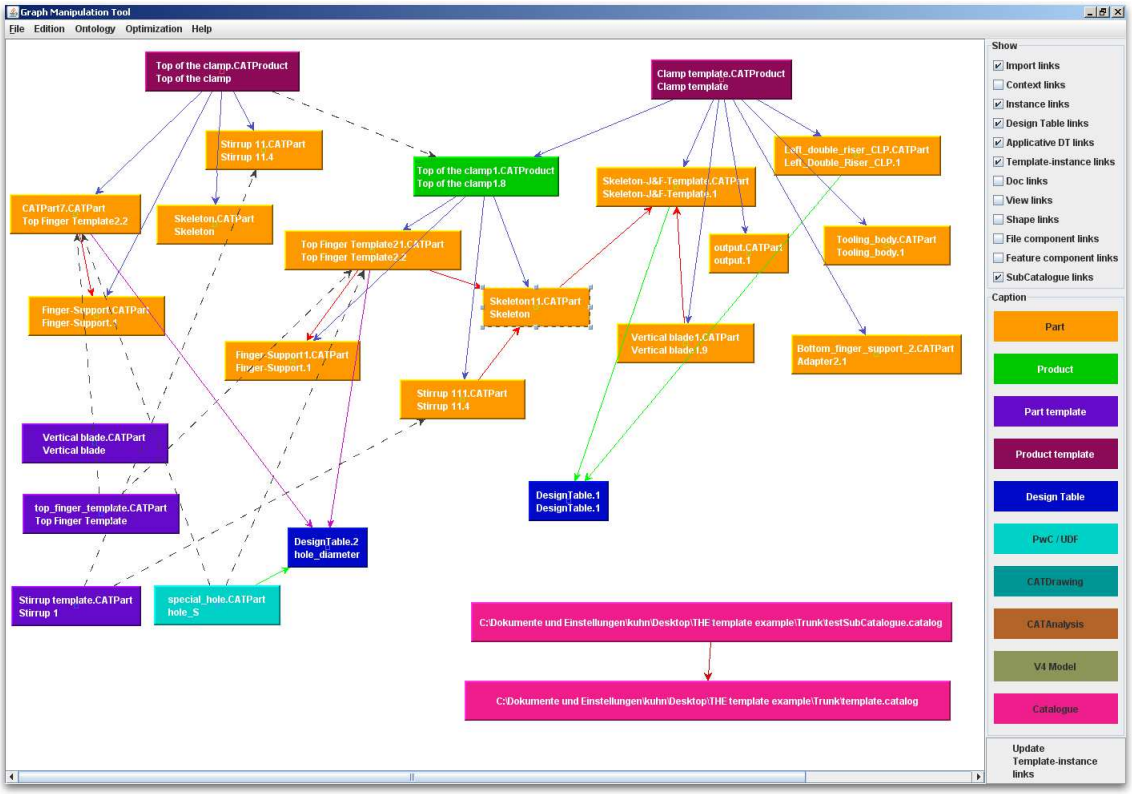


Figure 11.2 – Screenshot of the dependencies visualisation software.

11.5 Chapter summary

In the scope of this work several solutions were elaborated and resulted in several demonstrators. The main software concern the CAD model analysis software and the dependencies visualisation tool including the update sequence computation algorithm. They are all part of the global instance update methodology.

An application has been developed that analyses CATIA V5 CAD models and writes the extracted data in an XML file. Data present in the XML file is then used to instantiate the ontology. The instantiated ontology can be visualised by a visualisation tool that has also been developed. It provides an overview of documents and their relationships. The same software also allows to compute update sequences. Thus the user can directly compute and visualise update sequences according to the modified documents.

The following chapter will present the application of the developed tools in the scope of a scenario.

Chapter 12

Template Modification Scenario

Contents

12.1 Scenario presentation

In the previous chapter the developed tool that support the proposed methodology for template update were presented. In order to demonstrate the approach, a scenario has been defined. This scenario involves several templates that are used in different assemblies. The CAD models and templates were designed using CATIA V5.

The scenario is composed of the following documents:

- 77 CATParts, with six of them that are document templates and one containing a User Defined Function.
- Fourteen assemblies (CATProducts), with two of them being document templates.
- Two external spreadsheets (design tables).
- One CATIA V4 model.
- Two catalogues.

Figure 1.10 on page 18 presents two assemblies, the first being a clamp template and the second a tooling station containing instances from the clamp template. The clamp template contains instances of other templates (part and assembly templates), which creates indirect instances and an interweaving of templates and instances. Both were issues to be addressed.

In order to illustrate the methodology, let us suppose the following case. An engineer is involved in the design of a new tooling station. In order to avoid designing it from scratch, he wants to resort to templates provided within a library, here a catalogue. The tooling station requires to handle metal sheets. So he searches in the library for a template providing this functionality and finds one template that fulfils the requirement, which is a clamp template. However when looking into details, he realises that the clamp is only meant to fasten horizontal surfaces. So he raises a new issue in the decision support system, which will become a new requirement for the clamp template. The new requirement is to allow the clamp to fasten non-horizontal metal sheets.

12.2 Application of the methodology

12.2.1 Solution research

When the engineer has formulated the new requirement for the template, he can add it into the adapted issue-based system. Then the decision process can start, involving several stakeholders: a template maintainer (A), the engineer who has raised the new requirement (B), a person in charge of the costs (C), and an electrical engineer (D). The schema of figure 12.1 illustrates the content of the system at the end of the decision process. The letter in each box presents the participant who has created the element.

12.2.1.1 Collaborative process

The requirement has been formulated as a question (“How to handle non-horizontal metal sheets?”) and is visible at the top of the schema. The “issue” representing the question refers external documents, which are the CAD model of the metal sheet and the metal sheet specifications. The template maintainer proposed three solutions that can fulfil the new requirement. A first solution would be to allow the rotation of the clamp head, a second is to fix the clamp on a dedicated inclined support, and the third is to permanently modify the angle of the clamp head. Each stakeholder can then contribute to by providing his own positions and arguments according to their point of view. For example two possibilities in order to rotate the clamp are proposed, either use one or two degrees of freedom for the rotation of the head of the clamp. This solution seemed the most interesting for the engineer because a manufactured clamp would be reusable for various tooling operations. However the upgrade of the template would require many investigations and analyses and thus a lot of time. Hence the template maintainer questions on the period available to realise

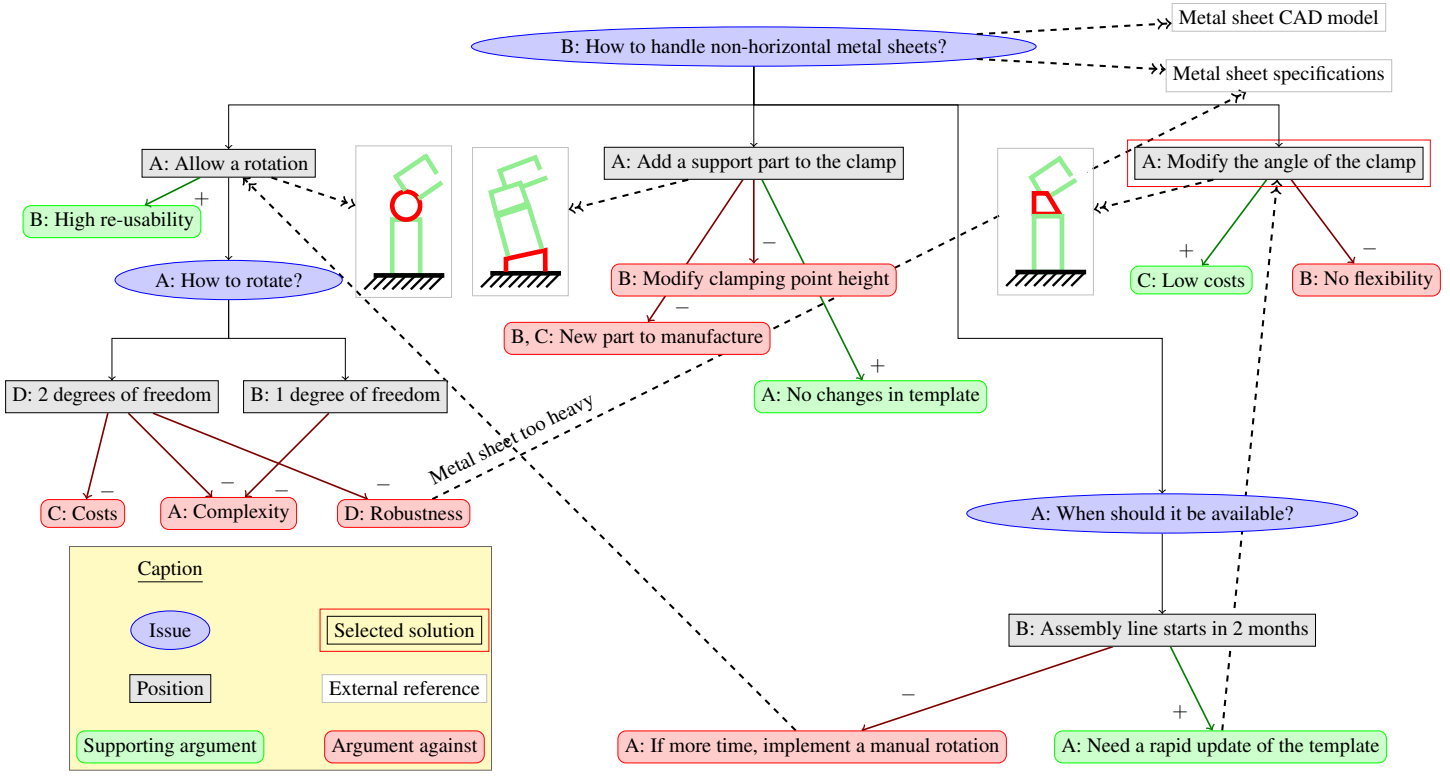


Figure 12.1 – Overview of the content of the decision support system concerning the requirement described in section 12.1.

the upgrade. Due to the short time period available to make the modifications, a more feasible solution has been selected, which is to manufacture the clamp with a fixed angle.

Once the decision has been validated, the planned modifications can be defined. First a user parameter will be defined in the clamp template in order to set the angle of the clamp. Then the necessary CAD models have to be updated in order to take the new parameter into account. The focus should be put towards the `Vertical blade`, which is the element at the centre of the clamp (see purple CAD model in figure 12.2).

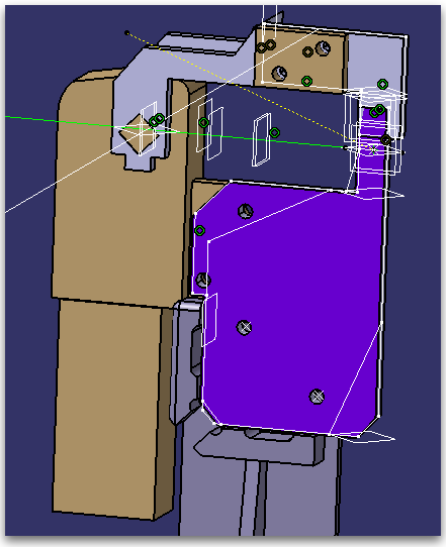


Figure 12.2 – Screenshot of the clamp, depicting the document that will be updated in order to add the new functionality in purple.

12.2.1.2 Update of the template

Before updating the template, the maintainer can have a look at the specifications available in the decision support system. This can be mandatory when the maintainer in charge of the update is not the same as the one who has participated in the decision process. Thus he will be in possession of the necessary information to achieve the update task.

The modifications concern the `Vertical blade` part within the `Clamp template`. However the `Vertical blade` is also a template. Thus the solution is to update its definition too, in order to take into account an angle.

So the actual modifications were made in the `Vertical blade` and the `Clamp template`. The clamp receives a new parameter that has a default value and thus is not considered as a mandatory input. Concerning the `Vertical blade` the geometry has been changed in order to take the new parameter into account. These information are also stored in the system in order to keep track of the evolution of both of the templates.

12.2.2 Assemblies analysis

In order to collect information about the templates, CAD models and their relationships, they have to be analysed. For this purpose all documents that have to be analysed have to be listed in a text file, one document with its full path per line. This text document is then given as argument to the script in charge of the analysis. An XML file will be generated for each analysed document (see appendix E for an example).

Thereafter in the visualisation software, the OWL ontology has to be instantiated from the generated XML files with the help of the "Converter" module. This is achieved by selecting the instantiation item in the menu, and then selecting the generated XML files. Then the inference engine, in this case Pellet, is used to classify the ontology and thus infer the necessary knowledge for further processing. Finally the instanced ontology can be loaded for visualisation.

The graph formed by the clamp template and its related documents and templates are depicted in figure 11.2 on page 134. Other assemblies are not presented within this graph in order to keep it readable. Now that the templates have been updated, the instances and related documents require to be updated in order to take the changes into account.

12.2.3 Generation of the update sequence

In order to determine which documents are impacted by the update of the two templates, the user has to select the modified documents (in this case the `Vertical blade` and the `Clamp template`) and start the sequence computation. The inference engine is run on the ontology and the ranking algorithm is used. The obtained result is presented in figure 12.3. It has been computed in about 100 milliseconds on a Pentium M 1.8 GHz with 2 GB of RAM.

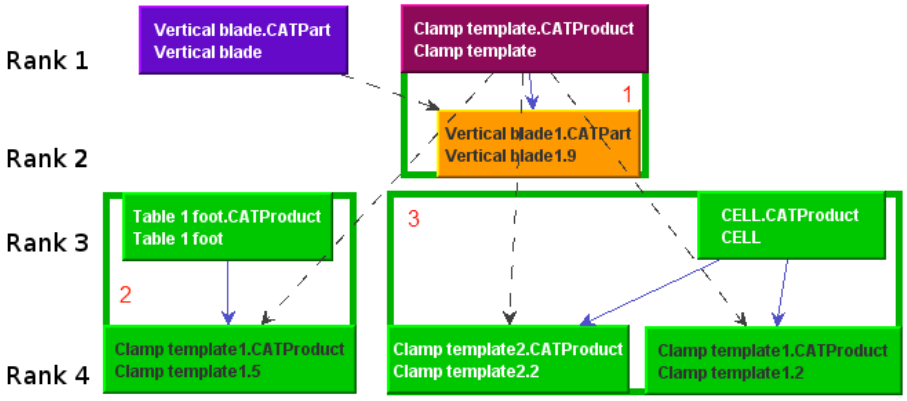


Figure 12.3 – Resulting update sequence with groups after the modification of two templates. Dashed arrows are *InstanceLocation* links (see section 9.2.6.1) and blue arrows are *Instance* links (see table 6.1, page 78). Rank 0 contains the modified documents. Ranks 1 to 3 contain the documents to update. Groups marked 1, 2 and 3 show documents that have to be checked-out together.

The resulting sequence is composed of 4 ranks. The modified documents are located at rank 0. An instance of the `Vertical blade` template is included in the clamp as explained previously. Then the `Clamp template` has three instances located in two assemblies. As you can see the `Vertical blade` is only updated once even if it is also present in the instances of the `Clamp template`. The indirect instances of this template will be updated by the update of the `Clamp template` instances.

Three groups were also created representing the documents that should be checked-out together from the PDM system. They are numbered 1, 2 and 3.

12.2.4 Instances replacement

In order to forward the modifications to existing instances and update the related documents, the sequence has to be followed rank after rank starting from rank 0. Rank 0 contains the updated documents. An instance of the `Vertical blade` has to be updated in the `Clamp template`. Due to the modifications of the geometry in the blade, its re-instantiation in the clamp template is recommended (impact may not be predicted). For this purpose the group marked 1 has to be checked-out.

Now that the `Clamp template` is up-to-date, its instances can be updated. So when looking at rank 2, two assemblies are present, located in two different groups. Hence the groups 2 and 3 can be checked-out on two different computers if the resources are available in order to address these updates concurrently. In any cases, documents located at rank 2 have to be loaded first. When done, the instances present at rank 3 can be updated. Here again the re-instantiation of the `Clamp template` is the fastest approach to update the instances.

This case illustrates the usefulness of the groups, especially group 3 because it avoids possible conflicting updates. As depicted on figure 12.3, the `CELL` assembly contains two instances of the `Clamp template`. Thus if the two instances are re-instantiated concurrently by two engineers, the `CELL` will be retrieved by each engineer who will update an instance. It will result in two `CELL` assemblies with one updated instance in each. Thus both will have to be merged, in order to finally have one `CELL` assembly with both template instances up-to-date.

Once the sequence has been traversed and the respective documents were updated, the result has been positively verified by an expert that all documents requiring an update are up-to-date.

12.3 Chapter summary

This chapter presents a possible scenario on a small set of assemblies. It allows to illustrate and demonstrate the proposed approach and tools. The steps of the process depicted in figure 7.1 are gone through.

In this scenario a requirement is formulated, which will create a discussion in order to find a solution. This decision process involves four persons concerned by the possible changes of a CAD template: a template maintainer, the engineer who raised the requirement, a person in charge of the costs and an electrical engineer. The solutions resulting from the consensus implies modifications in two templates. The template maintainer in charge of the modifications is guided by the information added in the decision support system after the solution validation. After the update of the template, he stores back the realised changes. Then all the CAD models and templates have to be analysed in order to create the ontology instances. The content of the ontology can be visualised and will be used to compute an update sequence for the defined set of CAD documents. By following the sequence, the necessary updates of documents can be achieved efficiently due to the fact that (a) there are no redundant updates, (b) and no updates are missed out. Furthermore the time saved by the automation of the update strategy definition can be used for other activities.

PART VI

Final conclusion

Chapter 13

Conclusions and Perspectives

NOWADAYS the reuse of knowledge during the design of complex products like cars or aircrafts is a key success factor. Knowledge-based engineering aims at the capture, storage and reuse of product related knowledge. Key KBE components are templates, which are intelligent components that are able to adapt themselves to various contexts. During the last years, the use of knowledge templates has become more accessible within companies, which leads to new issues.

The current issues that have been identified concern the update of templates. In the scope of this work two main issues were addressed: the enhancement of the collaboration concerning the solving of the cause implying the modification of a template, and the forwarding of realised modifications to template instances. The complexity of both tasks comes from the collaborative environment wherein several stakeholders are working concurrently on large assemblies. Thus supporting methodologies and tools are beneficial to support and enhance engineers' work in this respect.

The aim of this work was to reduce the time and the effort needed to achieve some defined tasks related to the update of KBE templates. The reduction of time and effort would lead to a higher adoption of KBE templates within the industry, as well as a reduction of costs.

13.1 Contributions

In the scope of this three-year work, several contributions have been accomplished. The contributions are resulting from a multidisciplinary research work involving knowledge-based engineering, decision support systems, knowledge representation, graph theory and computer sciences. All these disciplines interact together in order to provide an original solution to the described issues.

The application area of this work is mainly the design of complex products. Due to the time and quality constraints in this domain, repetitive and mundane tasks have to be reduced as much as possible. The presented contributions propose a first step toward this objective. Moreover it may facilitate the adoption of KBE templates, which are meant to reuse design knowledge and thus save time, as well as enhance the quality of products. The adoption of the methodology would raise the competitiveness of the company.

13.1.1 Theoretical contributions

A generic methodology that supports engineers in the tasks related to the template update has been designed. The process defined in chapter 7 provides an overview on how the specific steps of the proposed methodology have to be processed. The process allows to guide people involved in the template update in order to successfully achieve defined related tasks. The process covers the steps from the definition of the issue or the new requirement up to the update of template instances. This involves the bookkeeping of a computer representation of document knowledge and the computation of an update sequence to guide engineers in the template instance update task. The process is not meant to be comprehensive - it could be enhanced in order to address other template related issues. For example, the template update phase can be further detailed because it was not directly addressed by this work.

A decision support system has been proposed that is an extension of the well known Issue-Based Information System (IBIS) framework. It was specialised in order to help stakeholders find an appropriate solution to a template issue or new requirement concerning a template. This

choice has been made because the IBIS also provides design rationale as the users' contributions that have led to a decision are stored. Additional information concerning planned and actual modifications are also stored in the system. Planned modifications aim at supporting template maintainers. Actual modifications are stored for documentation purposes as well as an aid for engineers who will have to update template instances. In this way they will directly be informed of the realised modifications and will be able to choose the appropriate approach for the corresponding instance update.
In addition to the collaboration aspects, the decision system also allows to document the evolution of the templates as well as to provide a rationale behind the evolution. So these information can be reused or referred to in further discussions. Thus within the same system, pieces of information from the template life cycle are gathered. This type of system can be used for other solving purposes than for template update. For this reason, an ontological representation has been chosen in order to ease its evolution and integration with other existing ontologies.

An ontology has been defined in order to have a computer understandable and processable representation of documents and their relationships. This ontology is dedicated to be used for the update sequence computation. The documents and their relationships have been represented with the Web Ontology Language (OWL). OWL provides a formal representation as well as reasoning mechanisms. Reasoning allows to infer new knowledge and to classify documents under concepts that could not be explicitly retrieved from the analysis of CAD documents. For example, in CATIA V5, no distinction is made between a template instance and a standard CAD document. Within the ontology, a document that is an instance will be explicitly classified as a template instance *via* inference. Moreover three levels of concepts have been defined in the ontology, each level corresponding to a different degree of abstraction. The mid level presents concepts related to the template instance update problem. These concepts are independent from the used CAD system. Then instances created from the analysis of CAD documents will be classified within these concepts *via* inference.
The use of an ontology based on open standards is a good choice for the sustainability of the system. Hence its maintenance, evolution as well as integration within other systems is facilitated. For example, the ontology could also be reused and serve as a basis for building a more comprehensive definition of the KBE domain, which can thereafter be used by other applications. Furthermore as the computation of the algorithm is based on generic concepts from the ontology, the addition of a new CAD system can be easily achieved by an update of the ontology.

An algorithm has been elaborated, which constructs update sequences by using the knowledge present in the ontology. The sequences are meant to guide the engineers when updating necessary documents after the modification of one or more templates. A sequence is constructed based on the dependencies between documents. It uses the CAD system independent concepts defined in the ontology. In this way a unique algorithm is capable to address several CAD systems. Thus the definition of the domain in the ontology plays an important role. The algorithm creates a ranked sequence that allows to process parts of the sequence concurrently, if the necessary resources are available. This appears to be important in today's world, where product design is a concurrent engineering task. Finally three approaches to update instances have been proposed and compared, each presenting benefits and drawbacks according to various criteria.
The update of template instances is a time consuming task. To accelerate this task, the combination of an ontology and a ranking algorithm allows to generate update sequences. The analysis and definition of the update strategy becomes automated and engineers can focus on tasks with more added value. Furthermore it avoids incomplete updates or unnecessary updates due to a bad update strategy.
Besides the forwarding of modifications, the update sequence can also be used when intending to replace a template and all its instances by a different template. The template which should be replaced has just to be considered as modified and the resulting sequence will present the location of the instances in order to replace them with instances of the new template. More generally, the same approach could be also used to estimate the impact of the modification

of elements within documents, for example, to compute the impact of the deletion of a parameter.

13.1.2 Practical contributions

A C++ batch application that analyses and extracts relevant information from CATIA V5 documents has been developed using the CATIA V5 CAA API. It allows to analyse CATParts, CATProducts, catalogues, CATDrawings and CATAnalysis in order to extract links, KBE elements, and document and template related information. The extracted information is stored within an XML file for each analysed document.

An OWL ontology has been created using Protégé 4. This ontology can be accessed, instantiated and modified by using the OWL API. Thus it can easily be reused within different applications.

A Java software has also been developed. It is composed of several modules:

- An XML to OWL converter, which allows to instantiate the ontology from the information extracted from the CATIA V5 documents.
- An update sequence computation module, which uses the knowledge contained within the ontology to generate update sequences for template instances. It allows to automate this task so engineers can focus on value adding tasks.
- A visualisation tool, which allows to visualise the ontology content, *i.e.*, documents and their relations. The knowledge gathered within the ontology can be visualised as graphs in order to provide an overview of the assemblies and their relationships. This feature is interesting as it provides more information than the CAD system's tools, for example, CATIA V5 only allows to see links from one document at a time. Thus engineers can resort to this overview in order to analyse and understand the relations between documents, which results in a speed up of the design. Update sequences can also be visualised with this module.

13.2 Perspectives

This work opens many perspectives concerning the enhancement and extension of the introduced methodology and tools.

Further investigations could benefit from the availability of highly complex scenarios with template usage (not given as far today). The realised tests were limited on an actual scenario that does not have the same complexity as the set of documents by a car manufacturer. This would allow to enhance evaluations on the scalability and robustness of the combination realised by the ontology and the algorithm.

Then, the access to the decision system *via* Web Services has to be implemented. This will provide a standard access that will allow to connect heterogeneous systems. The decision support system currently provides generic concepts. It can thus be specialised to also address other issues besides the solving of template problems. This can be easily achieved by adding the necessary concepts. The ontological representation allows to define a hierarchy of concepts in order to specialise the existing ones. Domain specific concepts could also be added in order to, for example, formalise the ideas of the contributors. Another functionality that is currently missing is the search in past decisions and the template documentation. This functionality could be added by using SQWRL [O'Connor and Das, 2008], which is a query language for OWL. Furthermore, the addition of case-based reasoning by using the information present in the decision support system may be worth the investigation. Past decisions can thus be reused and then adapted to the new case.

Regarding the ontology designed to represent documents and their relations, it would be interesting to investigate the potential benefits of using OWL 2, which was released end of 2009. The possible benefits would be the smaller reasoning computational time. Thus larger ontologies with more instances could be addressed and classified more quickly. The latest version of the OWL API

supports OWL 2, but also a new reasoner called HermiT [Motik *et al.*, 2009b]. According to its specifications and description, HermiT has a LGPL license, supports rules and includes a faster calculus algorithm called "hypertableau." Furthermore it would be a good substitute for the Pellet reasoner due to its less restrictive license.

This ontology could also be extended from an application ontology to a domain ontology by a more comprehensive definition of the KBE domain. Another extension direction would be to specify concepts related to the functionalities of the templates, not only their implementation. In this way the search for an adequate template would be more efficient than searching a simple library, for example, by using the SQWRL language to answer semantic queries.

Other CAD system concepts should also be added to the ontology in order to extend the range of supported systems. The latest CAD system from Dassault Systèmes, CATIA V6, should also be investigated, in order to identify the differences concerning KBE templates. The addition of a new CAD system in the ontology implies the development of a new tool to extract the data from the documents in order to be able to instantiate the new concepts, and thus compute an update sequence.

The visualisation of the ontology content can also be enhanced by the addition of a layout algorithm, especially for the visualisation of large graphs, which make it difficult to represent and to focus on relevant information. A solution would be to create a context aware layout algorithm that shows only the necessary elements from the graph according to user's preferences and the current selection. An easy navigation system would add substantial value.

The representation of the update sequence can also be improved by adding an explicit action to undertake for each document. Right now, the sequence only shows the documents and the order in which they have to be processed. Thus, specific training or experience is required to understand the implicit meaning of the boxes in the sequence.

To complete the reduction of repetitive and non-productive tasks related to the template update, the update of instances has to be automated. Updating instances is certainly a time consuming task, however in some cases it can be achieved by just one engineer. When addressing large assemblies including several thousands of instances, the challenge becomes impossible. Thus the automation of the replacement is required. However such an automation has to be developed for each CAD system individually, which requires an excellent mastering of the different CAD system APIs.

Bibliography

Abdul-Ghafour, S. *Interopérabilité Sémantique Des Connaissances Des Modèles De Produits à Base De Features*. Ph.D. thesis, Université de Lyon, 2009.

Alani, L. I. A. M. *Template-basierte Erfassung von Produktanforderungen in einem CAD System*. Ph.D. thesis, Technischen Universität Berlin, 2007.

Alexander, L. Decision support systems in the 21st century. *SIGSOFT Softw. Eng. Notes*, 27(5):pp. 104–104, 2002. ISSN 0163-5948. doi:http://doi.acm.org/10.1145/571681.571692.

Alon, N. Ranking tournaments. *SIAM J. Discret. Math.*, 20(1):pp. 137–142, 2006. ISSN 0895-4801. doi:http://dx.doi.org/10.1137/050623905.

Alonso, G., Casati, F., Kuno, H., and Machiraju, V. *Web Services. Concepts, Architectures and Applications*. Springer-Verlag, 2004. ISBN 3-540-44008-9.

Alvarado, M., Sheremetov, L., Bañares-Alcántara, R., and Cantú-Ortiz, F. Current challenges and trends in intelligent computing and knowledge management in industry. *Knowledge and Information Systems*, 12(2):pp. 117–127, 2007.

Andersen, O. A. and Vasilakis, G. *Geometric Modelling, Numerical Simulation, and Optimization*, chapter Building an Ontology of CAD Model Information, pp. 11–40. Springer Berlin Heidelberg, 2007. ISBN 978-3-540-68782-5 (Print) 978-3-540-68783-2 (Online). doi:10.1007/978-3-540-68783-2_2. URL `http://www.springerlink.com/content/qx4r21362gu59632/`.

Angele, J., Erdmann, M., and Wenke, D. *Ontology Management*, volume 7 of *Semantic Web and Beyond*, chapter Ontology-Based Knowledge Management In Automotive Engineering Scenarios, pp. 245–264. Springer US, 2007.

Angele, J., Fensel, D., Landes, D., and Studer, R. Developing Knowledge-Based Systems with MIKE. *Automated Software Engineering*, 5(4):pp. 389–418, 1998. ISSN 0928-8910. doi:http://dx.doi.org/10.1023/A:1008653328901.

Antegnard, L., Liese, H., and Stjepandic, J. Intellectual property protection in concurrent engineering domains. In Ghodous, P., Dieng-Kuntz, R., and Loureiro, G. (Editors), *Leading the Web in Concurrent Engineering, Next Generation Concurrent Engineering*, volume 143 of *Frontiers in Artificial Intelligence and Applications*, pp. 338 – 345. IOS Press, 2006.

Arndt, H. *Eine Ontologie-basierte Methode zur Entscheidungsunterstützung in der Produktentwicklung*. Ph.D. thesis, Technische Universität Berlin, 2007.

Arndt, H., Haasis, S., and Rehner, H.-P. CATIA V5 Template zur Umsetzung von Standardkonzepten. In *Karosseriebautage Hamburg, Internationale Tagung, Vieweg Technology Forum Verlag*, 2006.

Arndt, H., Haasis, S., and Winterstein, R. Roll-out template-based engineering process. In *DaimlerChrysler AG, EDM Forum "Integration for Cooperation"*, 2007.

Arpìrez, J., Gómez-Pérez, A., Lozano, A., and Pinto, S. $(ONTO)^2$ agent: an ontology-based WWW broker to select ontologies. In *Workshop on Applications of Ontologies and Problems Solving Methods. ECAI'98*, 1998.

Baader, F., Horrocks, I., and Sattler, U. Description logics as ontology languages for the semantic web. In *Festschrift in honor of Jürg Siekmann, Lecture Notes in Artificial Intelligence*, pp. 228–248. Springer-Verlag, 2003.

Baader, F., Horrocks, I., and Sattler, U. *Handbook of Knowledge Representation*, chapter Description Logics. Elsevier, 2007.

Baader, F. and Nutt, W. *Description Logic Handbook*, chapter Basic Description Logics, pp. 47–100. Cambridge University Press, 2002.

Bachimont, B. Engagement sémantique et engagement ontologique : conception et réalisation. In *Ingénierie des connaissances. Évolution Récentes et nouveaux défis*, pp. 205–323. Paris: Eyrolles, Z. M. Charlet J., Kassel G., Bourgault D. edition, 2000.

Bechhofer, S., van Harmelen, F., Hendler, J., Horrocks, I., McGuinness, D. L., Patel-Schneider, P. F., and Stein, L. A. OWL Web Ontology Language Reference, 2004. URL `http://www.w3.org/TR/owl-ref/`.

Bernaras, A., Laresgoiti, I., and Corera, J. Building and reusing ontologies for electrical network applications. In Wahlster, W. (Editor), *Proceedings of the 12th European Conference on Artificial Intelligence (ECAI 96): Budapest, Hungary: August 11-16*, pp. 298–302. Wiley, 1996. ISBN 0471968099.

Berners-Lee, T. Information management: A proposal, 1989. URL `http://www.w3.org/History/1989/proposal.html`.

Bohanec, M. What is decision support? In Škrjanc, M. and Mladenić, D. (Editors), *Proceedings of the 4th International Multi-conference Information Society 2001*, volume A, pp. 86–89. Institut Jozef Stefan, Ljubljana, 2001. URL `http://www-ai.ijs.si/MarkoBohanec/WhatDS.pdf`.

Borgo, S., Guarino, N., and Masolo, C. Stratified ontologies: the case of physical objects. In *Proceedings of the Workshop on Ontological Engineering, held in conjunction with ECAI96*, pp. 5–15, 1996.

Borst, W. N. *Construction Of Engineering Ontologies For Knowledge Sharing And Reuse*. Ph.D. thesis, Center for Telematica and Information Technology, University of Twente, NL, 1997.

Brass, E. *Konstruieren mit CATIA V5. Methodik der parametrisch-assoziativen Flächenmodellierung*. Hanser Fachbuch, 2005. ISBN 3-446-41378-2.

Broder, A., Kumar, R., Maghoul, F., Raghavan, P., Rajagopalan, S., Stata, R., Tomkins, A., and Wiener, J. Graph structure in the web. *Comput. Netw.*, 33(1-6):pp. 309–320, 2000. ISSN 1389-1286. doi:http://dx.doi.org/10.1016/S1389-1286(00)00083-9.

Burge, J. and Brown, D. C. Reasoning with design rationale. In *Artificial Intelligence in Design '00*, pp. 611–629. Kluwer Academic Publishers, 2000.

Bylander, T. and Chandrasekaran, B. Generic tasks in knowledge-based reasoning. The right level of abstraction for knowledge acquisition. *Knowledge Acquisition for Knowledge-Based Systems*, 1:pp. 65–77, 1988.

Cebrian-Tarrason, D. and Vidal, R. How an ontology can infer knowledge to be used in product conceptual design. In Springer, B. (Editor), *Computer-Aided Innovation (CAI)*, volume 277 of *IFIP International Federation for Information Processing*, pp. 57–68. Gaetano Cascini, 2008.

Chapman, C. B. and Pinfold, M. The application of a knowledge based engineering approach to the rapid design and analysis of an automotive structure. *Advances in Engineering Software*, 32(12):pp. 903–912, 2001.

Chitta, A., Shankar, K., and Jain, V. K. A decision support system for process planning. *Computers in Industry*, 14(4):pp. 307–318, 2008.

Conklin, J. and Begeman, M. L. gIBIS: a hypertext tool for exploratory policy discussion. In *CSCW '88: Proceedings of the 1988 ACM conference on Computer-supported cooperative work*, pp. 140–152. ACM, New York, NY, USA, 1988. ISBN 0-89791-282-9. doi:http://doi.acm.org/10.1145/62266.62278.

Connolly, D., van Harmelen, F., Horrocks, I., McGuinness, D. L., Patel-Schneider, P. F., and Stein, L. A. DAML+OIL reference description, 18 December 2001. URL `http://www.w3.org/TR/daml+oil-reference`.

Cox, J. J., Roach, G. M., and Teare, S. S. *The Customer Centric Enterprise - Advances in Mass Customization and Personalization*, chapter Reconfigurable Models and Product Templates, pp. 183–208. Springer, 2003.

Crabb, H. C. *The virtual engineer: 21st century product development*. SME, 1998. ISBN 0-872-63491-4.

Davis, R., Shrobe, H., and Szolowits, P. What is a knowledge representation?, 1993. URL `http://groups.csail.mit.edu/medg/ftp/psz/k-rep.html`.

Denker, G. DAML+OIL plug-in for Protégé-2000, 2003. URL `http://www.ai.sri.com/daml/DAML+OIL-plugin/`.

Dieng-Kuntz, R., Minier, D., Růžička, M., Corby, F., Corby, O., and Alamarguy, L. Building and using a medical ontology for knowledge management and cooperative work in a health care network. *Computers in Biology and Medicine*, 36(7-8):pp. 871–892, 2006.

Diestel, R. *Graph Theory*, volume 173 of *Graduate Texts in Mathematics*. Springer-Verlag, Heidelberg, third edition, 2005. URL `http://vg00.met.vgwort.de/na/ddfc84df913d6ef96f8f?l=http://www.math.uni-hamburg.de/home/diestel/books/graph.theory/GraphTheoryIII.pdf`.

Dudenhöffer, F. Plattform-effekte in der Fahrzeugindustrie. In *Controlling*, volume 3, pp. 145–151, 2000.

Dustdar, S., Gall, H., and Schmidt, R. Web services for groupware in distributed and mobile collaboration. In *12th Euromicro Conference on Parallel, Distributed and Network-Based Processing*, pp. 241–247, 2004.

Eades, P. A heuristic for graph drawing. *Congressus Numerantium*, 42:pp. 149–160, 1984.

Eades, P. and Lin, X. Spring algorithms and symmetry. *Theor. Comput. Sci.*, 240(2):pp. 379–405, 2000. ISSN 0304-3975. doi:http://dx.doi.org/10.1016/S0304-3975(99)00239-X.

Ebadi, T., Purvis, M., and Purvis, M. A collaborative Web-based issue based information system (IBIS) framework. Technical report, Department of Information Science, University of Otago, Dunedin, New Zealand, 2009.

Ellis, C. A., Gibbs, S. J., and Rein, G. Groupware: some issues and experiences. *Commun. ACM*, 34(1):pp. 39–58, 1991. ISSN 0001-0782. doi:http://doi.acm.org/10.1145/99977.99987.

van der Elst, S. and van Tooren, M. Application of a knowledge engineering process to support engineering design application development. In Curran, R., Chou, S.-Y., and Trappey, A. (Editors), *Collaborative Product and Service Life Cycle Management for a Sustainable World*, volume 15 of *ISPE International conference on Concurrent Engineering*, pp. 417–431. Springer-Verlag London, 2008.

ESPRIT, . URL `http://cordis.europa.eu/esprit/home.html`.

FaCT++, . URL `http://code.google.com/p/factplusplus/`. FaCT++ is a DL reasoner. It supports OWL DL and (partially) OWL 2.

Farquhar, A., Fikes, R., and Rice, J. The ontolingua server: a tool for collaborative ontology construction. *Int. J. Hum.-Comput. Stud.*, 46(6):pp. 707–727, 1997.

Feigenbaum, E. and McCorduck, P. *The fifth generation: artificial intelligence and Japan's computer challenge to the world*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1983. ISBN 0-201-11519-0.

Fensel, D., Horrocks, I., van Harmelen, F., McGuinness, D. L., and Patel-Schneider, P. F. OIL: An Ontology Infrastructure for the Semantic Web. *IEEE Intelligent Systems*, 16(2), 2001.

Fernandez, M., Gomez-Perez, A., and Juristo, N. METHONTOLOGY: from ontological art towards ontological engineering. In *Proceedings of the AAAI97 Spring Symposium Series on Ontological Engineering*, pp. 33–40. Stanford, USA, 1997.

Fernández-López, M. and Gómez-Pérez, A. Overview and analysis of methodologies for building ontologies. *Knowl. Eng. Rev.*, 17(2):pp. 129–156, 2002. doi:http://dx.doi.org/10.1017/S0269888902000462.

Ferreira da Silva, C. *Découverte de correspondances sémantiques entre ressources hétérogénes dans un environnement coopératif*. Thèse de doctorat en informatique, Université Claude Bernard Lyon 1, 2007. URL `http://liris.cnrs.fr/publis/?id=3527`.

Festa, P., Pardalos, P. M., Mauricio, and Resende, M. G. Feedback set problems. In *Handbook of Combinatorial Optimization*, pp. 209–258. Kluwer Academic Publishers, 1999.

Floyd, R. W. Nondeterministic algorithms. *J. ACM*, 14(4):pp. 636–644, 1967. ISSN 0004-5411. doi:http://doi.acm.org/10.1145/321420.321422.

Frost, R. A. *Introduction to knowledge base systems*. Macmillan Publishing Co., Inc., Indianapolis, IN, USA, 1986. ISBN 0-029-48490-1.

Fruchterman, T. and Reingold, E. Graph drawing by force-directed placement. *Software-Practice and Experience*, 21(11):pp. 1129–1164, 1991.

Gay, P. Achieving competitive advantage through knowledge-based engineering: A best practice guide. Technical report, British Department of Trade and Industry, 2000.

Ghodous, P., Martinez., M., Hassas, S., and Pimont, S. Distributed architecture for design activities. *International Journal of IT in Architecture, Engineering and Construction*, Chimay J. Anumba, 2003.

Ghodous, P. and Vandorpe, D. (Editors). *Advances in Concurrent Engineering*. 7 th ISPE International Conference on Concurrent Engineering : Research and Applications, CE2000, 2000.

Gómez-Pérez, A. Ontological engineering: a state of the art. In *British Computer Society*, volume 2, pp. 33–43. Expert Update, 1999.

Gómez-Pérez, A., Fernández-López, M., and Corcho, O. *Ontological Engineering: With Examples from the Areas of Knowledge Management, E-Commerce and Semantic Web*. Springer, 2004.

Grau, B. C., Horrocks, I., Parsia, B., Patel-Schneider, P., and Sattler, U. Next steps for OWL. In *Proceeding of OWLED-2006*, 2006.

Gross, J. L. and Yellen, J. *Handbook of Graph Theory (Discrete Mathematics and Its Applications)*. CRC Press, 1 edition, 2003. ISBN 1584880902. URL `http://www.worldcat.org/isbn/1584880902`.

Gruber, T. Toward principles for the design of ontologies used for knowledge sharing. In *International Journal Human-Computer Studies*, volume 43, pp. 907–928. Elsevier, 1993a.

Gruber, T. A translation approach to portable ontology specifications. Technical report, Knowledge Systems Laboratory, Computer Science Department, Stanford University, 1993b.

Grundstein, M. *Knowledge Management, Classic and Contemporary Works*, chapter From capitalizing on Company Knowledge to Knowledge Management, pp. 261–287. The MIT Press, Massachusetts, 2000.

Grüninger, M. and Fox, M. S. The role of competency questions in enterprise engineering. In *IFIP WG5.7 Workshop on Benchmarking - Theory and Practice*, 1994.

Grüninger, M. and Fox, M. S. Methodology for the design and evaluation of ontologies. In *International Joint Conference on Artificial Intelligence (IJCQI95), Workshop on Basic Ontological Issues in Knowledge Sharing*, 1995.

Guarino, N. and Giaretta, P. Ontologies and Knowledge Bases: Towards a Terminological Clarification. *Towards Very Large Knowledge Bases: Knowledge Building and Knowledge Sharing*, pp. 25–32, 1995. URL `http://www.csee.umbc.edu/771/papers/KBKS95.pdf.Z`.

Guennuni, I. E. *Konzeption und Entwicklung eines modularen Software-Bausteins zur Extraktion von Beziehungswissen aus CATIA V5*. Master's thesis, Hochschule Darmstadt, 2008.

Hakkarainen, S., Strasunskas, D., Hella, L., and Tuxen, S. Choosing appropriate method guidelines for web-ontology building. In *Conceptual Modeling - ER 2005*, volume 3716 of *Lecture Notes in Computer Science*, pp. 270–287. Springer Berlin / Heidelberg, 2005. ISBN 978-3-540-29389-7. ISSN 0302-9743 (Print) 1611-3349 (Online). doi:10.1007/11568322_18. URL `http://www.springerlink.com/content/p3r88922j8146737/`.

Heffernan, M. and Wilken, K. Data-dependency graph transformations for instruction scheduling. *Journal of Scheduling*, 8(5):pp. 427–451, 2005. ISSN 1094-6136 (Print) 1099-1425 (Online). doi:10.1007/s10951-005-2862-8. URL `http://www.springerlink.com/content/tk0363gt5482j683/`.

van Heijst, G., Schreiber, A. T., and Wielinga, B. J. Using explicit ontologies in KBS development. *Int. J. Hum.-Comput. Stud.*, 46(2–3):Academic Press, Inc., 1997.

Hepp, M. Ontologies: State of the art, business potential, and grand challenges. In *Ontology Management: Semantic Web, Semantic Web Services, and Business Applications*, pp. 3–22. Springer, 2007a.

Hepp, M. Possible ontologies: How reality constrains the development of relevant ontologies. In *IEEE Internet Computing*, volume 11, pp. 90–96, 2007b. URL `http://www.heppnetz.de/files/IEEE-IC-PossibleOntologies-published.pdf`.

Horrocks, I., Kutz, O., and Sattler, U. The even more irresistible SROIQ. In *Proceeding of the 10th International Conference on Principles of Knowledge Representation and Reasoning*, pp. 57–67, 2006.

Horrocks, I., Patel-Schneider, P., and van Harmelen, F. From SHIQ and RDF to OWL: The making of a web ontology language. *Journal of Web Semantics*, 1(1):pp. 7–26, 2003. URL `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.2.7039`.

Horrocks, I., Patel-Schneider, P. F., Boley, H., Tabet, S., Grosof, B., and Dean, M. SWRL: A Semantic Web Rule Language Combining OWL and RuleML, 2004. URL `http://www.w3.org/Submission/SWRL/`.

Hovda, P. What is classical mereology? *Journal of Philosophical Logic*, 38(1):pp. 55–82, 2009. ISSN 0022-3611 (Print) 1573-0433 (Online). doi:10.1007/s10992-008-9092-4. URL `http://www.springerlink.com/content/76l18850p2325p16/`.

Hu, Y., Zhou, X., and Li, C. Internet-based intelligent service-oriented system architecture for collaborative product development. *Computer Integrated Manufacturing*, 23(2):pp. 113–125, 2010.

ICARE forms, . URL `http://web1.eng.coventry.ac.uk/moka/informal.htm`.

IEEE P1600.1. Standard upper ontology working group, 2003. URL `http://suo.ieee.org/`.

Inaba, A., Supnithi, T., Ikeda, M., Mizoguchi, R., and Toyoda, J. Learning goal ontology. In *ECAI2000 Workshop on Analysis and Modelling of Collaborative Learning interactions*, pp. 23–30, 2000.

ISO 10303. Industrial automation systems and integration - Product data representation and exchange, 1994. URL `http://www.steptools.com/library/standard/`.

Jackson, C. The transition from 2D drafting to 3D modeling benchmark report: Improving engineering efficiency. Technical report, Aberdeen Group, 2006.

JGraph 5, . URL `http://www.jgraph.com/jgraph5.html`.

Jula, H. and Candea, G. A scalable, sound, eventually-complete algorithm for deadlock immunity. In *Runtime Verification*, pp. 119–136. Springer, 2008.

Kahn, A. B. Topological sorting of large networks. *Commun. ACM*, 5(11):pp. 558–562, 1962. ISSN 0001-0782. doi:http://doi.acm.org/10.1145/368996.369025.

Kamrani, A. and Vijayan, A. A methodology for integrated product development using design and manufacturing templates. *Journal of Manufacturing Technology Management*, 17(5):Emerald Group Publishing Limited, 2006. URL `http://www.emeraldinsight.com/Insight/viewContentItem.do?contentType=Article&contentId=1742488`.

Katzenbach, A., Bergholz, W., and Rolinger, A. Knowledge-based design - an integrated approach. In Heidelberg, S. B. (Editor), *The Future of Product Development*, pp. 13–22, 2007.

Kifer, M., Lausen, G., and Wu, J. Logical foundations of object-oriented and frame-based languages. *J. ACM*, 42(4):pp. 741–843, 1995.

Knublauch, H., Musen, M. A., and Noy, N. F. *Editing OWL Ontologies with Protégé*, 2003.

Knuth, D. E. *Art of Computer Programming, Volume 1: Fundamental Algorithms*. Addison-Wesley Professional, 3rd edition, 1997. ISBN 0201896834. URL `http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20&path=ASIN/0201896834`.

Knuth, D. E. *Art of Computer Programming, Volume 3: Sorting and Searching*. Addison-Wesley Professional, 2nd edition, 1998. ISBN 0201896850. URL `http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20&path=ASIN/0201896850`.

Koufteros, X. A., Vonderembse, M. A., and Doll, W. J. Integrated product development practices and competitive capabilities: the effects of uncertainty, equivocality, and platform strategy. *Journal of Operations Management*, 20(4):pp. 331–355, 2002.

Kuhn, O., Dutra, M. L., Ghodous, P., Dusch, T., and Collet, P. Collaborative architecture based on Web-Services. In Curran, R., Chou, S.-Y., and Trappey, A. (Editors), *Collaborative Product and Service Life Cycle Management for a Sustainable World*, volume 15 of *ISPE International conference on Concurrent Engineering*, pp. 53–61. Springer-Verlag London, 2008.

Kulon, J., Broomhead, P., and Mynors, D. Applying knowledge-based engineering to traditional manufacturing design. *The International Journal of Advanced Manufacturing Technologies*, 30(9–10):pp. 945–951, 2006.

Kunz, W. and Rittel, H. Issues as elements of information systems. Working paper No. 131, Studiengruppe für Systemforschung, Heidelberg, Germany, 1970.

La Rocca, G. and van Tooren, M. Development of Design and Engineering Engines to support multidisciplinary design and analysis of aircraft. In *Delft Science in Design–A Congress on Interdisciplinary Design*, pp. 107–124. by Faculty of Architecture First edition, 2005.

La Rocca, G. and van Tooren, M. Knowledge-Based Engineering Approach to Support Aircraft Multidisciplinary Design and Optimization. *Journal of aircraft*, 46(6):pp. 1875–1885, 2009.

Lantzman, E. Iterative vs. recursive approaches, 2007. URL `http://blogs.microsoft.co.il/blogs/eyal/archive/2007/11/05/iterative-vs-recursive-approaches.aspx`.

Lee, J. Extending the potts and bruns model for recording design rationale. In *ICSE '91: Proceedings of the 13th international conference on Software engineering*, pp. 114–125. IEEE Computer Society Press, Los Alamitos, CA, USA, 1991. ISBN 0-89791-391-4.

Lee, T., Hendler, J., Lassila, O., *et al.* The semantic web. *Scientific American*, 284(5):pp. 34–43, 2001.

Lehmann, F. *Semantic Networks in Artificial Intelligence*. Elsevier Science Inc., New York, NY, USA, 1992. ISBN 0080420125.

Lenat, D. B., Guha, R. V., Pittman, K., Pratt, D., and Shepherd, M. Cyc: toward programs with common sense. *Commun. ACM*, 33(8):pp. 30–49, 1990. ISSN 0001-0782. doi: http://doi.acm.org/10.1145/79173.79176.

Lenzerini, M., Milano, D., and Poggi, A. Ontology representation & reasoning. Technical report, Dipartimento di Informatica e Sistemistica Antonio Ruberti, Universit di Roma La Sapienza, Roma, Italy, Year unknown.

Liese, H. *Wissensbasierte 3D-CAD Repräsentation*. Ph.D. thesis, Technische Universität Darmstadt, 2003.

Lima Dutra, M., Ghodous, P., Kuhn, O., and Minh, T. A Generic and Synchronous Ontology-based Architecture for Collaborative Design. *Concurrent Engineering, Research and Applications*, 18(1):pp. 65–74, 2010. ISSN 1063 293X. URL `http://liris.cnrs.fr/publis/?id=4569`.

Lukibanov, O. Use of ontologies to support design activities at DaimlerChrysler. In *8th International Protégé Conference*, 2005.

MacLean, A., Young, R. M., Bellotti, V. M. E., and Moran, T. P. Questions, options, and criteria: elements of design space analysis. *Hum.-Comput. Interact.*, 6(3):pp. 201–250, 1991. ISSN 0737-0024. doi:http://dx.doi.org/10.1207/s15327051hci0603\&4_2.

Maculet, R. and Daniel, M. Conception, modélisation géométrique et contraintes en CAO : Une synthèse. *Revue d'Intelligence Artificielle*, 18(5–6):pp. 619–645, 2004.

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

Mbang, S. Durchgängige Integration von Produktmodellierung, Prozessplannung und Produktion am Beispiel Karosserie. In *CAD - Produktdaten "Top Secret" ?!*, 2008.

McCarthy. Circumscription – a form of non-monotonic reasoning. *Artificial Intelligence*, 5(13):pp. 27–39, 1980.

McLean, C. R. Computer-aided manufacturing system engineering. In *APMS '93: Proceedings of the IFIP TC5/WG5.7 Fifth International Conference on Advances in Production Management Systems*, pp. 341–348. North-Holland Publishing Co., Amsterdam, The Netherlands, The Netherlands, 1993. ISBN 0-444-81598-8. URL `http://www.mel.nist.gov/msidlibrary/doc/mclean93.pdf`.

Milton, N. *Knowledge Technologies*. Polimetrica - International Scientific Publisher, 2008.

Minsky, M. A framework for representing knowledge. *AIM-306*, 1974.

Mizoguchi, R. Tutorial on ontological engineering - part 1: Introduction to ontological engineering. volume 21, pp. 365–384. Ohmsha, Ldt, 2003.

Mizoguchi, R. and Ikeda, M. Towards ontology engineering. In *Pacific Asian Conference on Expert systems*, pp. 259–266, 1997.

Moran, T. and Carroll, J. *Design rationale: concepts, techniques, and use*. L. Erlbaum Associates Inc, 1996. ISBN 0-8058-1567-8.

Motik, B., Grau, B. C., Horrocks, I., Wu, Z., Fokoue, A., and Lutz, C. OWL 2 Web Ontology Language Profiles, 2009a. URL `http://www.w3.org/TR/owl2-profiles/`.

Motik, B., Shearer, R., and Horrocks, I. Hypertableau Reasoning for Description Logics. *Journal of Artificial Intelligence Research*, 36(1):pp. 165–228, 2009b.

Musen, M., Gennari, J., Eriksson, H., Tu, S., and Puerta, A. PROTÉGÉ-II: Computer support for development of intelligent systems from libraries of components. *Medinfo*, 8(Pt 1):pp. 766–770, 1995.

Nordmann, K. Standardization of ontologies, 2009. URL `http://kore-nordmann.de/blog/0089_standardization_of_ontologies.html`.

Noy, N. and McGuinness, D. Ontology development 101: A guide to creating your first ontology. Technical report, Stanford University, 2001.

O'Connor, M. The Semantic Web Rule Language. Protégé conference 2009 Tutorial, 2009.

O'Connor, M., Knublauch, H., Tu, S., Grosof, B., Dean, M., Grosso, W., and Musen, M. Supporting Rule System Interoperability on the Semantic Web with SWRL. *Fourth International Semantic Web Conference*, pp. 974–986, 2005.

O'Connor, M. J. and Das, A. K. SQWRL: A Query Language for OWL. In Hoekstra, R. and Patel-Schneider, P. F. (Editors), *OWLED*, volume 529 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2008. URL `http://dblp.uni-trier.de/db/conf/semweb/owled2009.html#OConnorD08`.

OLDHAM, K., KNEEBONE, S., CALLOT, M., MURTON, A., and BRIMBLE, R. A methodology and tools oriented to knowledge-based engineering applications. In *Changing the Ways We Work, Advances in Design and Manufacturing*, volume 8, pp. 198–207, 1998.

Olsen, G., Cutkosky, M., Tenenbaum, J., and Gruber, T. Collaborative engineering based on knowledge sharing agreements. *Concurrent Engineering*, 3(2):pp. 145–159, 1995.

OWL 2. Web ontology language document overview, 2009. URL `http://www.w3.org/TR/owl2-overview/`.

OWL API, . URL `http://owlapi.sourceforge.net/`. The OWL API is a Java API and reference implementation for creating, manipulating and serialising OWL Ontologies.

Parsia, B., Sirin, E., Grau, B., Ruckhaus, E., and Hewlett, D. Cautiously approaching SWRL, 2005. URL `http://www.mindswap.org/papers/CautiousSWRL.pdf`.

Patel-Schneider, P. F. and Horrocks, I. OWL 1.1, 2006. URL `http://www.w3.org/Submission/owl11-overview/`.

Pellet, . URL `http://clarkparsia.com/pellet`. Pellet is an OWL 2 reasoner. Pellet provides standard and cutting-edge reasoning services for OWL ontologies.

Pena-Mora, F., Sriram, D., and Logcher, R. SHARED-DRIMS: SHARED design recommendation-intent management system. *Enabling technologies: Infrastructure for collaborative Enterprises*, pp. 213–221, 1993.

Pennell, J., Winner, R., Bertrand, H., and Slusarczuk, M. Concurrent Engineering - An overview for Autotestcon. In *AUTOTESTCON'89- IEEE International Automatic Testing Conference, Philadelphia, PA*, pp. 88–99, 1989.

Peurriere, J.-L. Dependency graph, 2006. URL `http://www.blender.org/development/release-logs/blender-240/dependency-graph/`.

Phillips, R. E. Dynamic objects for engineering automation. *Commun. ACM*, 40(5):pp. 59–65, 1997. ISSN 0001-0782. doi:http://doi.acm.org/10.1145/253769.253791.

Polanyi, M. *The Tacit Dimension*. Peter Smith Publisher Inc, 1967. ISBN 0844659991.

Prasad, B. What distinguishes KBE from automation, 2005. URL `http://www.coe.org/newsnet/Jun05/knowledge.cfm`. Parker Aerospace, Irvine, CA.

Protégé 4, . URL `http://protege.stanford.edu/`. Developed by Stanford Center for Biomedical Informatics Research at the Stanford University School of Medicine.

Psyché, V., Mendes, O., and Bourdeau, J. Apport de l'ingénierie ontologique aux environnements de formation à distance. *Journal of Sciences et Technologies de l'Information et de la Communication pour l'Education et la Formation*, 10, 2003. URL `http://sticef.univ-lemans.fr/`.

Regli, W., Hu, X., Atwood, M., and Sun, W. A survey of design rationale systems: Approaches, representation, capture and retrieval. *Engineering with Computers*, 16(3-4):pp. 209–235, 2000. ISSN 0177-0667 (Print) 1435-5663 (Online). doi:10.1007/PL00013715. URL `http://www.springerlink.com/content/w7ltxaufpay1rpwm/`.

Rittel, H. W. J. and Webber, M. M. Dilemmas in a general theory of planning. *Policy Sciences*, 4(2):pp. 155–169, 1973. doi:doi:10.1007/BF01405730. URL `http://dx.doi.org/doi:10.1007/BF01405730`.

Row, L. A., Davis, M., Messinger, E., Meyer, C., Spirakis, C., and Tuan, A. A browser for directed graphs. *Softw. Pract. Exper.*, 17(1):pp. 61–76, 1987. ISSN 0038-0644. doi:http://dx.doi.org/10.1002/spe.4380170107.

Russell, S. J. and Norvig, P. *Artificial Intelligence: A Modern Approach*. Pearson Education, 2003. ISBN 0137903952. URL `http://portal.acm.org/citation.cfm?id=773294`.

Sandberg, M. Knowledge based engineering - in product development. Technical report, Luleå Universtity of Technology, 2003.

Schreiber, G., Akkermans, H., and Anjewierden, A. *Knowledge engineering and management: the CommonKADS methodology*. the MIT Press, 1999.

Shadbolt, N., Berners-Lee, T., and Hall, W. The Semantic Web Revisited. *IEEE Intelligent Systems*, 21(3):pp. 96–101, 2006. ISSN 1541-1672. doi:http://dx.doi.org/10.1109/MIS.2006.62.

Shen, W., Hao, Q., and Li, W. Computer supported collaborative design: next term retrospective and perspective. *Computers in Industry*, 59(9):pp. 855–862, 2008.

Siddique, Z. and Boddu, K. A CAD template approach to support web-based customer centric product design. *Journal of Computing and Information Science in Engineering*, 5(4):American Society of Mechanical Engineers, 2005. URL `http://scitation.aip.org/getabs/servlet/GetabsServlet?prog=normal&id=JCISB6000005000004000381000001&idtype=cvips&gifs=yes`.

Simon, H. A. *The New Science of Management Decision*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1977. ISBN 0136161367.

Sirin, E., Parsia, B., Grau, B. C., Kalyanpur, A., and Katz, Y. Pellet: A practical OWL-DL reasoner. In *Software Engineering and the Semantic Web*, volume 5, pp. 51–53, 2007.

Skarka, W. Application of MOKA methodology in generative model creation using catia. *Engineering Applications of Artificial Intelligence*, 20(5):pp. 677–699, 2007.

Smith, M. K., Welty, C., and McGuinness, D. L. OWL Web Ontology Language Guide, 2004. URL `http://www.w3.org/TR/owl-guide/`.

Smullyan, R. *First-order logic*. Dover Publications, 1995. ISBN 0-486-68370-2.

Sowa, J. Distinction, combination, and constraints. In *IJCAI-95 Workshop on Basic Ontological Issues in Knowledge Sharing*, 1995.

Sriram, R. D. *Distributed and Integrated Collaborative Engineering Design*. Sarven, 2002. ISBN 0972506403.

Stevens, R., Goble, C., Horrocks, I., and Bechhofer, S. Building a bioinformatics ontology using OIL. *IEEE Transactions on Information Technology in Biomedicine*, 6:pp. 135–141, 2002.

Stokes, M. *Managing engineering knowledge. MOKA: Methodology for knowledge-based engineering applications*. Professional Engineering Publishing, 2001.

Studer, R., Benjamins, V. R., and Fensel, D. Knowledge engineering: Principles and methods. In *Data Knowledge Engineering*, volume 25, pp. 161–197. Elsevier, 1998.

Su, X. and Ilebrekke, L. A comparative study of ontology languages and tools. In *Advanced Information Systems Engineering*, volume 2348 of *Lecture Notes in Computer Science*, pp. 761–765. Springer Berlin / Heidelberg, 2002. ISBN 978-3-540-43738-3. ISSN 0302-9743 (Print) 1611-3349 (Online). doi:10.1007/3-540-47961-9_62. URL `http://www.springerlink.com/content/0pgul6j9yjhjdqrn/`.

Sugiyama, K., Tagawa, S., and Toda, M. Methods for visual understanding of hierarchical system structures. *IEEE Transactions On Systems, Man, And Cybernetics*, 11(2):pp. 109–125, 1981.

Swartout, B., Ramesh, P., Knight, K., and Russ, T. Toward distributed use of large-scale ontologies. *AAAI Symposium on Ontological Engineering*, 25:pp. 161–197, 1997.

Tamassia, R., Battista, G. D., and Batini, C. Automatic graph drawing and readability of diagrams. *IEEE Transactions on Systems, Man and Cybernetics*, 18:pp. 61–79, 1988.

Tenenbaum, J., Gruber, T., McGuire, J., Weber, D., and Olsen, G. SHADE: Technology for knowledge-based collaborative engineering. *Journal of Concurrent Engineering : Applications and Research*, 1(3):pp. 137–146, 1993.

Thomason, L., . URL `http://www.grinninglizard.com/tinyxml/`. TinyXML is a simple, small, C++ XML parser that can be easily integrating into other programs.

van Tooren, M. and Rocca, G. L. Systems engineering and multi-disciplinary design optimization. In Curran, R., Chou, S.-Y., and Trappey, A. (Editors), *Collaborative Product and Service Life Cycle Management for a Sustainable World*, volume 15 of *ISPE International conference on Concurrent Engineering*, pp. 401–415. Springer-Verlag London, 2008.

Tseng, M. M. and Jiao, J. *Handbook of Industrial Engineering, Third Edition*, chapter Mass Customization, pp. 684–709. Wiley-Interscience, 2007. ISBN 0-471-33057-4.

Uschold, M. and Gruninge, M. Ontologies: Principles, methods and applications. In *Knowledge Engineering Review*, volume 11, pp. 93–136. Cambridge University Press, 1996.

Villemur, T. Modèles et services logiciels pour le travail collaboratif, 2006. Habilitation à diriger des recherches - Laboratoire d'analyse et d'architecture des systèmes du CNRS (LAAS).

Web Services Activity, 2002. URL `http://www.w3.org/2002/ws/`.

Wielinga, B., Schreiber, A., and Breuker, J. KADS: A modelling approach to knowledge engineering. *Knowledge acquisition*, 4(1):p. 53, 1992.

Woods, W. A. What's in a link: Foundation for semantic networks. In Bobrow, D. G. and Collins, A. M. (Editors), *Representation and Understanding: Studies in Cognitive Science*, pp. 35–82. Academic Press, London, 1975.

Zha, X. F., Sriram, R. D., Fernandez, M. G., and Mistree, F. Knowledge-intensive collaborative decision support for design processes: A hybrid decision support model and agent. *Computers in Industry*, 59(9):pp. 905–922, 2008.

Index

Appendices

Appendix A

XML Example

You can find below a short example of XML document[1].

Source code A.1 – XML document generated by the application presented in section 11.3. It describes the template presented in figure E.1.

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<shiporder orderid="889923"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="shiporder.xsd">
  <orderperson>John Smith</orderperson>
  <shipto>
    <name>Ola Nordmann</name>
    <address>Langgt 23</address>
    <city>4000 Stavanger</city>
    <country>Norway</country>
  </shipto>
  <item>
    <title>Empire Burlesque</title>
    <note>Special Edition</note>
    <quantity>1</quantity>
    <price>10.90</price>
  </item>
  <item>
    <title>Hide your heart</title>
    <quantity>1</quantity>
    <price>9.90</price>
  </item>
</shiporder>
```

The structure of XML document can be constraint to match a given scheme. For this purpose a W3C recommendation called XML Schema[2] was published. It defines how the XML document has to be structured. For instances it defines that there can be any number of "items" elements with the "maxOccurs" attribute. An XML schema corresponding to the above XML example would be:

Source code A.2 – XML document generated by the application presented in section 11.3. It describes the template presented in figure E.1.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

<xs:element name="shiporder">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="orderperson" type="xs:string"/>
      <xs:element name="shipto">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="name" type="xs:string"/>
            <xs:element name="address" type="xs:string"/>
```

[1] Available at `http://www.w3schools.com/schema/schema_example.asp`

[2] `http://www.w3.org/XML/Schema`

```
            <xs:element name="city" type="xs:string"/>
            <xs:element name="country" type="xs:string"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="item" maxOccurs="unbounded">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="title" type="xs:string"/>
            <xs:element name="note" type="xs:string" minOccurs="0"/>
            <xs:element name="quantity" type="xs:positiveInteger"/>
            <xs:element name="price" type="xs:decimal"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
    <xs:attribute name="orderid" type="xs:string" use="required"/>
  </xs:complexType>
</xs:element>
</xs:schema>
```

Appendix B

OWL DL

Table B.1 – OWL DL descriptions, data ranges, properties, indivdiuals and data values [Horrocks *et al.*, 2003]

Abstract Syntax	DL syntax	Semantic
Descriptions (C)		
A (URI reference)	A	$A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$
`owl:Thing`	$\top$	`owl:Thing`$^{\mathcal{I}} = \Delta^{\mathcal{I}}$
`owl:Nothing`	$\bot$	`owl:Nothing`$^{\mathcal{I}} = \{\}$
`intersectionOf`$(C_1 \dots C_n)$	$C_1 \sqcap \dots \sqcap C_n$	$(C_1 \sqcap \dots \sqcap C_n)^{\mathcal{I}} = C_1^{\mathcal{I}} \cap \dots \cap C_n^{\mathcal{I}}$
`unionOf`$(C_1 \dots C_n)$	$C_1 \sqcup \dots \sqcup C_n$	$(C_1 \sqcup \dots \sqcup C_n)^{\mathcal{I}} = C_1^{\mathcal{I}} \cup \dots \cup C_n^{\mathcal{I}}$
`complementOf` (C)	$\neg C$	$(\neg C)^{\mathcal{I}} = \Delta^{\mathcal{I}} \backslash C^{\mathcal{I}}$
`oneOf`$(o_1 \dots o_n)$	$\{o_1, \dots, o_n\}$	$\{o_1, \dots, o_n\}^{\mathcal{I}} = \{o_1^{\mathcal{I}}, \dots, o_n^{\mathcal{I}}\}$
`restriction`$(R$ `someValuesFrom`$(C))$	$\exists R.C$	$(\exists R.C)^{\mathcal{I}} = \{ x \mid \exists y.\langle x, y\rangle \in R^{\mathcal{I}}$ and $y \in C^{\mathcal{I}}\}$
`restriction`$(R$ `allValuesFrom` $(C))$	$\forall R.C$	$(\forall R.C)^{\mathcal{I}} = \{ x \mid \forall y.\langle x, y\rangle \in R^{\mathcal{I}} \rightarrow y \in C^{\mathcal{I}}\}$
`restriction`$(R$ `hasValue`$(o))$	$R : o$	$(\forall R.o)^{\mathcal{I}} = \{ x \mid \langle x, o^{\mathcal{I}}\rangle \in R^{\mathcal{I}}\}$
`restriction`$(R$ `minCardinality`$(n))$	$\geqslant n\ R$	$(\geqslant n\ R)^{\mathcal{I}} = \{x \mid \sharp(\{y.\langle x, y\rangle \in R^{\mathcal{I}}\}) \geqslant n\}$
`restriction`$(R$ `maxCardinality`$(n))$	$\leqslant n\ R$	$(\leqslant n\ R)^{\mathcal{I}} = \{x \mid \sharp(\{y.\langle x, y\rangle \in R^{\mathcal{I}}\}) \leqslant n\}$
`restriction`$(U$ `someValuesFrom`$(D))$	$\exists U.D$	$(\exists U.D)^{\mathcal{I}} = \{ x \mid \exists y.\langle x, y\rangle \in U^{\mathcal{I}}$ and $y \in D^{\mathcal{I}}\}$
`restriction`$(U$ `allValuesFrom` $(D))$	$\forall U.D$	$(\forall U.D)^{\mathcal{I}} = \{ x \mid \forall y.\langle x, y\rangle \in U^{\mathcal{I}} \rightarrow y \in D^{\mathcal{I}}\}$
`restriction`$(U$ `hasValue`$(v))$	$U : v$	$(\forall U.v)^{\mathcal{I}} = \{ x \mid \langle x, o^{\mathcal{I}}\rangle \in R^{\mathcal{I}}\}$
`restriction`$(U$ `minCardinality`$(n))$	$\geqslant n\ U$	$(\geqslant n\ U)^{\mathcal{I}} = \{x \mid \sharp(\{y.\langle x, y\rangle \in U^{\mathcal{I}}\}) \geqslant n\}$
`restriction`$(U$ `maxCardinality`$(n))$	$\leqslant n\ U$	$(\leqslant n\ U)^{\mathcal{I}} = \{x \mid \sharp(\{y.\langle x, y\rangle \in U^{\mathcal{I}}\}) \leqslant n\}$
Data Range (D)		
D (URI reference)	D	$D^{\mathcal{D}} \subseteq \Delta_{\mathcal{D}}^{\mathcal{I}}$
`oneOf`$(v_1 \dots v_n)$	$\{v_1, \dots, v_n\}$	$\{v_1, \dots, v_n\}^{\mathcal{I}} = \{v_1^{\mathcal{I}}, \dots, v_n^{\mathcal{I}}\}$
Object Properties (R)		
R (URI reference)	R	$R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$
	R^-	$(R^{\mathcal{I}})^- = (R^{\mathcal{I}})^-$
Datatype Properties (U)		
U (URI reference)	U	$U^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta_{\mathcal{D}}^{\mathcal{I}}$
Individuals (o)		
o (URI reference)	o	$o^{\mathcal{I}} \in \Delta^{\mathcal{I}}$
Data Values (v)		
v (RDF literal)	v	$v^{\mathcal{I}} = v^{\mathcal{D}}$

Table B.2 – OWL DL axioms and facts [HORROCKS *et al.*, 2003]

Abstract Syntax	DL Syntax	Semantics
`Class (`A `partial` $C_1 \dots C_n$`)`	$A \sqsubseteq C_1 \sqcap \dots \sqcap C_n$	$A^{\mathcal{I}} \subseteq C_1^{\mathcal{I}} \cap \dots \cap C_n^{\mathcal{I}}$
`Class (`A `complete` $C_1 \dots C_n$`)`	$A = C_1 \sqcap \dots \sqcap C_n$	$A^{\mathcal{I}} = C_1^{\mathcal{I}} \cap \dots \cap C_n^{\mathcal{I}}$
`EnumerateClasses(`A $o_1 \dots o_n$`)`	$A = \{ o_1, \dots, o_n \}$	$A^{\mathcal{I}} = \{ o_1^{\mathcal{I}}, \dots, o_n^{\mathcal{I}} \}$
`SubClassOf(`C_1 C_2`)`	$C_1 \sqsubseteq C_2$	$C_1^{\mathcal{I}} \subseteq C_2^{\mathcal{I}}$
`EquivalentClasses(`$C_1 \dots C_n$`)`	$C_1 = \dots = C_n$	$C_1^{\mathcal{I}} = \dots = C_n^{\mathcal{I}}$
`DisjointClasses(`$C_1 \dots C_n$`)`	$C_i \sqcap C_j = \bot, i \neq j$	$C_i^{\mathcal{I}} \cap C_j^{\mathcal{I}} = \{\}, i \neq j$
`Datatype(`D`)`		$D^{\mathcal{I}} \subset \Delta_{\mathcal{D}}^{\mathcal{I}}$
`DatatypeProperty(`U `super(`U_1`)...super(`U_n`) )`	$U \sqsubseteq U_i$	$U^{\mathcal{I}} \subseteq U_i^{\mathcal{I}}$
`domain(`C_1`) ... domain(`C_m`)`	$\geqslant 1U \sqsubseteq C_i$	$U^{\mathcal{I}} \subseteq C_i^{\mathcal{I}} \times \Delta_{\mathcal{D}}^{\mathcal{I}}$
`range(`D_1`) ... range(`D_l`)`	$\top \sqsubseteq \forall U.D_i$	$U^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times D_i^{\mathcal{I}}$
`[ Functional ]`	$\top \sqsubseteq \leqslant 1U$	$U^{\mathcal{I}}$ `is functional`
`SubPropertyOf(`U_1 U_2`)`	$U_1 \sqsubseteq U_2$	$U_1^{\mathcal{I}} \subseteq U_2^{\mathcal{I}}$
`EquivalentPropertyOf(`$U_1 \dots U_n$`)`	$U_1 = \dots = U_n$	$U_1^{\mathcal{I}} = \dots = U_n^{\mathcal{I}}$
`ObjectProperty(`R `super(`R_1`)...super(`R_n`) )`	$R \sqsubseteq R_i$	$R^{\mathcal{I}} \subseteq R_i^{\mathcal{I}}$
`domain(`C_1`) ... domain(`C_m`)`	$\geqslant 1R \sqsubseteq C_i$	$R^{\mathcal{I}} \subseteq C_i^{\mathcal{I}} \times \Delta^{\mathcal{I}}$
`range(`D_1`) ... range(`C_l`)`	$\top \sqsubseteq \forall R.C_i$	$R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times C_i^{\mathcal{I}}$
`[InverseOf(`R_0`)]`	$R = ({}^{-}R_0)$	$R^{\mathcal{I}} = (R_0^{\mathcal{I}})^{-}$
`[Symmetric]`	$R = ({}^{-}R)$	$R^{\mathcal{I}} = (R^{\mathcal{I}})^{-}$
`[Functional]`	$\top \sqsubseteq \leqslant 1R$	$R^{\mathcal{I}}$ `is functional`
`[InverseFunctional]`	$\top \sqsubseteq \leqslant 1R^{-}$	$(R^{\mathcal{I}})^{-}$ `is functional`
`[Transitive]`	$Tr(R)$	$R^{\mathcal{I}} = (R^{\mathcal{I}})^{+}$
`SubPropertyOf(`R_1 R_2`)`	$R_1 \sqsubseteq R_2$	$R_1^{\mathcal{I}} \subseteq R_2^{\mathcal{I}}$
`EquivalentPropertyOf(`$R_1 \dots R_n$`)`	$R_1 = \dots = R_n$	$R_1^{\mathcal{I}} = \dots = R_n^{\mathcal{I}}$
`AnnotationProperty(`S`)`		
`Individual (`o `type(`C_1`)) ... type(`C_n`)`	$o \in C_i$	$o^{\mathcal{I}} \in C_i^{\mathcal{I}}$
`value (`R_1 o_1`)...value(`R_n o_n`)`	$\langle o, o_i \rangle \in R_i$	$\langle o^{\mathcal{I}}, o_i^{\mathcal{I}} \rangle \in R_i^{\mathcal{I}}$
`value (`U_1 v_1`)...value(`U_n v_n`)`	$\langle o, v_i \rangle \in U_i$	$\langle o^{\mathcal{I}}, v_i^{\mathcal{I}} \rangle \in U_i^{\mathcal{I}}$
`SameIndividual(`$o_1 \dots$ o_n`)`	$o_1 = \dots = o_n$	$o_1^{\mathcal{I}} = \dots = o_n^{\mathcal{I}}$
`DifferentIndivials(`$o_1 \dots o_n$`)`	$o_i \neq o_j, i \neq j$	$o_i^{\mathcal{I}} \neq o_j^{\mathcal{I}}, i \neq j$

Appendix C

Indirect Instances

Figure C.1 illustrates direct and indirect instances in a real case, which corresponds to the assemblies shown in figure 1.10. Green boxes are templates, orange ones are standard parts and gray boxes represent assemblies. The blue box is the main document from the assembly. Large gray arrows represent the order in which the different models were designed. In this example the focus is pour on the `Top finger` template. The red arrow represents its direct instances and dashed red arrows the indirect instances.

Figure C.2 present the order in which the update of the instances of `Top finger` should be done, in the case of its update. In this case the sequence shown by red arrows is the optimal one because it minimises the number of updates. For example, if the instances present in the `table assembly` are updated first, they may be overwritten by the instantiation of the `Clamp`. When the indirect instance present in the `Clamp` template will be updated, it will result in a modification of the `Clamp` template. Hence its instances, which are not explicitly shown in this figure, will also have to be updated. They are located in the `Table assembly` and their update would overwrite the `Top finger` instances that have already be processed. Furthermore by updating the `Clamp` first, the designed ensure that the context is adapted to the new instance. The same reasoning has to be made for the `Top of the clamp`.

Thus the update should follow the design order from the assemblies, by starting with the direct instances.

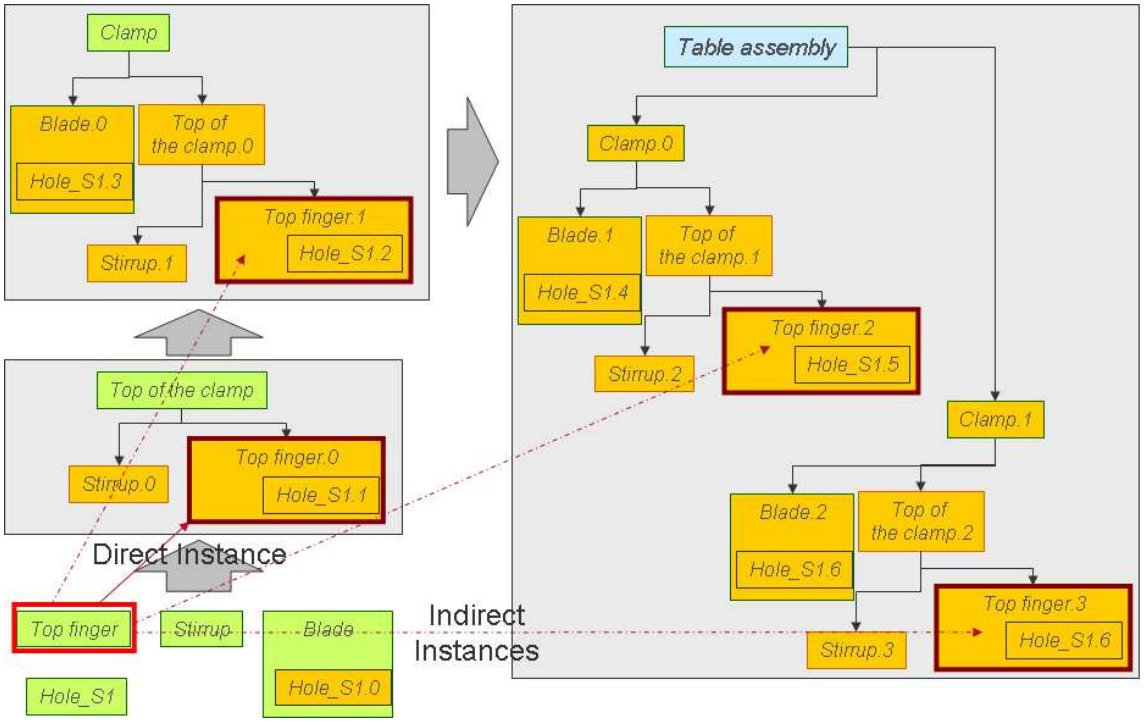


Figure C.1 – Example of direct (red arrow) and indirect (dashed red arrows) instances corresponding to the assemblies presented in figure 1.10.

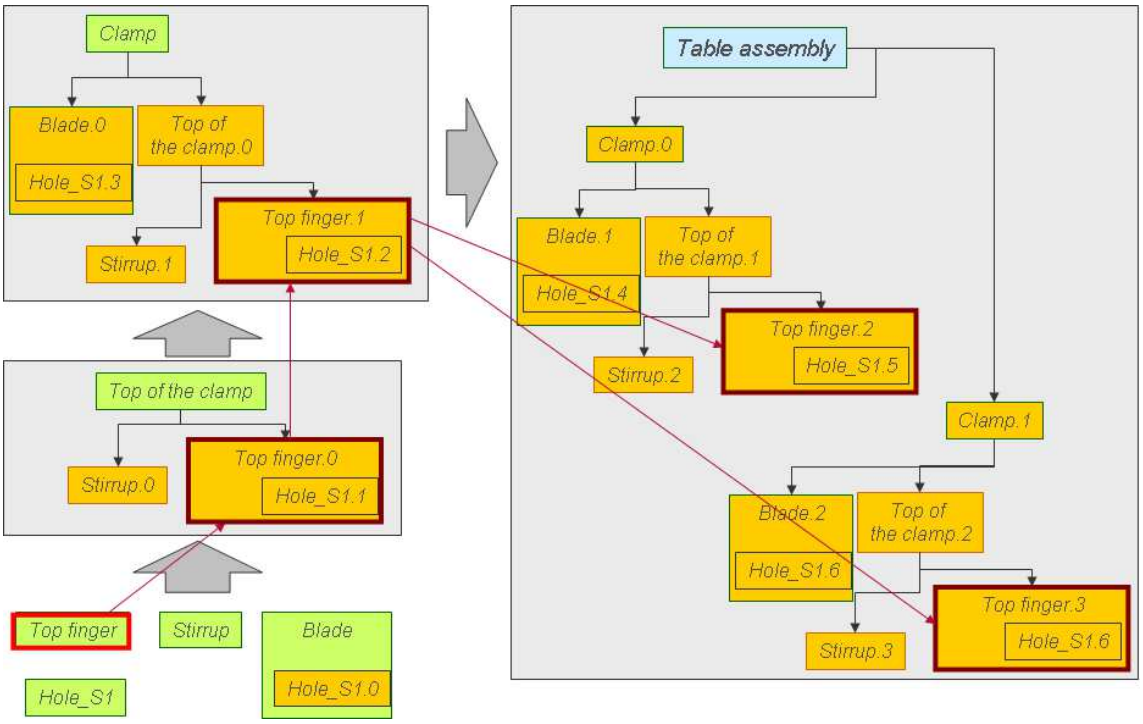


Figure C.2 – Order in which the instances of the top finger template should be updated.

Appendix D

Three-Tier Architecture

Three-tier architecture is a client-server architecture wherein the user interface, the logic layer and the data storage layer are independent modules. They can thus be developed and maintained individually. Any of these three components can be replaced by a new one because they communicate together through well defined interfaces.

Figure D.1 illustrates this architecture. At the bottom of the schema is the data layer, which is in charge of the storage. The storage can be realised by different technical solutions, such as databases, text files or on a peer-to-peer network. In the middle, the business logic layer provides the application's functionalities and processing. At the top, the presentation layer displays the information. This layer can be implemented by various technologies and on heterogeneous devices and systems. The communication between the three modules is linear, that means that the presentation layer does not communicate directly with the data layer.

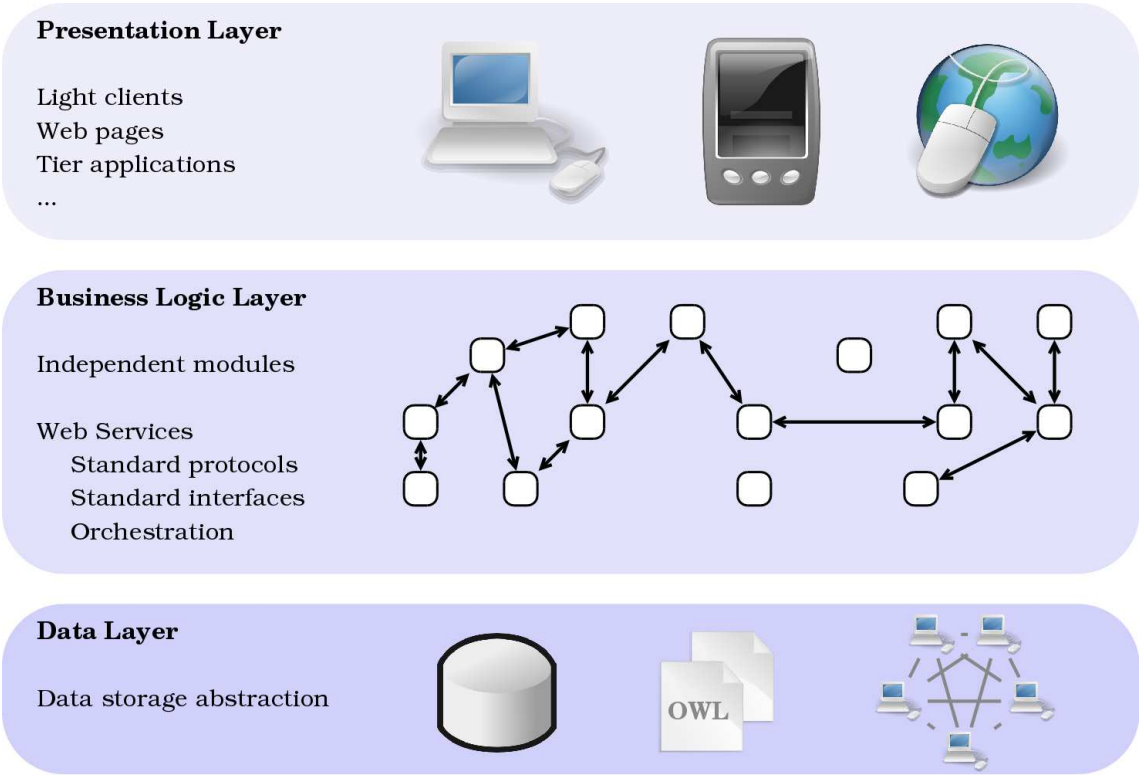


Figure D.1 – Schema representing a three-tier architecture.

Appendix E

XML Description of CAD Models

The XML file resulting from the analysis of the CAD model and template presented in figure E.1 is available in the source code E.1.

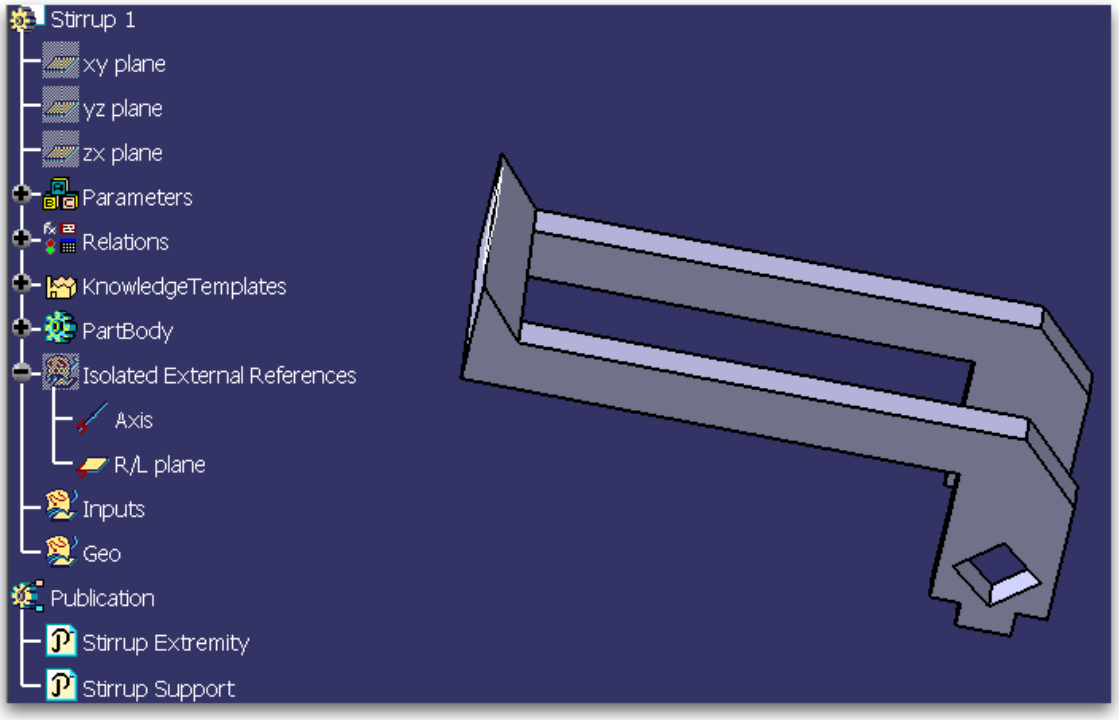


Figure E.1 – Screenshot of a CAD model under CATIA V5.

Source code E.1 – XML document generated by the application presented in section 11.3. It describes the template presented in figure E.1.

```
<ROOT>
    <PRODUCTS>
        <Product Name="PLMInfos">
            <MainProduct ItemName="C:\Dokumente␣und␣Einstellungen\kuhn\
                Desktop\THE␣template␣example\Trunk\Templates\Stirrup␣template
                .CATPart" ItemType="CATPart" ItemVersion="&lt;Version&gt;5&lt
                ;/Version&gt;&lt;Release&gt;19&lt;/Release&gt;&lt;ServicePack
                &gt;0&lt;/ServicePack&gt;&lt;HotFix&gt;0&lt;/HotFix&gt;"
                DisplayName="Stirrup␣template.CATPart" CN_PART_NUMBER="
                Stirrup␣1" CN_REVISION="" CN_DEFINITION="" CN_NOMENCLATURE=""
                 CN_DESCRIPTIONREF="" CN_SOURCE="unknown" CN_VOLUME="134561"
                CN_MASS="0,134561" CN_SURFACE="39781,2" TemplateID="2a968271-
                f0df-4a5a-afc8-30c57c15fc01" Length="180mm" Height="40mm" />
        </Product>
    </PRODUCTS>
    <PARTS>
```

```
<Part Name="Stirrup 1" InstanceName="Stirrup 1" StorageName="C:\
    Dokumente und Einstellungen\kuhn\Desktop\THE template example\
    Trunk\Templates\Stirrup template.CATPart">
    <Set Name="PartBody">
        <Feature Name="Pad.1" InternName="Pad.1" Type="Pad" />
        <Feature Name="Pocket.2" InternName="Pocket.2" Type="Pocket"
            />
        <Feature Name="Plane.12" InternName="GSMPlane.12" Type="
            GSMPlane" />
        <Feature Name="Point.1" InternName="GSMPoint.1" Type="
            GSMPoint" />
    </Set>
    <Set Name="Isolated External References">
        <Feature Name="Axis" InternName="GSMLine.4" Type="GSMLine" />
        <Feature Name="R/L plane" InternName="GSMPlane.11" Type="
            GSMPlane" />
    </Set>
    <Publications>
        <Publication Name="Stirrup Extremity" PublishedElement="
            GSMPlane.12" PublishedElementDisplayedName="Plane.12"
            PublishedElementType="GSMPlane" />
        <Publication Name="Stirrup Support" PublishedElement="??" />
    </Publications>
    <Relations>
        <Relation Name="RelationExpFct.9" DisplayedName="Formula.9: 
            PartBody\Pocket.2\Sketch.3\Offset.55\Offset=Height  + 45
            mm" body="Height  + 45mm">
            <In>
                <Param content="40" path="Height" />
            </In>
            <Out>
                <Param content="85" path="Stirrup 1\PartBody\Pocket
                    .2\Sketch.3\Offset.55\Offset" />
            </Out>
        </Relation>
        <Relation Name="RelationExpFct.10" DisplayedName="Formula.10:
            PartBody\Pocket.2\FirstLimit\Depth=Length  -14mm + 18mm"
             body="Length  -14mm + 18mm">
            <In>
                <Param content="180" path="Length" />
            </In>
            <Out>
                <Param content="184" path="Stirrup 1\PartBody\Pocket
                    .2\FirstLimit\Depth" />
            </Out>
        </Relation>
        <Relation Name="RelationExpFct.11" DisplayedName="Formula.11:
            PartBody\Pad.1\Sketch.1\Offset.137\Offset=Height " body=
            "Height ">
            <In>
                <Param content="40" path="Height" />
            </In>
            <Out>
                <Param content="40" path="Stirrup 1\PartBody\Pad.1\
                    Sketch.1\Offset.137\Offset" />
            </Out>
        </Relation>
        <Relation Name="RelationExpFct.12" DisplayedName="Formula.12:
            PartBody\Pad.1\Sketch.1\Offset.120\Offset=Length " body=
            "Length ">
            <In>
```

```
                    <Param content="180" path="Length" />
                </In>
                <Out>
                    <Param content="180" path="Stirrup 1\PartBody\Pad.1\
                        Sketch.1\Offset.120\Offset" />
                </Out>
            </Relation>
        </Relations>
        <Parameters>
            <Set Name="Parameters">
                <Param Name="Length" Value="180mm" />
                <Param Name="Height" Value="40mm" />
            </Set>
        </Parameters>
        <Templates>
            <Documents>
                <Template Name="Stirrup template">
                    <Input name="GSMLine.4" role="Axis" />
                    <Input name="GSMPlane.11" role="R/L plane" />
                    <Input name="LENGTH.2" role="Length" />
                    <Input name="LENGTH.3" role="Height" />
                </Template>
            </Documents>
            <UDF />
            <PwC />
        </Templates>
        <PLMInfos ItemName="C:\Dokumente und Einstellungen\kuhn\Desktop\
            THE template example\Trunk\Templates\Stirrup template.CATPart
            " ItemType="CATPart" ItemVersion="&lt;Version&gt;5&lt;/
            Version&gt;&lt;Release&gt;19&lt;/Release&gt;&lt;ServicePack&
            gt;0&lt;/ServicePack&gt;&lt;HotFix&gt;0&lt;/HotFix&gt;"
            DisplayName="Stirrup template.CATPart" CN_PART_NUMBER="
            Stirrup 1" CN_REVISION="" CN_DEFINITION="" CN_NOMENCLATURE=""
             CN_DESCRIPTIONREF="" CN_SOURCE="unknown" CN_VOLUME="134561"
            CN_MASS="0,134561" CN_SURFACE="39781,2" TemplateID="2a968271-
            f0df-4a5a-afc8-30c57c15fc01" Length="180mm" Height="40mm" />
    </Part>
</PARTS>
</ROOT>
```

Appendix F

XML-Schema for CAD Models Description

Source code F.1 – XML-Schema corresponding to the XML output from the CAD document analysis.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="
    qualified">
  <xs:element name="ROOT">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="PRODUCTS"/>
        <xs:element ref="PARTS"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="PRODUCTS">
    <xs:complexType>
      <xs:sequence>
        <xs:element maxOccurs="unbounded" ref="Product"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="PARTS">
    <xs:complexType>
      <xs:sequence>
        <xs:element maxOccurs="unbounded" ref="Part"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="Product">
    <xs:complexType>
      <xs:sequence>
        <xs:element minOccurs="0" ref="MainProduct"/>
        <xs:element minOccurs="0" ref="Children"/>
        <xs:element minOccurs="0" ref="DocumentTemplate"/>
        <xs:element minOccurs="0" ref="Relations"/>
        <xs:sequence minOccurs="0">
          <xs:element ref="Constraints"/>
          <xs:element ref="PLMInfos"/>
        </xs:sequence>
      </xs:sequence>
      <xs:attribute name="InstanceName"/>
      <xs:attribute name="Name" use="required"/>
      <xs:attribute name="StorageName"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="MainProduct">
    <xs:complexType>
      <xs:sequence>
        <xs:element maxOccurs="unbounded" ref="Children"/>
      </xs:sequence>
      <xs:attribute name="CN_DEFINITION" use="required"/>
```

```
      <xs:attribute name="CN_DESCRIPTIONREF" use="required"/>
      <xs:attribute name="CN_MASS" use="required"/>
      <xs:attribute name="CN_NOMENCLATURE" use="required"/>
      <xs:attribute name="CN_PART_NUMBER" use="required"/>
      <xs:attribute name="CN_REVISION" use="required"/>
      <xs:attribute name="CN_SOURCE" use="required" type="xs:NCName"/>
      <xs:attribute name="CN_SURFACE" use="required" type="xs:integer"/>
      <xs:attribute name="CN_VOLUME" use="required"/>
      <xs:attribute name="DisplayName" use="required"/>
      <xs:attribute name="ItemName" use="required"/>
      <xs:attribute name="ItemType" use="required" type="xs:NCName"/>
      <xs:attribute name="ItemVersion" use="required"/>
      <xs:attribute name="TemplateID" use="required"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="DocumentTemplate">
    <xs:complexType>
      <xs:sequence>
        <xs:element minOccurs="0" ref="Template"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="Template">
    <xs:complexType>
      <xs:sequence>
        <xs:element maxOccurs="unbounded" ref="Input"/>
      </xs:sequence>
      <xs:attribute name="Name" use="required"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="Input">
    <xs:complexType>
      <xs:attribute name="name" use="required" type="xs:NCName"/>
      <xs:attribute name="role" use="required" type="xs:NCName"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="Constraints">
    <xs:complexType>
      <xs:sequence minOccurs="0">
        <xs:element maxOccurs="unbounded" ref="Constraint"/>
        <xs:element maxOccurs="unbounded" ref="ConstraintSet"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="ConstraintSet">
    <xs:complexType>
      <xs:sequence>
        <xs:element maxOccurs="unbounded" ref="Constraint"/>
      </xs:sequence>
      <xs:attribute name="Name" use="required"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="Part">
    <xs:complexType>
      <xs:sequence>
        <xs:element minOccurs="0" maxOccurs="unbounded" ref="Set"/>
        <xs:sequence minOccurs="0">
          <xs:element ref="ImportLinks"/>
          <xs:element ref="ContextLinks"/>
        </xs:sequence>
        <xs:element minOccurs="0" ref="Publications"/>
```

```
      <xs:element minOccurs="0" ref="ExtRef"/>
      <xs:element minOccurs="0" ref="ExtParam"/>
      <xs:element minOccurs="0" ref="Relations"/>
      <xs:element minOccurs="0" ref="Parameters"/>
      <xs:sequence minOccurs="0">
        <xs:element ref="Templates"/>
        <xs:element ref="PLMInfos"/>
      </xs:sequence>
    </xs:sequence>
    <xs:attribute name="InstanceName" use="required"/>
    <xs:attribute name="Name" use="required"/>
    <xs:attribute name="StorageName" use="required"/>
  </xs:complexType>
</xs:element>
<xs:element name="ImportLinks">
  <xs:complexType>
    <xs:sequence>
      <xs:element maxOccurs="unbounded" ref="Link"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="ContextLinks">
  <xs:complexType>
    <xs:sequence>
      <xs:element minOccurs="0" maxOccurs="unbounded" ref="Link"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="Publications">
  <xs:complexType>
    <xs:sequence>
      <xs:element maxOccurs="unbounded" ref="Publication"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="Publication">
  <xs:complexType>
    <xs:attribute name="Name" use="required"/>
    <xs:attribute name="PublishedElement" use="required"/>
    <xs:attribute name="PublishedElementDisplayedName"/>
    <xs:attribute name="PublishedElementType" type="xs:NCName"/>
  </xs:complexType>
</xs:element>
<xs:element name="ExtRef">
  <xs:complexType>
    <xs:sequence>
      <xs:element maxOccurs="unbounded" ref="ref"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="ref">
  <xs:complexType>
    <xs:attribute name="Name" use="required"/>
    <xs:attribute name="Target"/>
  </xs:complexType>
</xs:element>
<xs:element name="ExtParam">
  <xs:complexType/>
</xs:element>
<xs:element name="Parameters">
  <xs:complexType>
```

```
      <xs:sequence>
        <xs:element maxOccurs="unbounded" ref="Set"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="Templates">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="Documents"/>
        <xs:element ref="UDF"/>
        <xs:element ref="PwC"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="Documents">
    <xs:complexType/>
  </xs:element>
  <xs:element name="UDF">
    <xs:complexType/>
  </xs:element>
  <xs:element name="PwC">
    <xs:complexType/>
  </xs:element>
  <xs:element name="Children">
    <xs:complexType>
      <xs:sequence>
        <xs:element minOccurs="0" maxOccurs="unbounded" ref="Children"/>
        <xs:element minOccurs="0" maxOccurs="unbounded" ref="Part"/>
        <xs:element minOccurs="0" ref="Product"/>
      </xs:sequence>
      <xs:attribute name="BBVolume"/>
      <xs:attribute name="CN_DEFINITION"/>
      <xs:attribute name="CN_DESCRIPTIONREF"/>
      <xs:attribute name="CN_MASS"/>
      <xs:attribute name="CN_NOMENCLATURE"/>
      <xs:attribute name="CN_PART_NUMBER"/>
      <xs:attribute name="CN_REVISION"/>
      <xs:attribute name="CN_SOURCE" type="xs:NCName"/>
      <xs:attribute name="CN_SURFACE"/>
      <xs:attribute name="CN_VOLUME"/>
      <xs:attribute name="DisplayName"/>
      <xs:attribute name="Height" type="xs:NMTOKEN"/>
      <xs:attribute name="ItemName"/>
      <xs:attribute name="ItemType" type="xs:NCName"/>
      <xs:attribute name="ItemVersion"/>
      <xs:attribute name="Length" type="xs:NMTOKEN"/>
      <xs:attribute name="Mass1"/>
      <xs:attribute name="Riser-Config" type="xs:integer"/>
      <xs:attribute name="Riser_config" type="xs:NCName"/>
      <xs:attribute name="Support_config" type="xs:NCName"/>
      <xs:attribute name="TOOLING-CONFIGURATION" type="xs:NCName"/>
      <xs:attribute name="TemplateID"/>
      <xs:attribute name="Thickness" type="xs:NMTOKEN"/>
      <xs:attribute name="Thikness" type="xs:NMTOKEN"/>
      <xs:attribute name="Width" type="xs:NMTOKEN"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="Relations">
    <xs:complexType>
      <xs:choice minOccurs="0" maxOccurs="unbounded">
        <xs:element ref="DesignTable"/>
```

```
      <xs:element ref="Relation"/>
    </xs:choice>
  </xs:complexType>
</xs:element>
<xs:element name="DesignTable">
  <xs:complexType>
    <xs:attribute name="ConfigurationName" use="required"/>
    <xs:attribute name="ConfigurationValue" use="required" type="xs:integer
        "/>
    <xs:attribute name="CurrentVaue" use="required"/>
    <xs:attribute name="DisplayName" use="required" type="xs:NCName"/>
    <xs:attribute name="Name" use="required" type="xs:NCName"/>
    <xs:attribute name="SheetDocument" use="required"/>
  </xs:complexType>
</xs:element>
<xs:element name="Relation">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="In"/>
      <xs:element ref="Out"/>
    </xs:sequence>
    <xs:attribute name="DisplayedName"/>
    <xs:attribute name="Name" use="required" type="xs:NCName"/>
    <xs:attribute name="body" use="required"/>
  </xs:complexType>
</xs:element>
<xs:element name="In">
  <xs:complexType>
    <xs:sequence>
      <xs:element maxOccurs="unbounded" ref="Param"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="Out">
  <xs:complexType>
    <xs:sequence>
      <xs:element minOccurs="0" maxOccurs="unbounded" ref="Param"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="Constraint">
  <xs:complexType>
    <xs:attribute name="Name" use="required" type="xs:NCName"/>
    <xs:attribute name="element1" use="required"/>
    <xs:attribute name="element2"/>
    <xs:attribute name="status" type="xs:NCName"/>
  </xs:complexType>
</xs:element>
<xs:element name="PLMInfos">
  <xs:complexType>
    <xs:attribute name="BBVolume"/>
    <xs:attribute name="CN_DEFINITION" use="required"/>
    <xs:attribute name="CN_DESCRIPTIONREF" use="required"/>
    <xs:attribute name="CN_MASS" use="required"/>
    <xs:attribute name="CN_NOMENCLATURE" use="required"/>
    <xs:attribute name="CN_PART_NUMBER" use="required"/>
    <xs:attribute name="CN_REVISION" use="required"/>
    <xs:attribute name="CN_SOURCE" use="required" type="xs:NCName"/>
    <xs:attribute name="CN_SURFACE" use="required"/>
    <xs:attribute name="CN_VOLUME" use="required"/>
    <xs:attribute name="DisplayName" use="required"/>
```

```
      <xs:attribute name="Height" type="xs:NMTOKEN"/>
      <xs:attribute name="ItemName" use="required"/>
      <xs:attribute name="ItemType" use="required" type="xs:NCName"/>
      <xs:attribute name="ItemVersion" use="required"/>
      <xs:attribute name="Length" type="xs:NMTOKEN"/>
      <xs:attribute name="Mass1"/>
      <xs:attribute name="Riser-Config" type="xs:integer"/>
      <xs:attribute name="Riser_config" type="xs:NCName"/>
      <xs:attribute name="TOOLING-CONFIGURATION" type="xs:NCName"/>
      <xs:attribute name="TemplateID"/>
      <xs:attribute name="Thickness" type="xs:NMTOKEN"/>
      <xs:attribute name="Thikness" type="xs:NMTOKEN"/>
      <xs:attribute name="Width" type="xs:NMTOKEN"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="Set">
    <xs:complexType>
      <xs:choice>
        <xs:element minOccurs="0" maxOccurs="unbounded" ref="Feature"/>
        <xs:element minOccurs="0" maxOccurs="unbounded" ref="Param"/>
      </xs:choice>
      <xs:attribute name="Name" use="required"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="Feature">
    <xs:complexType>
      <xs:attribute name="InternName" use="required" type="xs:NMTOKEN"/>
      <xs:attribute name="Name" use="required"/>
      <xs:attribute name="Type" use="required" type="xs:NCName"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="Link">
    <xs:complexType>
      <xs:attribute name="Target" use="required"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="Param">
    <xs:complexType>
      <xs:attribute name="Name"/>
      <xs:attribute name="Value"/>
      <xs:attribute name="content"/>
      <xs:attribute name="path"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```