

# Ontology-driven and rules-based system for management and pricing of family of product.

**Vincent Dedeban**

Master of Science in Computer Science

Submission date: June 2007

Supervisor: Jon Atle Gulla, IDI

Co-supervisor: Darijus Strasunskas, IDI



## Problem Description

Currently, Sales department at DNV Software division is using a catalogue of products and each responsible person has locally implemented package composition rules often using MS Excel spreadsheets (i.e. stand-alone solution). A certain need is to have a centralized system to manage composition and pricing of packages. Therefore, it is necessary to find a flexible way to define possible configurations of the packages that can be integrated and sold. The task is to create ontology-driven and rules-based system for management and pricing of family of product. Consequent sub-tasks are as follows. Investigate existing semantic technologies applicable for the task, their pros and cons; analyse state-of-the-art in management of product families; implement and validate the method based on the case provided by DNV Software.

Assignment given: 09. January 2007  
Supervisor: Jon Atle Gulla, IDI



Norwegian University of Science and Technology  
Faculty of Information Technology, Mathematics and Electrical  
Engineering  
Department of Computer and Information Science

Master of Science Thesis

# **Ontology-driven and rules-based system for management and pricing of family of product**

by

**Vincent DEDEBAN**

Supervisor: Jon Atle Gulla Co-Supervisor: Darijus Strasunskas

Trondheim, 2007



---

# Abstract

---

This report presents an approach to product family management using semantic technologies. There is a need for more flexible configuration management and pricing, explicitly representing domain knowledge and configuration rules.

The thesis investigates requirements, needs and current issues to support product family in order to figure out the best candidate technologies. Ontology and rules are chosen to carry out the project. Our approach is applied to a case study provided by Det Norske Veritas Software, a company offering solutions for maritime, offshore and process industries. The contacts with the company have strengthened our work as the client helped us to clarify the problem and validate the suggested method.

The report discusses pros and cons of semantic technologies applied in this context. Semantic technologies and rules help to create highly flexible system that allows supporting product family engineering. Our work highlights consistency improvements and redundancy diminution in features models.





---

# Preface

---

This Master's thesis was submitted to the Norwegian University of Science and Technology (NTNU), Department of Computer and Information Science. The workload in this thesis is 30 European Credit Transfer System (ECTS) credits. It will be part of my year of study as an erasmus student at the NTNU.

It is the result of a thesis problem given by Det Norske Veritas AS. The work was carried out during the spring semester 2007, starting in January 2007 and ending in June 2007.

I would like to thank Darijus Strasunskas my co-supervisor and co-author on the paper "An Ontology-Centric Approach for Flexible Configuration and Pricing of Product Families" presented at the First International Workshop on Semantic Technology Adoption in Business. Your advises and comments have been very valuable throughout the thesis. Furthermore I would like to thank Espen Wien, Ketil Aamnes, and Kjell Tangen from Det Norske Veritas for their help.



---

# Contents

---

<b>Abstract</b>	<b>iii</b>
<b>Preface</b>	<b>v</b>
<b>Contents</b>	<b>i</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Background . . . . .	1
1.2 Project task and goals . . . . .	2
1.3 Constraints . . . . .	3
1.4 Structure of the thesis . . . . .	3
<b>2 Settings and background</b>	<b>5</b>
2.1 Working settings . . . . .	5
2.1.1 Layered architecture . . . . .	5
2.1.2 Brix . . . . .	6
2.2 Product Family . . . . .	9
2.2.1 Definition . . . . .	9
2.2.2 Variability dimensions . . . . .	9
2.2.3 Variability level . . . . .	10
2.2.4 The product family engineering . . . . .	11
2.2.5 Summary . . . . .	14
2.3 Requirements and needs . . . . .	14
2.3.1 Functional requirements . . . . .	14
2.3.2 Non functional requirement . . . . .	15
2.3.3 Needs . . . . .	15
2.4 Restrictions . . . . .	16
2.5 Summary . . . . .	17
<b>3 State of the art</b>	<b>19</b>
3.1 Current technical solution . . . . .	19

3.1.1	BigLeve software Gears . . . . .	19
3.1.2	Pure::Variant . . . . .	20
3.1.3	Kumbang . . . . .	20
3.2	Unsolved problems/Current issues . . . . .	20
3.3	Summary . . . . .	22
<b>4</b>	<b>Candidate technologies</b>	<b>23</b>
4.1	Semantic web . . . . .	23
4.1.1	Component . . . . .	23
4.1.2	Motivation to use semantic web technologies . . . . .	25
4.2	Ontology in more details . . . . .	26
4.2.1	Elements of an ontology . . . . .	26
4.2.2	Ontology types . . . . .	27
4.2.3	Web ontology languages . . . . .	28
4.3	Rule based system . . . . .	31
4.3.1	Theory . . . . .	31
4.3.2	Motivation to use rule-based system . . . . .	33
4.4	Software development using semantic technologies . . . . .	34
4.4.1	Semantic web in systems and software engineering . . . . .	34
4.4.2	Semantic web enabled software engineering . . . . .	35
4.5	Summary . . . . .	36
<b>5</b>	<b>Product family pricing calculator</b>	<b>39</b>
5.1	System definition . . . . .	39
5.1.1	Actors . . . . .	39
5.1.2	Use cases . . . . .	40
5.2	System overview . . . . .	42
5.2.1	Components . . . . .	42
5.2.2	Product selection . . . . .	43
5.2.3	Pricing . . . . .	44
5.2.4	Package management . . . . .	45
5.3	Technical overview . . . . .	46
5.3.1	System design . . . . .	46
5.3.2	Implementation . . . . .	50
5.4	Summary . . . . .	57
<b>6</b>	<b>Evaluation of the approach and system</b>	<b>59</b>
6.1	Objectives . . . . .	59
6.2	Results . . . . .	60
6.2.1	Incremental development and evaluation . . . . .	60
6.2.2	Final prototype validation . . . . .	61
6.3	Summary . . . . .	69

<b>7</b>	<b>Conclusions and future work</b>	<b>71</b>
7.1	Discussion and future work . . . . .	71
7.2	Conclusion . . . . .	72
	<b>Bibliography</b>	<b>75</b>
	<b>Appendices</b>	<b>79</b>
<b>A</b>	<b>Thesis resources and constraints</b>	<b>81</b>
A.1	Resources . . . . .	81
A.2	Economy . . . . .	81
A.3	Thesis lifespan . . . . .	81
<b>B</b>	<b>Detailed use cases</b>	<b>83</b>
B.1	UC:Create offer . . . . .	83
B.2	UC>Delete offer . . . . .	84
B.3	UC:Save offer . . . . .	84
B.4	UC:Open offer . . . . .	85
B.5	UC:Select offer . . . . .	86
B.6	UC:Modify offer . . . . .	86
B.7	UC:Add package to offer . . . . .	87
B.8	UC:Price package . . . . .	87
B.9	UC:Select package . . . . .	88
B.10	UC:Generate report . . . . .	88
B.11	UC:Manage Users . . . . .	89
	B.11.1 Create user . . . . .	89
	B.11.2 Delete user . . . . .	90
	B.11.3 Manage the roles of one user . . . . .	90
B.12	UC:Create package . . . . .	91
B.13	UC>Delete package . . . . .	91
B.14	UC:Update package . . . . .	92
<b>C</b>	<b>Components interaction</b>	<b>93</b>
C.1	Price a package . . . . .	93
C.2	Create a package . . . . .	93
<b>D</b>	<b>Guide to choose a Semantic Web Toolkit</b>	<b>97</b>
D.1	Comparison Criteria . . . . .	97
	D.1.1 License . . . . .	97
	D.1.2 API-Paradigm . . . . .	97
	D.1.3 Query-Languages . . . . .	98
	D.1.4 Model Storage . . . . .	98
	D.1.5 Supported Databases . . . . .	98

D.1.6	Supported Serialization Formats . . . . .	98
D.1.7	Reasoning Support . . . . .	98
D.1.8	RDF Server . . . . .	99
D.1.9	Other Features . . . . .	99
D.2	Toolkit Comparison . . . . .	99
D.3	Conclusion . . . . .	100
<b>E</b>	<b>Prototype user interface</b>	<b>101</b>
E.1	Web site component . . . . .	101
E.2	Package management tools . . . . .	101
E.2.1	Package creation tool . . . . .	102
E.2.2	Ontology editor . . . . .	103
<b>F</b>	<b>Tests</b>	<b>109</b>
F.1	Package selection . . . . .	110
F.2	Price calculation . . . . .	111
F.2.1	Normal pricing calculation . . . . .	111
F.2.2	Web site adaption to a change in pricing scheme . . . . .	112
F.2.3	Customize pricing options test . . . . .	113
F.3	Package creation . . . . .	113
F.3.1	Consistency of a new package . . . . .	113
F.3.2	Variability support during the configuration . . . . .	116
F.4	Family creation . . . . .	117
F.4.1	Family creation test . . . . .	117
F.4.2	Feature type support . . . . .	118
F.4.3	Family architecture consistency . . . . .	119
F.4.4	Redundancy test . . . . .	120
	<b>List of Symbols and Abbreviations</b>	<b>121</b>
	<b>List of Figures</b>	<b>122</b>
	<b>List of Tables</b>	<b>124</b>

# Chapter 1

---

## Introduction

---

### 1.1 Background

Mass Customization is the new paradigm that replaces mass production, which is no longer suitable for today's turbulent markets, growing product variety, and opportunities for commerce [4]. Mass customization proactively manages product variety in the environment of rapidly evolving markets and products. Mass customizers can customize products quickly for individual customers or for niche markets. Using the same principles, mass customizers can Build-to-Order both customized products and standard products without forecasts, inventory, or purchasing delays. This concept is the new battlefield of lot of software engineering companies to adapt quickly to a user specific demand. In this context the rationale of developing products families with respect to match customer desires have been well recognized in both industry and academia.

Build a family of products<sup>1</sup> is an efficient way to adapt to the products diversity; it also decreases risks by using approved components in the firm. The rationale to build family of products can be viewed as the combination of the knowledge about the market and of business rules to combine the products. Computers systems can obviously support this rationale.

An innovative way to do it should be an ontology-driven and rules-based system. The two technologies match with the problem definition. The business rules are a common field of research because much of the backlog in the IT departments of organization is due to the need for frequent updates of the business logic in existing applications. An externalization of the business rules maintenance can avoid translation errors in converting the policy changes into programming code, and save the IT staff from making constant updates. Ontologies are in both com-

---

<sup>1</sup>Products family, Family of products and Product line are used interchangeably in the report with the same meaning.

puter science and information science a data model that represents a domain and are used to reason about the objects in that domain and the relations between them. Main advantages of semantic technologies adoption in software engineering are as follows: reusability and extensibility of data models, improvements in data quality, discovery and automated execution of workflows.

The master thesis presented in this report has been done at DNV Software that is the commercial software house of DNV (Det Norske Veritas). The company is a market leader in software development for the maritime, offshore, and process industries. In order to support mass customization in its way to sell products DNV is trying to implant a branding structure (i.e. family of products). Motivation for the work is to facilitate and centralize configuration and pricing of product families by employing semantic technologies.

## 1.2 Project task and goals

The following paragraphs are taken from the project problem text. Currently, Sales department at DNV Software division is using a catalogue of products and each responsible person has locally implemented package composition rules often using MS Excel spreadsheets (i.e. stand-alone solution). The information related to the offers are store using the Salesforce software<sup>2</sup>. A certain need is to have a centralized system to manage composition and pricing of packages. Therefore, it is necessary to find a flexible way to define possible configurations of the packages that can be integrated and sold. The task is to create ontology-driven and rules-based system for management and pricing of family of product. Consequent sub-tasks are as follows. Investigate existing semantic technologies applicable for the task, their pros and cons; analyze state-of-the-art in management of product families; implement and validate the method based on the case provided by DNV Software.

This thesis deals with concepts such as product family, rules, semantic technologies and their interconnections. The work done present semantic technology adoption and deployment in practice. The case study tests the hypothesis that semantic web and rules can help to manage and price products families and tries to identify problems solved and caused by the usage of the approach presented here to support the family of products. We used the case study as a way to help identify questions, develop measures concerning the semantic technologies and their adoption. We hope that our work will serve to safeguard future research/development as a part of the results of this master thesis has been published as post-proceedings of the First International Workshop on Semantic Technology Adoption in Business – STAB’07 [11].

---

<sup>2</sup><http://www.salesforce.com/products/>



## 1.3 Constraints

This section will present the initial motivations of DNV concerning the Master Thesis. The thesis' wishes are turned into requirements later (see chap: 2.3). DNV would like the result of the master thesis to:

- Be used by the sales people in DNV Software to create the pricing offers to customers;
- Help the salesperson to select "the right" offer according to the branding structure (packages and products);
- Incorporate the rules described in the pricing scheme;
- Be developed using the DNV technology including the rule engine
- Be web based;
- Be accessed from the Internet using the DNV's security technology;
- Be role based with one role for a super user and another for a sales person;
- Have an interface to the CRM software used by DNV (SalesForce).

## 1.4 Structure of the thesis

This document is structured in 7 parts that enable to follow and to understand the research process accomplishment. The 7 parts are as follows:

1. Introduction. this chapter presents the project and the context of the project.
2. Settings and background. This part presents the product family concept and technological environment in Det Norske Veritas. Goal of this part is to identify requirements. The requirements are the basis for the rest of the project.
3. State of the art in managing family of products. This chapter tries to identify the current issues of the existing solutions.
4. Candidate technologies. This chapter investigates the motivations to use semantic and business rules technologies to support the management and pricing of the family of products.
5. Presentation of our system. Based on the previous chapters we present the system we designed to answer the identified problems.

6. Evaluation of the approach and system chapter. In this chapter we evaluate the results generated by the case study.
7. Conclusions and future work.
8. Appendices. They present more detailed information than in the core of the report. The appendices comport informations about the thesis resources and constraints, the use cases, the interactions between the components within the system, a guide to choose a semantic toolkit, screen shots of the final prototype graphical interface, and the detailed tests done to evaluate our approach.

## Chapter 2

---

# Settings and background

---

The goal of this chapter is to present the technical context in DNV and the product family concept. The chapter is concluded with a set of requirements, needs and restrictions for the future system.

### 2.1 Working settings

Working with DNV is a unique chance to strengthen our work as the client helped us to validate our concepts. Nevertheless this collaboration implies also to follow the company way of working. All software produced at DNV Software follow the company's architecture. All the softwares have to use the tools provided by the DNV Software's BRIX<sup>1</sup> technology. In order to define requirements we need to understand two essential architectural components:

- *The layered architecture* that is in use in all the software (even BRIX).
- *Brix*. A framework that serves as a basis for all the software developed in DNV.

#### 2.1.1 Layered architecture

DNV Software has developed a high quality layered architecture (see figure: 2.1) for their products. The architecture is based on three main layers and a data layer. Three-Layered Services Application, as presented here, is basically a relaxed three-layered architecture. The three layers are:

---

<sup>1</sup><http://www.dnv.com/software/workflow/brixFoundation.asp>

- **Presentation.** The presentation layer provides the application's user interface (UI). Typically, this involves the use of Windows Forms for smart client interaction, and ASP.NET technologies for browser-based interaction.
- **Business.** The business layer implements the business functionality of the application. The domain layer is typically composed of a number of components implemented using one or more .NET - enabled programming languages.
- **Data** The data layer provides access to external systems such as databases.

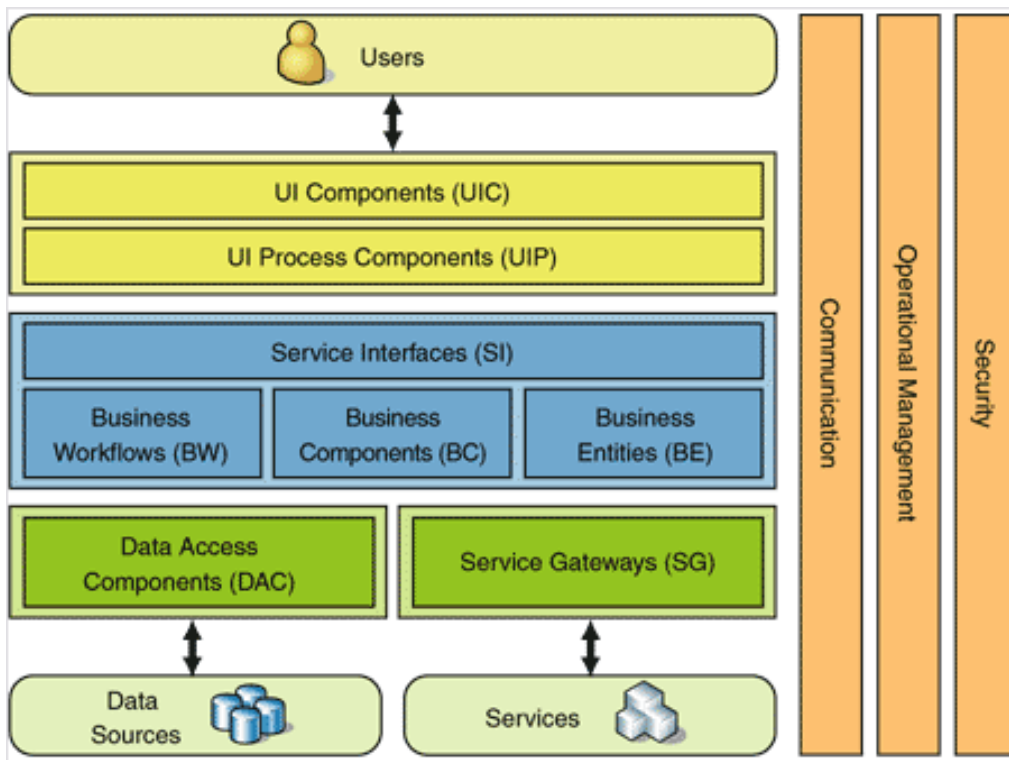


Figure 2.1: DNV's layered architecture

### 2.1.2 Brix

Brix is DNV Software's framework providing solutions that handle the life cycle of client industry knowledge. Brix enables to capture, spread, and improve industry knowledge by providing a portal to the best engineering practice. Brix combines how the client's business processes and business rules with the services needed to fulfill the business objectives. Brix can be considered as a set of components like: Brix Explorer, Brix Project Manager, Brix security etc. The idea behind Brix is to provide the tools that have to be used to guarantee a high quality of

the software. Brix is used in all of the new DNV's applications. The figure 2.2 describes the different components and their interrelations.

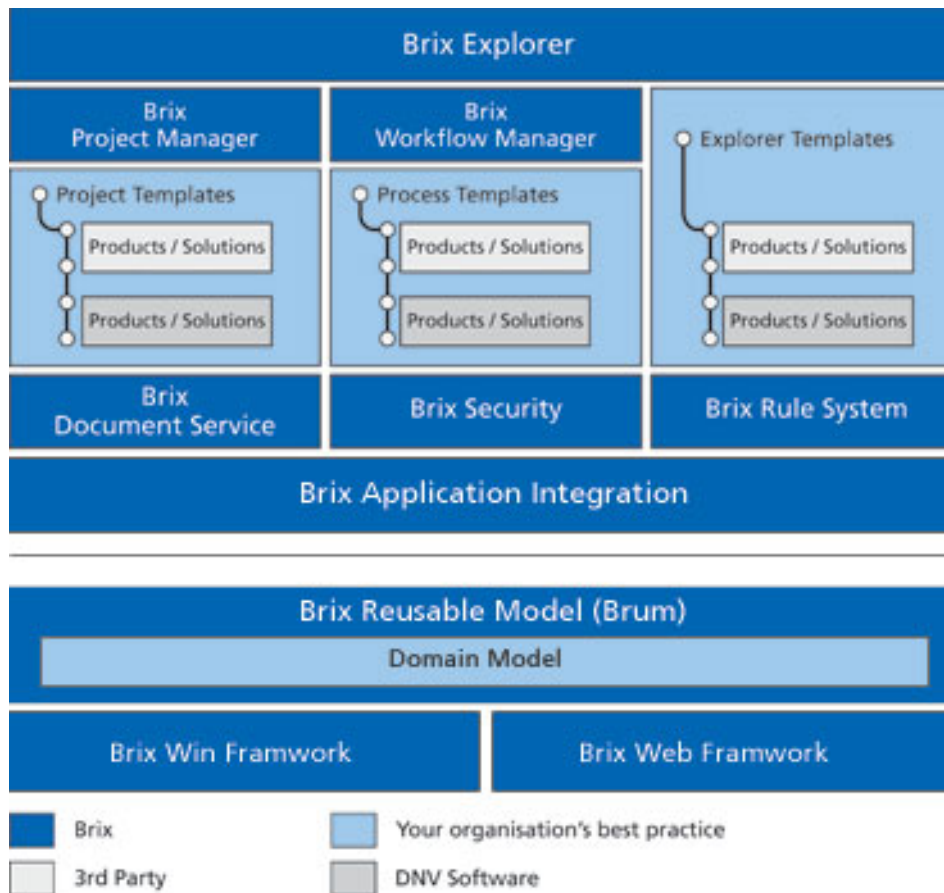


Figure 2.2: Brix framework organization

The two main components are a workflow process manager and a rules engine. The workflow manager enables to define complex process in order to coordinate people, applications and information. Using the Brix workflow manager, anyone can create a business process (see figure: 2.3). A business process is defined as a set of activity; each activity has a responsible, dependencies to others activities, and dataflow between activities. Each activity is carried out by performing some actions that use services in applications.

The Brix rules engine enables to externalize the logic of a program into a rule base (set of rules). The Brix rule engine has the ability to do a backward chaining. The backward chaining allows answering to complex question. The backward chaining is a simple mechanism in fact it starts with a list of goals (or a hypothesis) and works backwards to see if there are data available that will support any of these goals. An inference engine using backward chaining would

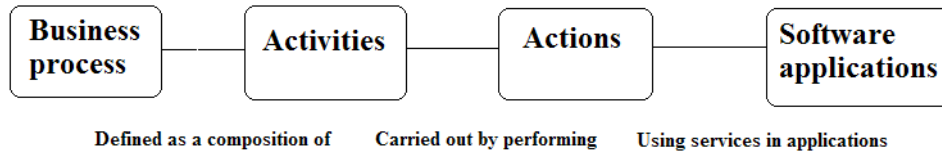


Figure 2.3: Business process for BRIX Workflow

search the inference rules until it finds one which has a *Then* clause that matches a desired goal. If the *If* clause of that inference rule is not known to be true, then it is added to the list of goals (In order for a goal to be confirmed the data that confirms this new rule must be provided). For example the rules engine (see figure 2.4) can be used to define the Tic-Tac-Toe game logic.

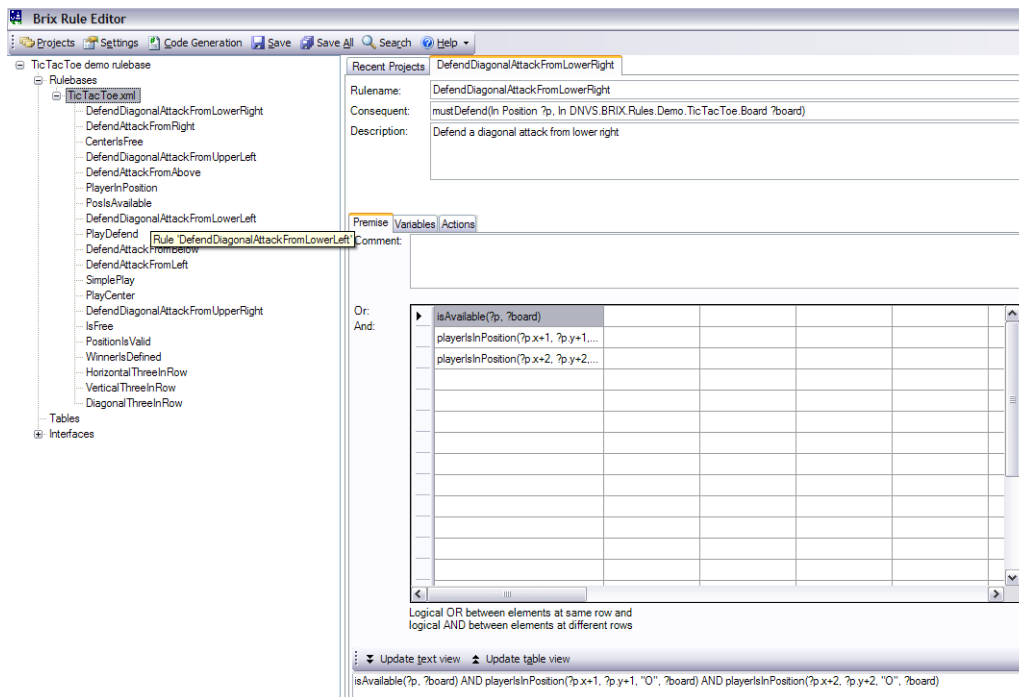


Figure 2.4: Screen shot of BRIX Rule Editor

The collaboration with DNV offers the opportunity to use a robust industrial strength. It offers also the opportunity to use tools such as the rule engine to implement our system.

## 2.2 Product Family

Goal of this section is to define the concept of product family. It will serve as a basis to find some requirements, needs, and restrictions for a system that has to manage product family.

### 2.2.1 Definition

A *product family* can be defined as a method that creates an underlying architecture of an organizations product. It provides an architecture that is based on *commonality* and *similarity*. The various products can be derived from the basic product family, which creates the opportunity to reuse and differentiate on products in the family. The *family of products* concept came from the factories to the software industry. The goal is to minimize the cost of a new product development, instead of developing the product from scratch; the components are reused to assemble a new product. All the products within a family are characterized by some common points but also by their differences (called *variance*). The management of this variance is a big issue to carry out the management of families of products. For example cars are made using common elements such as wheels, but they can have differences (different motor, air conditioned...). One difficulty to create a family of products is to choose some parameters to define the family.

This paradigm is not new it was first presented by David Parnas [26] in 1976, it has been studied since the end of the 70's but it has become more and more important with projects like ESAPS [1], CAFE [2], FAMILIES [3], and with the conferences SPLC (Software ProductLine Conference) and PFE (Product Family Engineering). In the literature exists a consensus on the definitions of a family of products. A family of products is a set of products with common properties that respond to a specific domain. A domain is characterized by a set of concepts and terms understandable by the users of this sector. A family of products is characterized by two concepts [33]:

- *Variability*. The variability gathers all the properties that differ within the members of the family.
- *Commonality* . The commonality gathers all the properties that are true for all the members of the family.

Commonality and variability are the central concepts in the family of products.

### 2.2.2 Variability dimensions

In this section, we discuss more in detail the variability concept than the one of commonality. This is justified by the fact that the management of variability

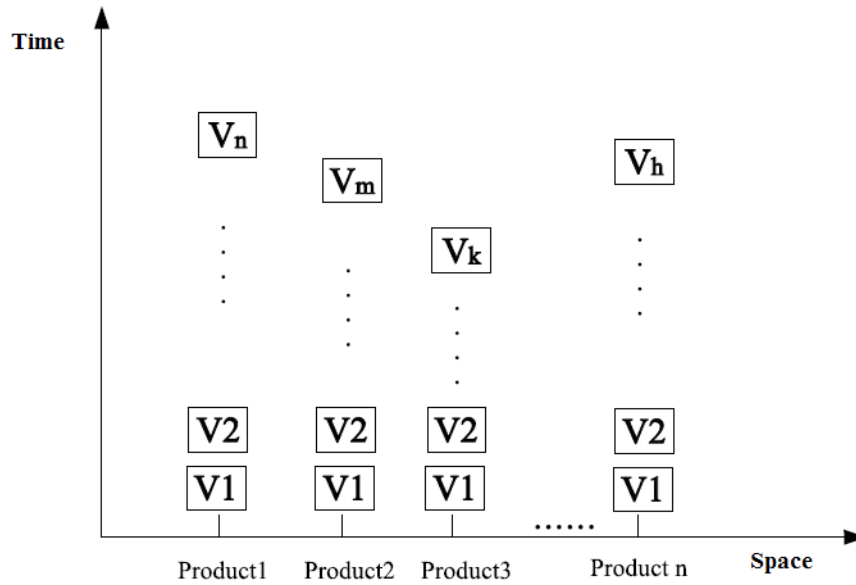


Figure 2.5: Dimension of variability in Product Family

requires more effort than the one of the commonality. In fact variability asks to be identified, but mechanisms for the variability management have to be done. Even if the Family of products is a new paradigm, the management of variability is not a new problem and several techniques of conception and programming allow managing it. The particularity of the variability in the family of products is that the variability must be specified explicitly and it is part of the Family of products architecture. In the family of products the variability has two dimensions: *Time* and *Space* [9, 20](see figure: 2.5).

- *Time*. The variation in time represents the variation of one product from one version to one other.
- *Space*. The variation in the space represents the variation between products of the same family.

### 2.2.3 Variability level

The variation points can be used to model different level of abstraction; for example the concept can be used to distinguish between product capabilities, the operating environment, domain technologies and implementation techniques. While product capabilities are general terms that also customers can understand and select the desired functionality from, implementation techniques are usually hidden from customers and used by application engineers that implement



products or product artefacts. Therefore, mappings between features on the diverse levels can be modelled through taxonomic and compositional relations and dependencies.

### 2.2.4 The product family engineering

The family of products concept is widely used in the industry because it allows developing a lot of products quite fast. As it is defined in the literature [12] the product family engineering is based on three phases:

- The *product management*.
- The *domain engineering*.
- The *product engineering*.

#### Product management

The *product management* phase consists in defining the scope of the product family regarding some economical criteria. At the end of this phase we know what should and should not be inside the product family.

#### Domain engineering

The *domain engineering* consists of developing and constructing the assets (for example an asset can be an element that allows developing software, for example a specification document, models, code etc.) that will be reused for the construction of products. It is a development for the reutilization. The *domain engineering* consists in three distinct steps [12]: analysis, conception and establishment of the domain. The goal of the analysis of the domain is to study the domain of the family of products to identify the commonality and the variability between the products. It exists several methods for the domain analysis, the more known is FODA [21]. The domain in FODA is described in a model of characteristics (a characteristic is called feature), specified in the form of a tree of which the vertex represent the domain characteristics and the arches specify composition links between the characteristics. FODA distinguishes different characteristics or type for the feature (see section: 2.2.4). The Figure 2.6 shows a model example of characteristic FODA of a family of products of cars. Each characteristic in the diagram corresponds to a concept of the domain. The obligatory characteristics are represented by rectangles with full circles, while the optional characteristics are represented by rectangles with empty circles. The characteristic Air-conditioning in the model FODA of this figure is optional. There exist three types of motors in the example family: electric, gasoline or diesel; this is specified by an alternative characteristic Motor with three characteristic Electric variants, Gasoline,

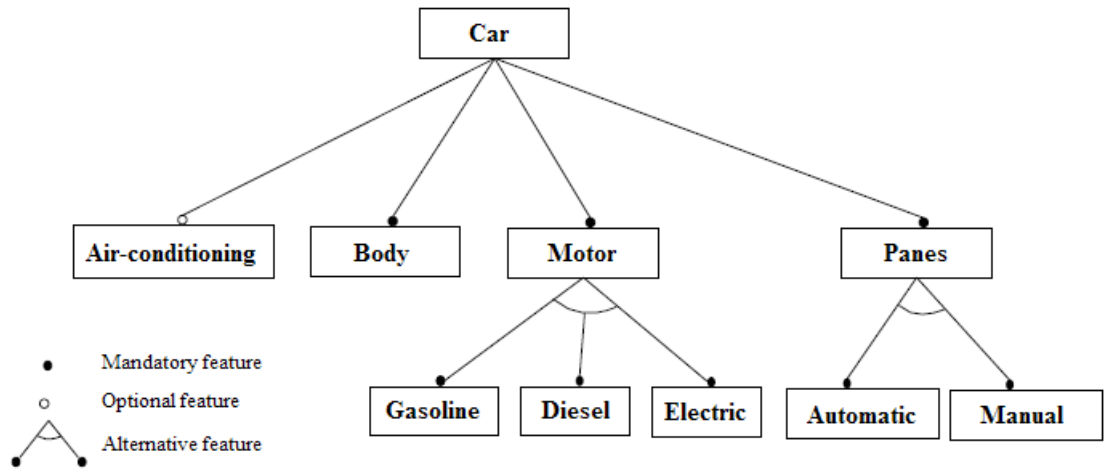


Figure 2.6: Example of feature model with FODA

and Diesel. An alternative characteristic is represented by a circle arch through the arches of the characteristic variants. The goal of the domain conception is to establish a generic architecture for the family of products. There is not consensus on the definition of such architecture. The architecture of a family of products is usually defined as a standard architecture that behaves components, connectors and constraints. For the family of products, the architecture should be as a reference architecture that serves to make each product. Identified variability during the domain analysis must be specified explicitly in the family architecture. The Domain engineering enables to establish the generic architecture defined in the domain conception that will be reused for the construction of each product.

### Feature Types

A feature is a prominent or distinctive user-visible aspect, quality or characteristic of a product or a system [21]. There exists a consensus in the literature about the different types of features. The diverse types of features are:

- *Mandatory features* are present in all products that belong to the product family in the modeled domain.
- *Optional features* may or may not be included in a product. If an optional feature is not part of the product, all sub-features of that feature are also excluded.
- *Alternative features* represent a choice between a set of features from which exactly one has to be chosen. For example the Motor type in figure 2.6 is an alter.

- *Multiple features* capture the possibility to choose multiple features from a set of features, but at least one has to be chosen.

### Constraints/Dependencies between Features

The variability in the family of product is characterized by constraints between the variations points (features). In fact, the resolution of a variation point can influence the resolution of other variation points. The resolution of a variation point can :

- *Requires*. Features can be required by other features - i.e. the existence of the required feature is needed for the former one.
- *Excludes*: Features may exclude each other. This happens when two features can not be selected together, e.g. when the system components that realize these features are incompatible. This is a mutual exclusion.
- *Recommends*: A weak form of the requires relation is a recommendation. The existence of features can be recommended for other features. This can also be seen as the semantics of a default value.
- *Discourages*: Contrary to the latter, features may be discouraged for other features in the system. This is a weak form of the mutual exclusion. Hence, it describes that a feature is not chosen per default.

These constraints are part of the family architecture but very few methods and tools are able to actually define these constraints.

### Product engineering

In the *product engineering* the results of the domain engineering are used for the construction of a specific product, also called derivation. As mentioned above, the results of domain engineering (the models of characteristics, the generic architecture, and the components) contain variability, the derivation of a special product has therefore need of decisions (or choices) associated with these variation points. The product engineering is characterized by how the derivation of the domain is done. A common way to do it is to use a configuration mechanism. Configuration is a well known approach to support the composition of products from several parts. The configuration of technical systems is one of the most successful application areas of knowledge-based systems [16]. The configuration has to be performed in an incremental approach, where each step represents a configuration decision and possibly includes testing, simulating or checking with constraint techniques. However, applying configuration methods to software systems is in an early stage. First attempts are described in [28]. The configuration mechanism can be for example to specify constraints on the features.

### 2.2.5 Summary

The family of products (see section 2.2.1) is a complex concept and we can figure out several fundamental issues in order to manage the product family. We have to define some shared elements within a product family. These elements can be common features from the sales/customer perspective or common product structures from an engineering perspective. There is also a necessity to define basic elements that make a product different from another. From a customer perspective it could be some feature or tools that are selectable. This issue is normally solved in the domain engineering (see section 2.2.4). Concerning the product engineering (see section 2.2.4) the main issue is to define a configuration mechanism that specifies rules for and means of product variation derivation.

## 2.3 Requirements and needs

This section presents both the wishes of the sales department in DNV and the needs to support the product family. These requirements and needs are the result of a collaboration with DNV. Requirements and needs are gathered in a table (see table: 2.1) with a code and their priority. Each requirement has an assigned priority. The different priorities are:

- Low. If a requirement has low priority then the requirement has not to be fulfilled to consider the project as successful for the client.
- Medium. If a requirement has medium priority then the requirement would be fulfilled if only I could. These requirements are considered as good ideas.
- High. If a requirement has high priority then the requirement must to be fulfilled to consider the project as successful.

### 2.3.1 Functional requirements

The functional requirements specify particular behaviors of a system. Functional requirements are:

- The system has to help the sales people to select the offer corresponding to the client needs.
- The system has to create the pricing offers to customers.
- The system has to support the products family engineering. For example we can imagine one application for the domain engineering and one for the product engineering (see section 2.2.4).

### 2.3.2 Non functional requirement

The non functional requirements figured out are as follows:

- Concerning the pricing of an offer one need is to provide a pricing structure that can be trusted and shared.
- The system has to use the Brix technology. Especially the Brix rule engine and the Brix web frame work.
- The system has to integrate the rules described in the pricing scheme.
- The system should be web based in its final version.
- The system should provide the ability to login in its final version.
- The system should provide an interface with the software Salesforce.

### 2.3.3 Needs

Goal of this section is to define high level needs that must be fulfilled in order to be in accord with the settings and background. Technical needs are:

- The future system has to allow business knowledge to be maintained by domain experts without requiring programming skills.
- Several applications may share knowledge about the family of products. The family of products is a model of the domain and it should be possible to use this model to do different tasks.
- The knowledge has to be easy to maintain on a corporate level. A change in the family architecture or in the rules for the pricing of this family must be propagated in all applications using it.
- The system has to support the variability inherent to the family of products concept.
- The system has to support the product engineering helping producing coherent packages.

This list is not exhaustive but it will help to choose among the available technologies usable for the case.

Id	Description	Priority
R01	The system has to help the sales people to select the offer corresponding to the client needs.	High
R02	The system has to integrate the rules described in the pricing scheme	High
R03	The system has to create the pricing offers to customers	High
R04	The system has to use the Brix technology	High
R05	The system has to support the products family engineering.	High
R06	The system has to be role based with one role for a super user and another for a sales person	Medium
R07	The system should be web based in its final version	Medium
R08	The system should provide the ability to login	Medium
R09	The pricing rules should be trusted and shared.	Medium
R10	The system should provide an interface with the software Salesforce	Low
N01	The future system has to allow business knowledge to be maintained by domain experts without requiring programming skills.	High
N02	A change in the family architecture or in the pricing rules must be propagated in all applications using it.	High
N03	The system has to support the variability inherent to the family of products concept.	High
N04	The system has to help to produce coherent packages.	High
N05	Several applications may share knowledge about the family of products.	Low

Table 2.1: Requirements and needs Table

## 2.4 Restrictions

Goal of this section is to define the boundaries of the project. It is important to focus on some part of the problem as the time span for the thesis is quite short (see section A.3). Here is a list of points that specify what is in the scope of the thesis and what is not:

- Variability dimension (see section 2.2.2). Only the space dimension will be considered in the thesis. This is not useful to consider the time dimension because the variation of one product from one version to one other is not important to create pricing offers. They are done one the latest version of each product. It must just affect the integrity of the existing offers.
- Variability level (see section 2.2.3). The variability level has been removed from the scope of the thesis due to the duration of the thesis. Anyway it

should be important to enable everybody to use the system independently of is knowledge.

All the others points of the product family are in the scope of the thesis and will be defined in the next parts of the report.

## **2.5 Summary**

This chapter described settings, requirements and needs that constraint our work. Next step is to figure out some current issues in managing product family in order to investigate which technologies will offer the best coverage of these requirements and needs.





## Chapter 3

---

# State of the art

---

Goal of this chapter is to serve as a basis to find the best available technology to support the case study presented by DNV. That is why this section presents the state of the art in product family and the unsolved problems and the current fields of interest concerning the product family paradigm.

### 3.1 Current technical solution

There are some available products on the market for the management of the family of products. In this section we will present an non exhausting list of the existing products.

#### 3.1.1 BigLeve software Gears

Gears<sup>1</sup> is used to create a software production line capable of producing all of the products in a software product line portfolio. A Gears software production line comprises three key elements:

- *Software Assets* are configurable software artifacts such as source code, requirements, and test cases engineered to be reused across the product line.
- *Product Feature Profiles* model each product in the portfolio in terms of optional and varying feature choices specified for the product line.
- *The Gears Configurator* automatically assembles and configures the software assets, guided by the product feature profiles, to produce the products in the portfolio.

---

<sup>1</sup><http://www.biglever.com/>

### 3.1.2 Pure::Variant

The pure::variants<sup>2</sup> Edition targets at individual developers as well as small to medium sized developer teams. It supports creation, management and evaluation of all necessary models and uses XML based data formats for model storage. Pure::Variant is available as a plug-in for the OpenSource Eclipse<sup>3</sup> Integrated Development Environment (IDE), which is a well known cross-platform development tool. Since pure::variants supports the extensibility of Eclipse, customers can easily perform functional enhancements and appearance changes if required. Pure::Variant process starts with the selection of a compatible set of features. This selection can be done by different type of users. Based on this selection the system helps the user to find a suitable solution. The last part of the process is to create the customized solution.

### 3.1.3 Kumbang

Kumbang is build using a domain ontology (see section 4.2.2) for modeling variability in software product families. Kumbang ontology is based on three layers of abstraction. At the highest level of abstraction is the meta layer that contains the modeling concepts. The next layer is the model layer that contains Kumbang models. The entities that appear in Kumbang models are termed classes and are instances of metaclasses. Finally, the third layer, instance layer, contains the instances of the classes appearing at the model layer [6]. In addition to the ontology Kumbag has a Tool for Configuring product families.

## 3.2 Unsolved problems/Current issues

The state of the art products presented above face some difficulties to handle the *Product family* concept. They face unsolved problems or current issues that will need further work to correct them. Current issues in product family are as follows:

- *The evolution of the domain.* In order to define the family architecture most of the methods specify a set of commonalities (see section 2.2). But architectural evolution can be particularly challenging when it concerns product family architecture. Especially when the core of the family architecture evolves due to a major common requirements or upgrades. In fact the set of commonalities that is assumed to be stable can become unstable over the time. This problem implies a maintenance effort to have the domain model up to date. On good example the example to illustrate the need to handle the evolution of the domain is the example presented in [17]. This

---

<sup>2</sup><http://www.pure-systems.com/>

<sup>3</sup><http://www.eclipse.org/>

example talk about routers in telecommunication networks. These routers are dynamically configured but in order to establish the required connection they need to be stable (i.e. they are enduring in the domain model).

- *Consistency of the data.* The consistency of the data is an issue, in fact it is difficult to check if the data in the product family architecture are consistent or not. This problem is tightly linked with the evolution of the domain.
- *Feature modeling problems.* The Feature modeling is a common way to support family of products nevertheless this approach has numerous problems cited in [10]. Among these problems we find:
  - Insufficient consideration of variability in feature-artefacts.
  - The Reasons of Variability that are capture in the feature get lost. This problem leads to an impossibility to reuse the feature in the same context. The article [10] gives the example of an Asian limousine that has an optional feature A, because the feature is considers as optional in Asia. The design of a new Asian car can consider and reuse the fact that A is not mandatory in Asia. But if the information is lost then it is no more possible to reuse it.
  - A high similarity among the product lines leads to a high redundancy among the feature models.
- *Family of product engineering supported by workflow system [30].* One element that is in the scope of current and future research concerning product family is to model the process to derivate the products family architecture. In fact as we seen in section 2.2.4, product management is used in domain engineering that it is used to construct a specific product. All this process can be supported by a workflow system. This is a simple idea but very few existing product actually are supported by a workflow system.
- *Complex management of the family architecture.* To have a more understandable architecture we can use an abstraction mechanism called filtering and view generation. A difficult problem to solve concerns the management of consistency among all views, especially if generated views must be stored and kept up to date [22]. One other problem with this kind of mechanism is that software engineers often do not know all consequences of the choices they make in the derivation process. As lot of relations between the entities in the family architecture are hidden. In other words the selection of one variation points can have some undesired side effects. This lead to one other issue called Implicit properties.

- *Implicit properties.* A core issue involves the large number of implicit properties (e.g. dependencies) of variation points and variants, properties that are undocumented and either unknown or only known by experts.
- *The testing of the family architecture* has emerged as one field of interest by its own in the last few years and a non negligible part of today's conferences is accorded to the subject.

### 3.3 Summary

This chapter is the second step of our method. It figured out some current issues to support family of products. In the restrictions section we defined as out of boundaries the time dimension of the variability (see section 2.4) so we will not focus on the evolution of the domain problem. Core problem that we assign in this project are the consistency of the data and the issues concerning the features models. Next step in our project is to investigate candidate technologies that fulfilled needs and requirements (see chapter 2) but also enable to solve current issues listed in this chapter.

## Chapter 4

---

# Candidate technologies

---

Goal of this section is to present some candidate technologies that can help to solve some current issues for the product family paradigm (see section 3.2). For each technology we will highlight the benefits to use in the case study provide by DNV. The semantic web technologies and the rules technologies are introduced in order to discover possible advantages to use them in the context of the case study.

### 4.1 Semantic web

The Semantic Web is an extension of the current web in which information is given well-defined meaning, better enabling computers and people to work in cooperation. Today most of the data are generally hidden away in *HTML* files. It could be useful in some contexts, but not in others. The problem with the majority of data on the Web in this form is that it is difficult to use on a large scale, because there is no global system for publishing data in such a way as it can be easily processed by anyone.

#### 4.1.1 Component

The W3C<sup>1</sup> web pages on the Semantic Web include a diagram labeled Architecture. This diagram, sometimes called the Semantic Web layer cake (see figure 4.1). Each layer is seen as building on and requiring the ones below it. The W3C has developed, or is in the process of developing, standards and recommendations for all these layers. Descriptions provided in the following sections are based on the semantic web tutorial by Ivan Herman [19].

---

<sup>1</sup><http://www.w3.org/2001/sw/>

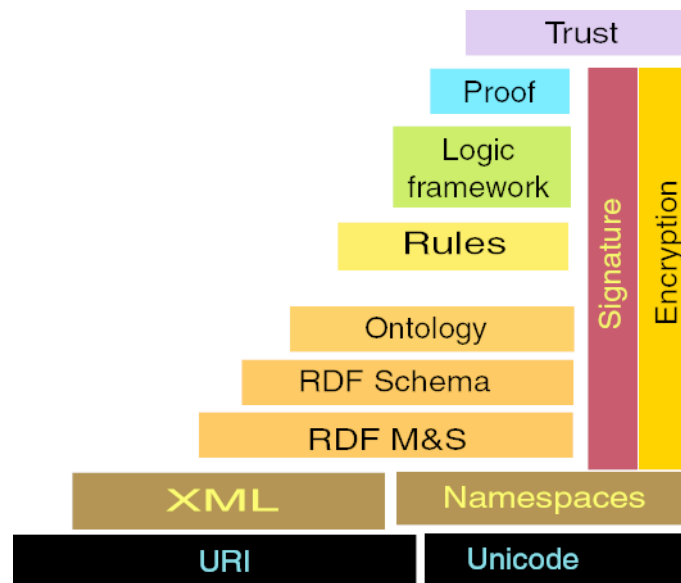


Figure 4.1: Semantic Web layer cake

## XML

*XML* is an evolution of *HTML* which is a language for describing the layout of documents. *XML* describes the structure of the document. *XML* allows authors to create their own markup (e.g. `< AUTHOR >`), which seems to carry some semantics. The problem is that for a computer the user's tag do not carry any semantic. A computer cannot infer, what an author is and how the concept author is related to a concept person for example.

## RDF

The *RDF metadata model* is based upon the idea of making statements about resources in the form of *subject-predicate-object* expressions, called triples in *RDF* terminology. The subject denotes the resource, and the predicate denotes traits or aspects of the resource and expresses a relationship between the subject and the object. All sentences are triples of the form (Property Subject Object) in which the Property is a binary relation, the Subject is a *URI* reference and the Object is either a *URI* reference or a literal<sup>2</sup>. *RDF* includes a resource *Class* and properties *type*, *subclassOf*, etc. *RDF Schema* is a framework that provides a means to specify basic vocabularies for specific *RDF* application languages to use. But *RDF* has some limitations for example in *RDF* there is no cardinality constraints, there is no possibility to specify the disjointness of classes, as well there are no axioms and negations.

<sup>2</sup>Example: (creatorOf <http://www.w3.org/Lassila> Ora Lassila)

## Ontology

Ontologies have moved beyond the domains of library science, philosophy, and knowledge representation. Ontology attracts attentions across many fields in computer science recently. There exists no consensus on definition about ontology. Anyway we can define an ontology as an explicit representation of a conceptualization, the conceptualization includes a set of concepts, their definition and inter-relationships. In many cases, the term ontology is another name denoting the result of familiar activities like conceptual analysis and domain modeling. The roles of ontology vary from knowledge management to semantic interoperability. One important reason for that ontology attracts so much attention recently is the Semantic Web, since ontology is considered the key enabler of Semantic Web. More details about the ontology are given in the section 4.2.

## Logic and Proof

The logic and proof layer is used to establish the consistency and correctness of data sets and to infer conclusions that are not explicitly stated but are required by or consistent with a known set of data. Proofs trace or explain the steps of logical reasoning.

## Trust

The trust layer is a means of providing authentication of identity and evidence of the trustworthiness of data, services, and agents.

### 4.1.2 Motivation to use semantic web technologies

The advantages of the semantic web have been in the focus of the researchers for few years now. They are present in the literature for example the article [34] lists some of these advantages in the context of Internet commerce. The advantages related with the case study are:

- *Reasoning*: Ontologies offer the opportunity to process reasoning, and query on data. This quality can help to manage the product family architecture. In fact the reasoning can be potentially used to support the product engineering (see section 2.2.4) and even help to improve the domain engineering support (i.e reasoning can help to fulfill requirement R05).
- *Sharing*: One of the main goals of the ontologies is to be machine understandable, to be processed by web agents. This particularity offers lots of possibilities to share the family architecture as expressed in the need N05.
- *Knowledge management*: refers to a range of practices used by organizations to identify, create, represent, and distribute knowledge for reuse,

awareness, and learning across the organizations. Knowledge management is characterized by key concepts such as: Tacit versus explicit knowledge [24], Knowledge capture stages, Ad hoc knowledge access. Knowledge management programs may lead to greater innovation, better customer experiences, consistency in good practices and knowledge access across a global organization, as well as many other benefits, and knowledge management programs may be driven with these goals in mind. The ontology can bring lot of benefits for the management for the product families. In fact an ontology can be used to enable the evolution of the domain, and a good management of the domain complexity.

- *Vocabulary flexibility and standardization*: Theoretically, ontology techniques allow users to flexibly choose the words they like. Since users are diverse and it is hard to require them to fully know the standards, this advantage could be interesting. In practice, [13] demonstrated in a financial group how to create a central vocabulary within an ontological context, to standardize the concepts, and to improve the communications between different departments. Flexibility and standardization seem conflicting. In fact they reflect different developmental stages of semantic web. In the initial stage, vocabulary standardization could be prioritized, whereas with the emerging and maturity of ontology mapping or manipulation tools the advantage of vocabulary flexibility will show up.

## 4.2 Ontology in more details

Goal of this section is to present the elements of an ontology, a classification of the different types of ontology, and the existing ontology languages.

### 4.2.1 Elements of an ontology

An ontology typically is built using four constructs [19, 5]:

- *Classes*.
- *Attributes*.
- *Relations*.
- *Instances*.

#### Classes

Classes or Concepts are abstract groups, sets, or collections of objects. They may contain individuals, other classes, or a combination of both. The classes of an ontology may be extensional or intensional. A class is extensional if and only if it



is characterized solely by its membership. More precisely, a class  $C$  is extensional if and only if for any class  $C'$ , if  $C'$  has exactly the same members as  $C$ , then  $C$  and  $C'$  are identical. If a class does not satisfy this condition, then it is intensional. While extensional classes are more well-behaved and well-understood mathematically, as well as less problematic philosophically, they do not permit the fine grained distinctions that ontologies often need to make. For example, an ontology may want to distinguish between the class of all creatures with a kidney and the class of all creatures with a heart, even if these classes happen to have exactly the same members. Some others particularity exist for example the partition.

### Attribute

Objects in the ontology can be described by assigning attributes to them. Each attribute has at least a name and a value, and is used to store information that is specific to the object it is attached to. If an ontology do not define attributes for the concepts it will be a taxonomy or a Controlled Vocabulary . These are useful, but are not considered true ontologies.

### Relations

An important use of attributes is to describe the relation between objects in the ontology. Typically a relation is an attribute whose value is another object in the ontology. The most important type of relation is the relation *is-superclass-of*, the converse of *is-a*, *is-subtype-of* or *is-subclass-of*. This defines which objects are members of classes of objects. For example an elephant is a mammal which implies that it is an animal.

### Instances

Instances or Individuals are the basic, "ground level" components of an ontology. The individuals in an ontology may include concrete objects such as people, animals, tables, automobiles, molecules, and planets, as well as abstract individuals such as numbers and words. Strictly speaking, an ontology need not include any individuals, but one of the general purposes of an ontology is to provide a means of classifying individuals, even if those individuals are not explicitly part of the ontology.

## 4.2.2 Ontology types

Different types of ontologies exist depending on level of generality [15].

- *Top-level ontologies or Upper ontologies* are models of the common objects that are generally applicable across a wide range of domain ontologies. It contains a core glossary in whose terms objects in a set of domains can

be described. There are several standardized upper ontologies available for use, including Dublin Core<sup>3</sup>, GFO<sup>4</sup>, OpenCyc ResearchCyc<sup>5</sup>, SUMO<sup>6</sup>, and DOLCE<sup>7</sup>.

- *Domain ontologies.* A domain ontology (or domain-specific ontology) models a specific domain, or part of the world. It represents the particular meanings of terms as they apply to that domain.
- *Task ontologies.* A task ontology models a specific task. An example of task ontology is provided in [23].
- *Application ontologies.* These ontologies are not reusable as they are specific to one application.

### 4.2.3 Web ontology languages

#### DAML

The firsts ontology languages were *DAML* and *OIL* [18, 7, 5]. *DAML* is a schema language that can be used to constrain and describe data following the *RDF data model*. To put it another way: *DAML* is an *RDF schema* language. *RDF* already has a schema language, called *RDF Schema*, and *DAML* is an extension of this language. The value of *DAML* is thus that it allows one to describe *RDF* data, and so makes it possible to add more semantics to the data. What *DAML* adds to *RDF Schema* is additional ways to constrain the allowed values of properties, and what properties a class may have. In addition, it provides some properties that can be truly useful to generic software, which are:

- *daml:samePropertyAs*, which makes it possible to say that two *RDF* properties from different schemas are in fact the same property.
- *daml:inverseOf*, which can be used to say that one *RDF* property is the inverse property of another. This means that together the two properties describe a relationship both ways, and so provides some of the two-way semantics of topic map associations. This mechanism does not work very well for relationships that are not binary, however.
- *daml:TransitiveProperty*, which is a property type which other property types can subclass to make it clear that they are transitive.

*DAML* strengthens the *RDF* schema language, and adds a little bit of semantics on top.

---

<sup>3</sup><http://dublincore.org/>

<sup>4</sup><http://www.onto-med.de/en/theories/gfo/index.html>

<sup>5</sup><http://www.cyc.com/>

<sup>6</sup><http://www.ontologyportal.org/>

<sup>7</sup><http://www.loa-cnr.it/DOLCE.html>

## OIL

*OIL* is very similar to *DAML* as it is also an extension of *RDF Schema*, and the capabilities of *OIL* and *DAML* are very similar. *OIL* is a proposal for a web-based representation and inference layer for ontologies, which combines the frame-based languages with the formal semantics and reasoning services provided by description logics. The two main precursors of *OIL* are:

- *DL* describes knowledge in terms of concepts and role restrictions that are used to automatically derive classification taxonomies. The main effort of the research in knowledge representation is in providing theories and systems for expressing structured knowledge and for accessing and reasoning with it in a principled way. Frame language is a metalanguage. It applies the frame concept to the structuring of language properties. *Frame languages* are usually software languages.
- *Frame languages* are rather focused on the recognition and description of objects and classes, and relations and interactions are considered as "secondary". In general, *frame* in this context means something that can be or has to be fulfilled. In such sense, for example: Object-oriented programming languages are frame languages, but also every grammar is a *frame language*. *Frames* are Roughly similar to the object-oriented paradigm, they represent classes with certain properties called attributes or slots whereas they do not have methods. Frames are thus a machine-usable formalization of concepts or schemata.

The two languages have been gathered to form the *DAML+OIL* language. *DAML+OIL* was developed by a group which was jointly funded by the US Defense Advanced Research Projects Agency (*DARPA*) under the *DAML* program and the European Union's IST funding project. As its predecessors *DAML+OIL* is a semantic markup language for Web resources. It is build on earlier W3C standards such as *RDF* and *RDF Schema*, and combines the strengths of each of *DAML* and *OIL*.

## OWL

The *OWL* Language is a research-based revision of the *DAML+OIL* web ontology language [5]. *OWL* is intended to be used when the information contained in documents needs to be processed by applications, as opposed to situations where the content only needs to be presented to humans. *OWL* can be used to explicitly represent the meaning of terms in vocabularies and the relationships between those terms.

### OWL sub-Languages

*OWL* currently has three sub-languages: *OWL Lite*, *OWL DL*, and *OWL Full*.

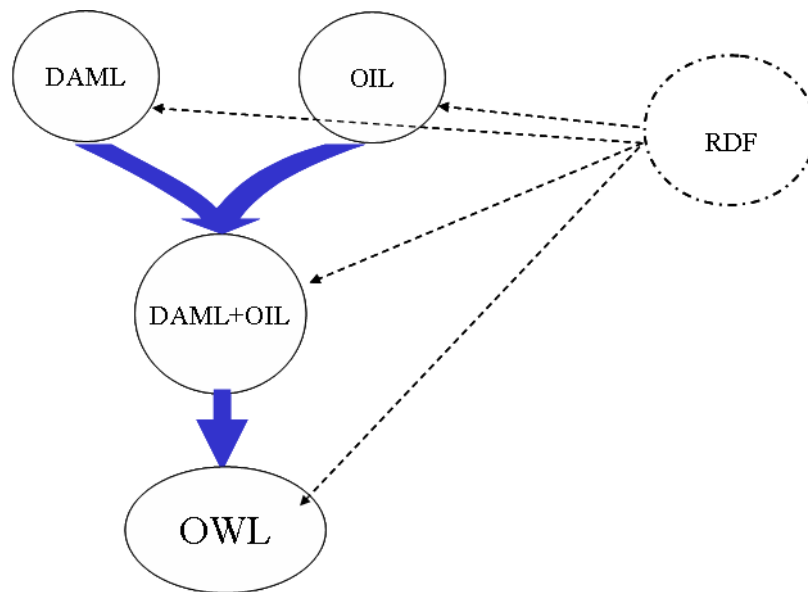


Figure 4.2: Ontology languages evolution.

These three increasingly expressive sub-languages are designed for use by specific communities of knowledge engineers and users. *OWL Lite* supports those users primarily needing a classification hierarchy and simple constraints. For example, while it supports cardinality constraints, it only permits cardinality values of 0 or 1. It should be simpler to provide tool support for *OWL Lite* than its more expressive relatives, and *OWL Lite* provides a quick migration path for thesaurus and other taxonomies. *OWL DL* supports those users who want the maximum expressiveness while retaining computational completeness (all conclusions are guaranteed to be computed) and decidability (all computations will finish in finite time). *OWL DL* includes all *OWL* language constructs, but they can be used only under certain restrictions (for example, while a class may be a subclass of many classes, a class cannot be an instance of another class). *OWL DL* is so named due to its correspondence with description logic, a field of research that has studied the logics that form the formal foundation of *OWL*. *OWL Full* is meant for users who want maximum expressiveness and the syntactic freedom of *RDF* with no computational guarantees. For example, in *OWL Full* a class can be treated simultaneously as a collection of individuals and as an individual in its own right. *OWL Full* allows an ontology to augment the meaning of the pre-defined (*RDF* or *OWL*) vocabulary. It is unlikely that any reasoning software will be able to support complete reasoning for every feature of *OWL Full*. Each of these sublanguages is an extension of its simpler predecessor, both in what can be legally expressed and in what can be validly concluded. The following set of relations holds. Their inverses do not.

- Every legal *OWL Lite* ontology is a legal *OWL DL* ontology.
- Every legal *OWL DL* ontology is a legal *OWL Full* ontology.
- Every valid *OWL Lite* conclusion is a valid *OWL DL* conclusion.
- Every valid *OWL DL* conclusion is a valid *OWL Full* conclusion.

## 4.3 Rule based system

Using a set of assertions, and a set of rules that specify how to act on the assertion set, a rule-based system can be created. Rule-based systems are fairly simplistic, consisting of little more than a set of *if-then* statements, but provide the basis for so-called expert systems which are widely used in many fields. The concept of an expert system is this: the knowledge of an expert is encoded into the rule set.

### 4.3.1 Theory

The rule-based systems use a simple technique. The system examines all the rule conditions (*IF*) and determines a subset, *the conflict set*, of the rules whose conditions are satisfied. One of the rules within the conflict set is triggered (fired). Which one is chosen is based on a *conflict resolution strategy*. When the rule is fired, any actions specified in its *THEN* clause are carried out. This loop of firing rules and performing actions continues until one of two conditions are met: there are no more rules whose conditions are satisfied or a rule is fired whose action specifies the program should terminate [14].

#### Conflict resolution

Which rule is chosen to fire is a function of the conflict resolution strategy. Which strategy is chosen can be determined by the problem or it may be a matter of preference. In any case, it is vital as it controls which of the applicable rules are fired and thus how the entire system behaves. There are several different strategies, but here are a few of the most common:

- *First Applicable*: If the rules are in a specified order, firing the first applicable one allows control over the order in which rules fire. This is the simplest strategy. But there is a risk that of an infinite loop on the same rule happens. In fact if the working memory remains unchanged after the rule has been fired then the conditions of the first rule have not changed and it will fire again and again. To solve this, it is a common practice to suspend a fired rule and prevent it from re-firing.

- *Random*: Though this strategy do not provide the predictability or control of the first-applicable strategy, but it does have its advantages. Its unpredictability is an advantage in some circumstances (such as games for example). A random strategy simply chooses a single random rule to fire from the conflict set. Another possibility for a random strategy is a fuzzy rule-based system in which each of the rules has a probability such that some rules are more likely to fire than others.
- *Most Specific*: This strategy is based on the number of conditions of the rules. From the conflict set, the rule with the most conditions is chosen. This is based on the assumption that if it has the most conditions then it has the most relevance to the existing data.
- *Least Recently Used*: In this strategy each of the rules is accompanied by a time or step stamp, which marks the last time it was used.
- *"Best" rule*: For this strategy to work, each rule is given a weight, which specifies how much it should be considered over the alternatives. The rule with the most preferable outcomes is chosen based on this weight.

## Chaining

This section explains the two different chaining methods applicable to chain rules.

### Forward-Chaining

Rule-based systems, are adaptable to a variety of problems. In some problems, information is provided with the rules and the Artificial intelligence (AI) follows them to see where they lead. An example of this is a medical diagnosis in which the problem is to diagnose the underlying disease based on a set of symptoms (the working memory). A problem of this nature is solved using a forward-chaining, data-driven, system that compares data in the working memory against the conditions (*IF* parts) of the rules and determines which rules to fire (see figure 4.3).

### Backward-Chaining

In other problems, a goal is specified and the AI must find a way to achieve that specified goal. For example, if there is an epidemic of a certain disease, this AI could presume a given individual had the disease and attempt to determine if its diagnosis is correct based on available information. A backward-chaining, goal-driven, system accomplishes this. To do this, the system looks for the action in the THEN clause of the rules that matches the specified goal. In other words, it looks for the rules that can produce this goal. If a rule is found and fired, it takes each of that rules conditions as goals and continues until either the available data

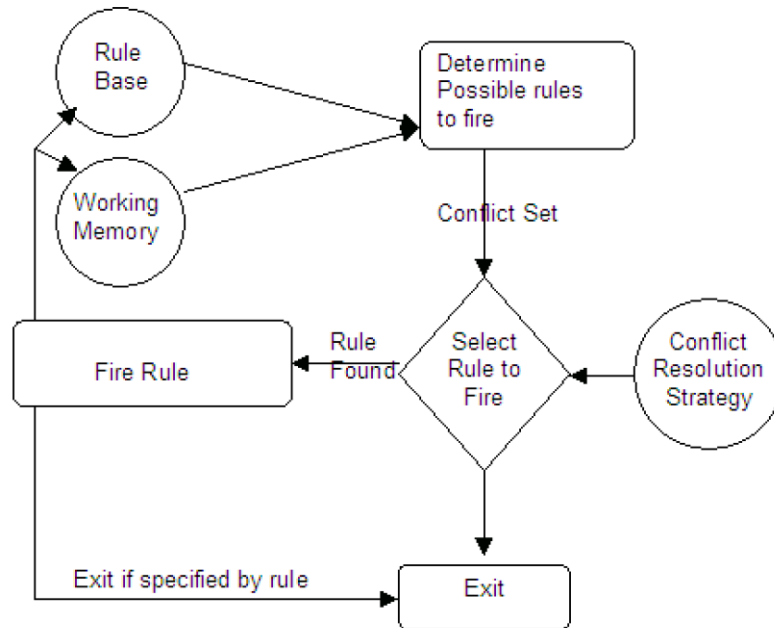


Figure 4.3: Forward chaining process (source [14])

satisfies all of the goals or there are no more rules that match (see figure 4.4).

#### Which method to use?

Of the two methods available, forward or backward chaining, the one to use is determined by the problem itself. A comparison of conditions to actions in the rule base can help determine which chaining method is preferred. If the average rule has more conditions than conclusions, that is the typical hypothesis or goal (the conclusions) can lead to many more questions (the conditions), forward-chaining is favored. If the opposite holds true and the average rule has more conclusions than conditions such that each fact may fan out into a large number of new facts or actions, backward-chaining is ideal.

If neither is dominant, the number of facts in the working memory may help the decision. If all (relevant) facts are already known, and the purpose of the system is to find where that information leads, forward-chaining should be selected. If, on the other hand, few or no facts are known and the goal is to find if one of many possible conclusions is true, use backward-chaining.

#### 4.3.2 Motivation to use rule-based system

The main reasons to use rules in the case study are:

- *Declarative Programming*: The rule based system can be used to easily

express solutions to hard problems, and consequently have those solutions verified (rules are much easier to read than code). Rule systems are capable of solving hard problems, yet providing a solution that is able to explain why a "decision" was made. This is not so easy with other types of AI systems. Regarding to the needs (see section 2.3.3) and requirements (see section 2.3) this particularity can allow sale person to trust the system. Moreover the quality and trustability of the system is improved using rule-based system because the usage of rules create a repository of knowlegde. This means its a single point of truth, for business policy (for instance).

- *Logic and Data Separation:* Using a rule-based system implies that the data are in the domain objects, and the logic is in the rules. This is fundamentally breaking the object oriented coupling of data and logic (this can be an advantage as well as a disadvantage depending on your point of view). The upshot is that the logic can be much easier to maintain as there are changes in the future, as the logic is all layed out in rules. In the case study regarding to the needs N01 and N02 (see section 2.3.3), it seams that it is an advantage.
- *Tool Integration:* There exists numerous tools on the market that provide ways to edit and manage rules and get immediate feedback, validation and content assistance. Auditing and debugging tools are also available. DNV's Brix Rule engine and rule editor (see figure 2.4) allow all these operations. The integration with these tools should answer to the requirement R04.
- *Understandable rules (readable by domain experts):* By creating object models that model a problem domain, rules can look very close to natural language. They lend themselves to logic that is understandable to domain experts who may be non technical (as all the program plumbing is in the usual code, hidden away). This is clearly an advantage because this particularity permits to fulfill the need N01 (see section 2.3.3).

## 4.4 Software development using semantic technologies

Goal of this section is to show that are attempts of employing semantics web technologies in closely related area to product family. This section will present recommendations from the W3C and workshops which interest is the usage the semantic web to support software development.

### 4.4.1 Semantic web in systems and software engineering

The W3C gives some ideas to use the semantic web for software development among them we find this idea: **Verifying Feature Models using OWL**. The



idea is based on the publications [32, 31]. Features are prominent and distinctive user visible characteristic of a system. Systems in a domain shared common features and also differ in certain features. A feature model consists of a feature diagram and other associated information (such as rationale, constraints and dependency rules). However, the lack of a formal semantics and reasoning support of feature models has hindered the development of this area. Industrial experiences show that methods and tools that can support feature model analysis are badly appreciated.

A feature diagram (see figure: 2.6) provides a graphical tree-like notation that shows the hierarchical organization of features. We can use OWL to represent all of them (see the OWL codes below). For example, we can represent "F is a mandatory sub-feature of C" with the following OWL DL axiom :

```
<owl:Class rdf:about="C"/>
  <owl:Restriction>
    <owl:onProperty rdf:resource="#hasF">
  <owl:someValuesFrom rdf:resource="#F">
  </owl:Restriction>
</owl:Class>
```

and represent "F is an optional sub-feature of C" with the following OWL DL axiom :

```
<owl:Class rdf:about="F"/>
  <owl:Restriction>
    <owl:onProperty rdf:resource="#hasC">
  <owl:someValuesFrom rdf:resource="#C">
  </owl:Restriction>
</owl:Class>
```

By representing feature models as OWL axioms, the consistency of feature configurations can be automatically checked by Description Logic-based Semantic Web reasoning engines [31].

#### 4.4.2 Semantic web enabled software engineering

The advantages of semantic technologies in software engineering include reusability and extensibility of data models, improvements in data quality, and discovery and automated execution of workflows. The Semantic Web can serve as a platform on which domain models can be created, shared and reused. This subject is approached in the International Workshop on Semantic Web Enabled Software Engineering (SWESE<sup>8</sup>). This workshops tries to find answers to questions such

---

<sup>8</sup><http://www.mel.nist.gov/msid/conferences/SWESE/>

as: Could the Web-based, semantically rich formality of OWL be combined with emerging model driven development tools to provide improvements in both the process and product of software development activities?, How to integrate OWL, UML and the Model Driven Architecture (MDA)? MDA and Product family are very close concepts. The popularity and power of the MDA approach has been used in many software developments in particular in product family. In parallel, Semantic Web language standards have arrived with substantial tool support that also provide a means of describing models, but providing different capabilities than the models typical of MDA tools. The members of these workshops try to investigate the advantages of bridging these approaches. The main topics of interest of the SWESE's members are:

- Visions for Semantic Web driven software engineering;
- Tools developed or being developed for software engineering using SW languages;
- Integration or application development projects combining Software Engineering techniques and Semantic Web tools or languages;
- Integration of UML, object oriented programming languages and Semantic Web languages Ontologies for software engineering;
- Feature modelling and ontologies;
- Ontology reasoning for software engineering;
- Ontology-Driven Architecture: How to introduce Semantic Web technology into mainstream development processes.

Regarding to these topics of interests it is obvious that there are efforts to use the semantic web in tightly related concept to the product family paradigm.

## 4.5 Summary

This section highlights some potential pros to use semantic technologies and rule base system in the case study. First the rules can help to fulfill the requirements R02, R03, R09 and the need N01 by separating logic and data and by providing understandable knowledge. The semantic technologies and especially the ontology web language may help to support the family of products. In fact the semantic technologies are useful to manage knowledge and they ability to process reasoning, and query on data can improve domain and product engineering (see section 2.2.4). That are the motivations to use these technologies in our project.

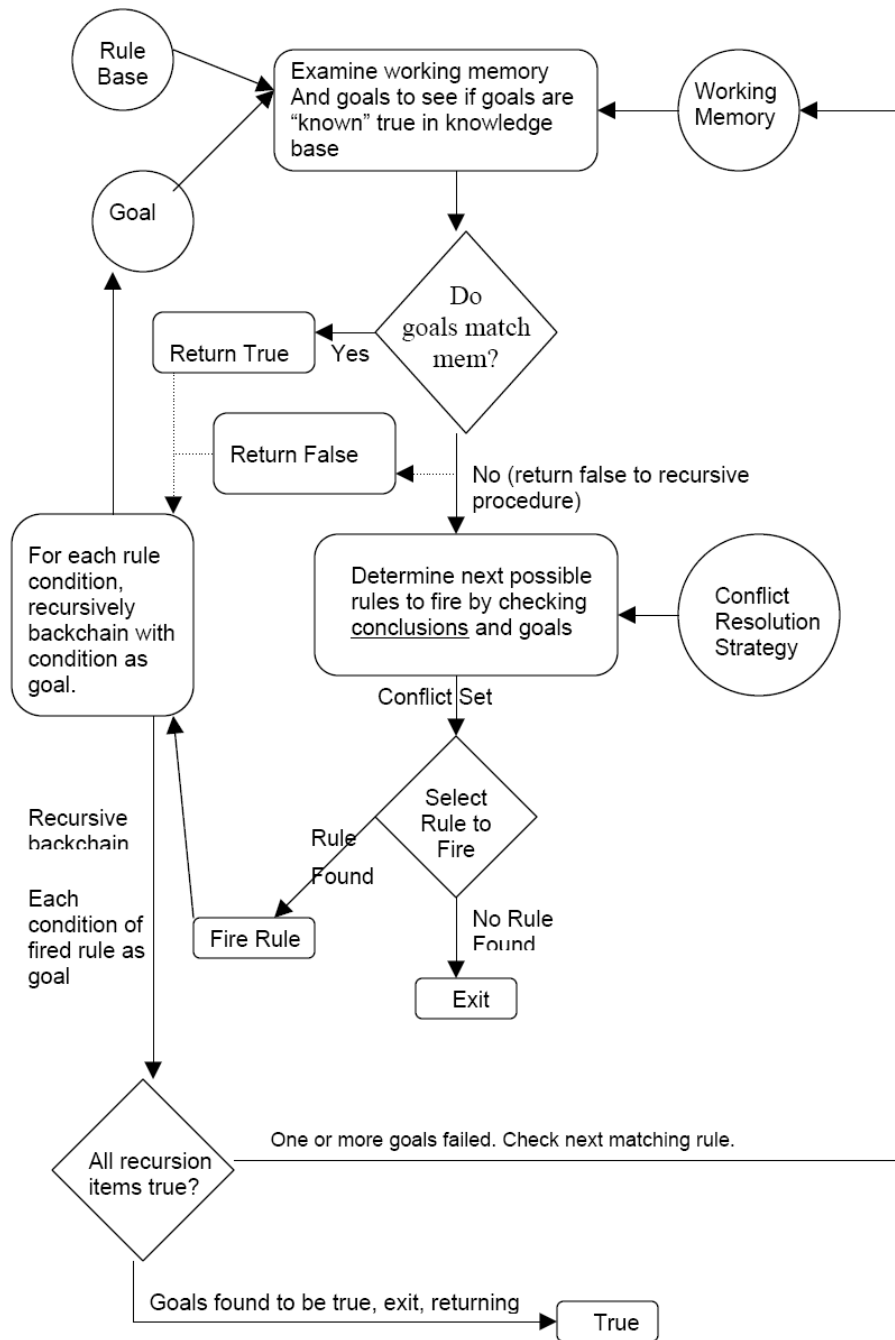


Figure 4.4: Backward chaining process (source [14])



## Chapter 5

---

# Product family pricing calculator

---

The previous chapters gave a solid background on product family and on the problems related to the master thesis subject. In the past chapters we analyzed the best available technologies to support the creation of a centralized system to create offers according to the product family. Next step is to define how the system should interact with the users to achieve business requirements and needs (see section 2.3 and section 2.3.3). Goal of this chapter is to reflect the decisions taken during the design and sketch out the implementation phase carried out.

### 5.1 System definition

The goal of this section is to define the system to create in a more structured way. This section answers to important questions like: who will use the system? or how will the system used?

#### 5.1.1 Actors

Based on the client's requirements and needs we can define two types of users:

- *Sales people.* The sales people use the system to build a pricing offer. This means that they must be able to create, save and modify an offer. The system must allow them to find the package corresponding to the client demand, and to price this package.
- *Administrator.* An administrator manages the system's user accounts, manages the DVN's branding structure, and he is able to manage the way to compute the price of each package.

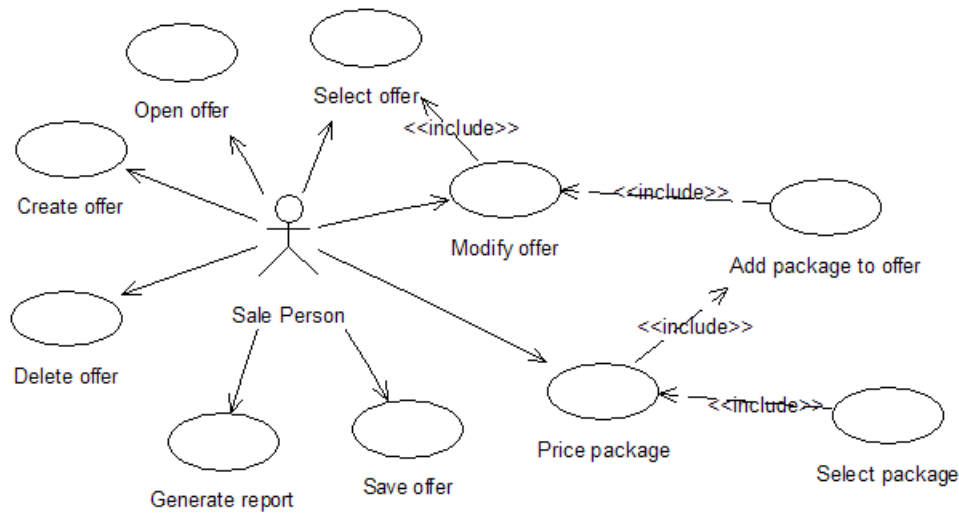


Figure 5.1: Use cases for a sales person

### 5.1.2 Use cases

Use case (UC ) diagrams are a way to structure the client requirements. They can be used to show the interaction between the system and actors. A use case should include all the alternative outcome of a system interaction. A use case is a first step to clarify the system interactions and it defines good basis to start the system architecture. The core of the report contains only top level use case diagram in order to make the system easy to understand. The detailed use cases are attached in appendix B.

#### Sale people use cases

The sales people are application users i.e. they just use the system to find packages, compute price or manage their offers. The diagram present in figure 5.1 shows the interactions between the sales people and the system. The use cases defined in the figure 5.1 relate the requirements R01, R02, R03, R04.

#### Administrator use cases

An administrator is a domain expert or knowledge worker that maintain the pricing rules and the families architectures. The diagram present in figure 5.2 shows how the administrator interact with the system. The table 5.2 relate the use cases with the requirements.

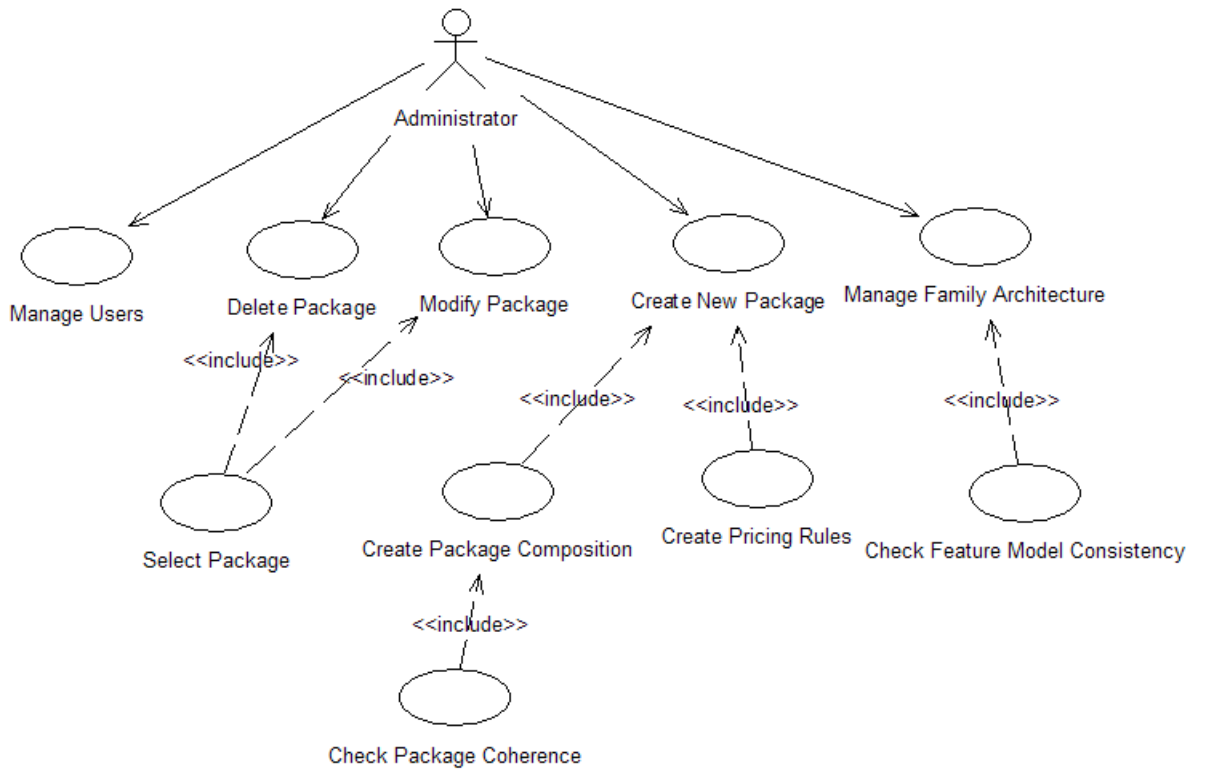


Figure 5.2: Use cases for an administrator

Id	Name	Related requirements	Related needs
UC01	manage users	R08	
UC02	delete package	R05	N01, N02, N03
UC03	modify package	R05	N01, N02, N03
UC04	select package	R01	
UC05	create new package	R02, R05	N01, N02, N03, N04
UC06	create pricing rules	R02	N01, N02
UC07	create package composition	R05	N01, N02, N03, N04
UC08	check consistency		N04
UC09	create new family	R05	N01, N02, N03, N04

Table 5.1: Administrator use cases related to requirements and needs

## 5.2 System overview

The goal of this section is to give an overall presentation of the system; to explain the main ideas that are used in our solution to achieve the goals of the thesis. A constant effort has been done to enable to develop a system to manage and price the family of products that does enable a high variability. More details and justifications are presented in the next sub sections.

### 5.2.1 Components

The system has two users (see section 5.1.1). These users interact with the system using different components. Goal of this section is to present the different components and how they fit all together. The system has four components:

- *Web site.* sales people interacts through web site. It allows creating offers and generating a paper version of an offer. A offer contains a set of packages that have been priced.
- *Ontology.* The role of the ontology is to be the operational data for the web site. The web site uses the ontology as a database component. The ontology encodes all constraints of the system. The ontology is also used by the others components.
- *Rules editor.* The rules editor is a DNV's software (see section 2.1.2). This component is used to create the pricing rules for the packages.
- *Packages management tools.* This component aims to support the family of products engineering (see section 2.2.4). The domain engineering is supported by an ontology editor (see section 5.3.1) whereas the product engineering is supported by a configurator tool. The ontology editor is used



to manage the family architecture (i.e. manage the ontology). The configurator enables to create, delete and edit packages. It uses the ontology as basis for all the operations it makes, but it also edits the ontology creating instances of class.

The system is built around an ontology. In our system a data author (i.e. an administrator) creates operational data based on a pre-existing ontology. The operational data are stored in the ontology itself. Application users then interact with the web site component to perform analysis or query on the operational data in order to find a package and calculate its price (see appendix C). The links between the components and the interactions the users can have with them are simplified in the figure 5.3. The architecture is inspired by the Neutral authoring scenario presented in the paper [29]. In the figure 5.3 the package management tool uses the ontology, and also changes it. In the package management tools the steps to be applied on the ontology to derivate a family architecture are encoded. The tool has no constraints defined; all the constraints are encoded in the ontology.

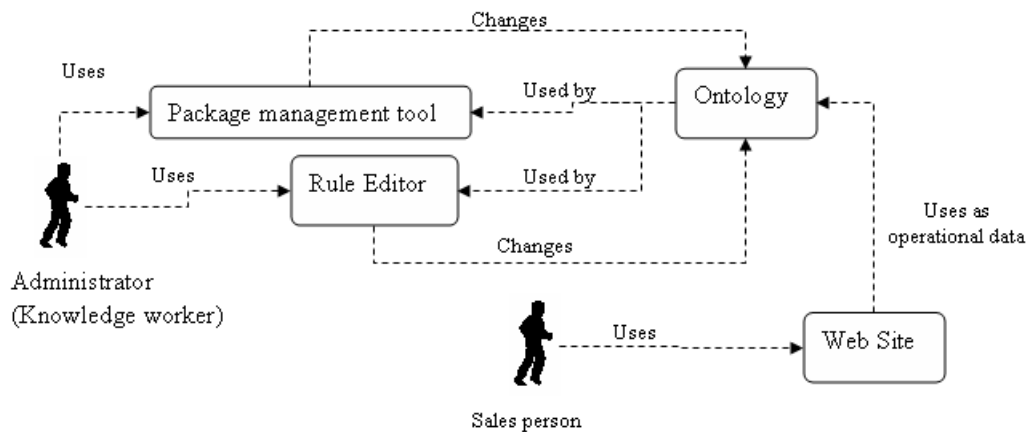


Figure 5.3: Application's components use

### 5.2.2 Product selection

In order to help the sales persons to choose the right package according to a specified branding structure we came up with the idea of a tool using feature model. One benefit of this approach is that it computes the possible packages based on the decisions made by the user. That really helps the sales person to choose the right package by decreasing the number of available packages after each feature selection. Typically, the user starts by selecting features that represent the functionalities of the desired package. As soon as the user has made the

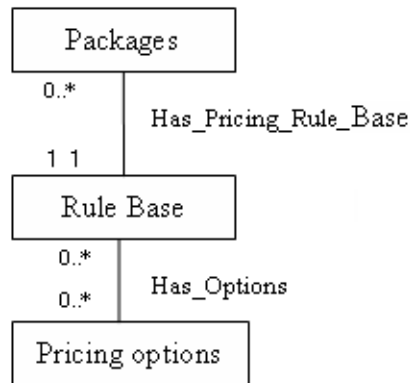


Figure 5.4: Pricing model

first decision, the system starts to compute the impacts of these user decisions to reduce the number selectable packages. Decisions can be made on an abstract level using customer understandable terms or on "Technical" level using expert understandable terms.

### 5.2.3 Pricing

The pricing ability is one of the core reasons that bring DNV Software to provide the project. There is a need to have a flexible structure, which follows some pricing rules for the pricing of the packages. Flexibility and rules seem difficult to integrate. It is not the case because the system allows having different rules for each package. These rules are shared between all the sales persons. The challenge was to find an architecture that allows defining different pricing rules and different options for these rules at the package level. Facing this problem we came up with a simple but very adaptive model (see figure 5.4).

A rule base is a set of rules. These rules follow the syntax of the DNV rule engine (see section 5.3.2). The pricing options are just a set of options that the users have to give to the rules in order to provide a result. Each package can have a different set of pricing options. An example of pricing option is a percentage of discount according to the client. The strengths of this model is that event the way to display the pricing options and the type (integer, string, positive integer...) of attended entry is stored in persistent data (Pricing Options) with a low complexity level that permits to everyone to change the pricing options. We can define in the pricing options if the percentage of discount must be selected among predefined values from a DropDownList or just enter in a TextBox.

### 5.2.4 Package management

The package management is a core point of the project. In fact it has been specified in the requirement that the system must help to manage the package composition and architecture. The package management is supported by the package management tools.

#### Packages composition management

The management of the packages composition includes:

- Create a new package. This is provided a piece of software that we developed. This piece of software is presented in the next sub-section.
- Delete a package.
- Change a package.

#### Create a new package

To create a package we came up with the idea of a configurator using feature model. Configuration activities consist of making decisions about the desired product based on the products features. The configuration tool is used to ensure a consistent, complete and correct solution. In our approach the features, packages, and program to sell (that will be called "Tool" in the rest of the document) are represented in a model (see figure: 5.5).

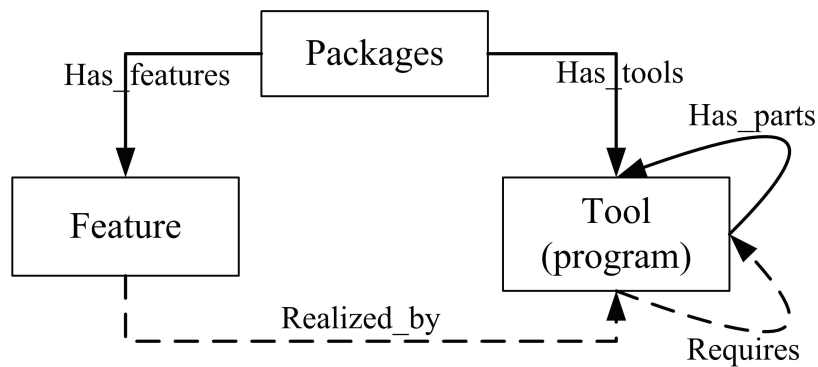


Figure 5.5: Configurator model

The *realized* relation is a further bidirectional dependency that expresses that features are supported by *Tools* in an n-to m mapping: one *feature* can be supported by one or more artifacts and the other way round one or more features can be realized by one artifact. This relation is useful to check the model consistency. In fact the *Tools* that are in the composition of a package realize a set of features. This set has to be the same as the set of feature defined by the relations *Has Features*. In other words a package have to be described by features that are

supported by the tools that compose the packages. The overall functioning of a package creation is simple (see figure 5.6). We were not able to find a domain expert capable to define exclusions between tools. So the constraints of exclusion between tools are not yet supported by the system and therefore, they are not represented in the figure 5.5.

### **Delete or change an existing package**

When deleting an existing package some offers that include the deleted package can become inconsistent with the data. It is the same problem when changing an existing package. Consequently, there is a need to handle versions of the ontology. At the development stage the ontology versioning is not supported. Further work is required in order to investigate and integrate a tool for robust versioning of an ontology, a candidate here is the PROMT tool [25].

### **Family architecture management**

Each instance of a package is based on a family architecture. It is a difficult process to define the architecture (see section 2.2.4). In our solution the ability to manage the families of products structure is provided using an ontology editor (see section 5.3.1).

## **5.3 Technical overview**

The goal of this sub section is to reflect some important technical aspects of the prototypes that has been developed.

### **5.3.1 System design**

This section presents important decisions that we made during the design phase of the project. In the design phase we established the architecture (see section 5.2.1). During this phase we expanded upon the information established in the requirements and in the use cases.

### **Ontology Language**

In this section we will study which ontology language is the most suitable to implement the family of products ontology. *OWL* has three sub languages (see section 4.2.3). As each sub-language is an extension of its simpler predecessor we can stop to look at the superior languages as soon as we find a suitable *OWL* sub-language to represent the ontology. We know that a package can have numerous features that describe it. *OWL Lite* supports cardinality constraints, but it only permits cardinality values of 0 or 1. If we admit that *OWL Lite* is suitable to represent the packages then it implies that *OWL Lite* being able to support

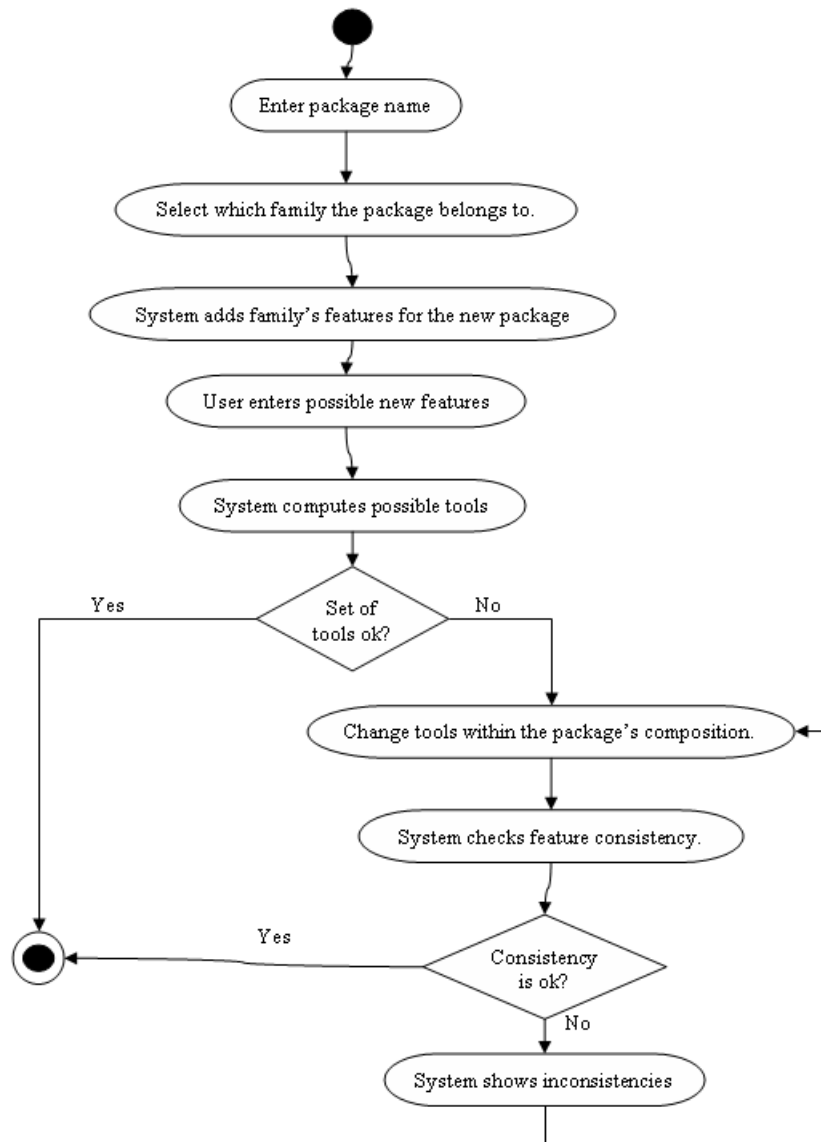


Figure 5.6: Package creation process

the set of features for each package. It is equivalent to the assertion *OWL Lite* permits cardinality value \*; that is absurd. *OWL Lite* is not suitable to represent the packages.

In order to evaluate the *OWL DL* we took the sub part of definition of the Description Logic expressiveness it is based on and we compare with the sub part of the Description Logic expressiveness needed to represent the product family. *OWL DL* provides the expressiveness of *SHOIN<sup>D</sup>* plus functional properties. *SHOIN<sup>D</sup>* means:

- $\mathcal{S}$ . An abbreviation for  $\mathcal{AL}$  and  $\mathcal{C}$  with transitive properties.  $\mathcal{C}$  means Complex concept negation.  $\mathcal{AL}$  means Attributive language. This is the base language which allows:
  - Atomic negation (negation of concepts that do not appear on the left hand side of axioms).
  - Concept intersection.
  - Universal restrictions.
  - Limited existential quantification.
- $\mathcal{H}$  Role hierarchy (subproperties - `rdfs:subPropertyOf`).
- $\mathcal{O}$  Nominals. (Enumerated classes of object value restrictions - `owl:oneOf`, `owl:hasValue`).
- $\mathcal{I}$  Inverse properties.
- $\mathcal{N}$  Cardinality restrictions (`owl:Cardinality`, `owl:MaxCardinality`).
- $\mathcal{D}$  Use of datatype properties, data values or data types.

The ontology language to pick has to support at least :

- $\mathcal{H}$  to create the hierarchy that characterize the DNV's branding structure.
- $\mathcal{S}$  and  $\mathcal{O}$  in order to enable the package management tool to infer the possible tools for a set of features and vise versa.
- $\mathcal{D}$  and functional properties in order to be able to use the ontology as operational data.

According to this enumeration OWL DL is suitable to support the ontology development. Moreover OWL FULL is not well supported by reasoning, (i.e. reasoning is indecisive).

### Guide to choose a Semantic Web Toolkit

Traditional programming languages like CSHARP and knowledge engineering ones (based on LISP, PROLOG, etc.) have more and more difficulties to interact. Traditional programmers and knowledge engineers often don't understand each other. Their tools poorly understand each other too. Knowledge engineers are experts in formal logic and other domains of mathematical theory and work in terms like *concept*, *fact*, *hypothesis*, *goal variable*, *frame*, *fuzzy logic*, *certainty factor*, etc. On the contrary, software engineers work in common terms like *program*, *declaration*, *statement*, *memory*, *procedure*, *stack*, *assignment*, etc. That is why it has been a real challenge to find the good semantic web toolkit to use in the web site code. After some evaluations that are presented in the appendix D we choose the SemWeb toolkit. This choice has been motivated by the fact that SemWeb supports the SPARQL<sup>1</sup> query language and also because it supports numerous databases and serialization formats.

### Ontology editor

Ontology editors are applications designed to assist in the creation or manipulation of ontologies. They often express ontologies in one of many ontology languages. Some provide export to other ontology languages. The choice of the ontology editor was important because this editor will be used to construct the ontology that will be utilized by the web site component and by the package management tools. Among the most relevant criteria for choosing an ontology editor are the degree to which the editor abstracts from the actual ontology representation language used for persistence and the visual navigation possibilities within the knowledge model. Next come built-in inference engines and information extraction facilities, and the support of upper ontologies such as OWL-S, Dublin Core, etc. Another important feature is the ability to import and export foreign knowledge representation languages for ontology matching. Among the most famous ontology editor we find:

- KAON<sup>2</sup> (single user and server based solutions possible, open source, from IPE Karlsruhe),
- Ontolingua<sup>3</sup> (Web service offered by Stanford University),
- Protege<sup>4</sup> (Java-based, downloadable, open source, many sample ontologies, from Stanford University),

---

<sup>1</sup><http://www.w3.org/TR/rdf-sparql-query/>

<sup>2</sup><http://kaon.semanticweb.org/>

<sup>3</sup><http://www.ksl.stanford.edu/software/ontolingua/>

<sup>4</sup><http://protege.stanford.edu/>

- OntoEdit<sup>5</sup> (Web service offered by the University of Karlsruhe)...

We made our choice based on a web page <sup>6</sup> and on the home page of each relevant editor. After this study we choose the Protege editor.

Protege is a free, open-source platform that provides a growing user community with a suite of tools to construct domain models and knowledge-based applications with ontologies. At its core, Protege implements a rich set of knowledge-modeling structures and actions that support the creation, visualization, and manipulation of ontologies in various representation formats. Protege can be customized to provide domain-friendly support for creating knowledge models and entering data. Further, Protege can be extended by way of a plug-in architecture and a Java-based Application Programming Interface (API) for building knowledge-based tools and applications. This extensibility of Protege is a great opportunity to develop the package management tools.

### 5.3.2 Implementation

This section presents informations about the implementation phase.

#### OWL for the product family

Here we illustrate the definition and usage of features and commonalities and discuss these aspects in more details. The conceptualization and definition of the domain establish a generic architecture for the product families that defines components, connectors and constraints. For the product family, the architecture serves as a reference point while creating each product. Variability must be specified explicitly in the architecture. As we saw it has been defined four types of features within the feature concept (see section 2.2.4). For the mandatory and optional feature we can replicate the method presented in the chapter 4.4.1. Nauticus Early Design (system that integrates computer aided design (CAD) and analysis, thereby enabling ship designers to perform contract and classification design faster and better) has to have as mandatory feature CAD. This assertion can be represented using *OWL* by the axiom:

```
<owl:Class rdf:about="#Nauticus">
  <owl:equivalentClass>
    <owl:Restriction>
      <owl:someValuesFrom rdf:resource="#CDA"/>
      <owl:onProperty rdf:resource="#hasMandatoryFeature"/>
    </owl:Restriction>
  </owl:equivalentClass>
```

---

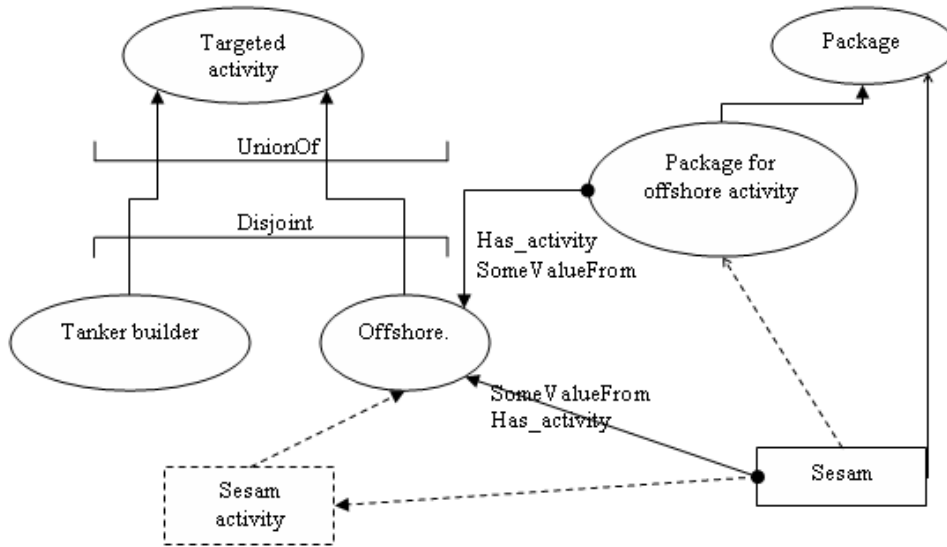
<sup>5</sup><http://www.ontoknowledge.org/tools/ontoedit.shtml>

<sup>6</sup>[www.xml.com/2002/11/06/Ontology\\_Editor\\_Survey.html](http://www.xml.com/2002/11/06/Ontology_Editor_Survey.html)



</owl:Class>

The optional features can be represented using the method presented by the W3C in the section 4.4.1. Alternative and multiple features we used our approach is based on the Pattern two in [27]. The section presents an example to illustrate how we represent the alternative features. In the example the targeted activity is the alternative feature. This feature has two sub features Tanker Builder and Offshore. Sesam is a package which targeted activity is offshore. The schema 5.7 represents how the ontology is constructed. The multiple features are represented using the same pattern but without the restriction disjoint for the sub classes.





**Legend:**


**Basic constructs:**


Ellipses represent classes            Squares represent instances      


**Arrows:**

Closed undecorated arrows represent `rdfs:subClassOf`      

Open undecorated arrows indicate `rdf:type`      

Arrows decorated with a blob on the origin indicate restrictions if between classes or facts if between individuals      

Dotted arrows indicate that the information represented is inferable by a reasoner and not present explicitly in the code.      

Upward facing square UnionOf symbols if spanning a set of `rdfs:subClassOf` arrows or `rdf:type` arrows indicate that the subclasses or individuals exhaust the class. This is expressed in OWL using `owl:unionOf` for classes or `owl:oneOf` for individuals      

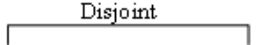
Downwards facing braces are used to indicate disjoint between subclasses or `owl:allDifferent` for individuals.      

Figure 5.7: Example of alternative feature.

License type	Factor A
Node locked	0.6
Dongle	0.7
Network	1.0

Table 5.2: Factor A table

### Rules

Rules are used to compute the price of the packages. Each package can have different pricing options to pass to the rules. A general concept that is applied to all the products is that the calculated price is found from multiplying the basic price of a product by factors. These factors can depend on the license type, the maintenance, the installation type etc. An example of pricing for the products of the family Nauticus is provided bellow. Nauticus is a family of solutions for ship early design integrating 3D computer assisted design (CAD) and ship analysis systems. The calculated price is computed using this formula:

$$Price = Basicprice * FactorA * FactorB * FactorC * FactorD \quad (5.1)$$

The factor A depends on the license type. The different types of license are:

- *Node locked.* This license allows one program to be accessed at any time on a designated PC.
- *Dongle license.* This license allows one program to be accessed at a time on the PC where the dongle is installed.
- *Network.* A n-User network license allows n copies of a licensed program to be accessed at any time.

Values for the factor A can be found using the table 5.2. The Factor B depends on the Type of license and on the number of users. For the network license the factor B can be found using the table 5.3. If the license type is dongle or node locked then the factor B can be found using the table 5.4. The factor C depends on the network installation. If the license type is not network then the factor c is 1.0 otherwise it can be found using the table 5.5.

The Factor D depends on the duration of the license. The different alternatives are:

- *Purchase.* For purchase the client is granted perpetual rights of use for the last program version received. Program maintenance and support for one year is included in the purchase price. The factor D is then 1.0.

No. of users in network	Individual factor	Factor B accumulated
1st user	1.0	1.0
2nd user	0.75	1.75
3rd user	0.5	2.25
4th user	0.4	2.65
5th user	0.4	3.05
6th - 10th user	0.3	
11th onwards	0.2	

Table 5.3: Factor B for network license

No. of users dongle or nor locked	Individual factor	Factor B accumulated
1st user	1.0	1.0
2nd user	0.75	1.75
3rd user	0.5	2.25
4th user	0.5	2.75
5th user	0.5	3.25
6th - onwards	0.4 each	

Table 5.4: Factor B for non network license

Network installation	Factor C
Single LAN	1.0
Additional LAN	Start counting at 2nd user on previous LAN
WAN	1.25

Table 5.5: Factor C table

- *Lease-Purchase.* A lease purchase contract is the same as a purchase contract, but the price is paid over 3 years. At the end of the period the client has the same perpetual rights of use as if purchase. In this case the Factor D is 0.52.
- *Annual Rental.* A rental agreement gives access to the programs for one year at a time. In this case the Factor D is 0.46.

The tables enable us to find the values of each factor. For example if a client wants to buy a Nauticus product with a Network License for 3 users, and he

wants to install the product in a WAN, and he wants to sign an annual rental contract then the price will be:

$$Price = Basicprice * 1.0 * 2.25 * 1.25 * 0.46 \quad (5.2)$$

### Rule implementation.

Rules express business knowledge. They are defined in a declarative language: the rule system enables administrators to express their domain knowledge and store them in a rule base which is used by the website component. The rule system consists of editor for editing rules and application interfaces as well as a runtime environment for rule execution. A rule is a conditional statement in a general form:

If <premise> then <consequent>

The rules are logically chained because predicate defined as the consequent in one rule is used as premise in another rule. The rule below define that if the license type is Node locked then the factor A is 0.6 and the consequent CalcA is true. If a Rule uses CalcA in the premise then the premise of this rule will be considered true.

```
<rule>
  <comment></comment>
  <rulename>NodeLockedLicense</rulename>
  <vardec>
    <varname>license</varname>
    <vartype>string</vartype>
  </vardec>
  <vardec>
    <varname>a</varname>
    <vartype>double</vartype>
  </vardec>
  <source></source>
  <priority>0</priority>
  <premise>
    <preactions />
    <atomic>?license=="node-locked";</atomic>
    <>trueactions>
      <action>?a:=0.6</action>
    </trueactions>
    <falseactions />
  </premise>
  <consequent>
    <named>
```

```
        <name>CalcA</name>
        <argument>?license</argument>
        <return>?a</return>
    </named>
</consequent>
</rule>
```

In the Nauticus case presented upon the rules to calculate the price need five arguments (basic price, license type, number of users, installation type, and contract type). For example one option for the rule for Nauticus is the License type that can have three values (Dongle, Node Locked, and Network). The web site component uses the options to display some graphical elements to the sales persons so it is mandatory that the web site is configured to figure out how to display the possible values for each option. For example the user can choose the license type from a list or he can just enter text in a TextBox. That is why we decided to save in the ontology the way to display each option. For example in the Nauticus example we decided that the license type has to be chosen from a list. In order to keep the web site code independent of the number and the type of the pricing options we made the choice that all the rule bases will have as starting rule a rule that follow this declaration: `double calcprice(HashTable pricingOptions, Double basicPrice)`.

### Web site

We implemented the web site using ASPNET and CSHARP. The main challenge was to learn a new language. We had some difficulties to understand the page life cycle and client side scripts. But the help provided by all the people at DNV Software allowed us to carry out the task. The web site uses the ontology as a database component. All the work done with the ontology is done using the SemWeb toolkit (see section 5.3.1). Some screen shots of the web sites are provided in appendix (see appendix E.1).

### Package creation tool

The package creation is a piece of software that enables to create new packages based on the products family architecture. We agreed with the sales person at DNV that the package creation tool (or Configurator) has not to be web based. That is why this tools is a local tool. We used the protege API to do the development. The mains problems that we have encountered are to understand how to use the protege API and how to link the data to JTable model. Some screen shots of the web sites are provided in appendix (see appendix E.2.1).

## 5.4 Summary

Our approach differs from existing tools that supports the product family since we are using the semantic power of OWL to support the management and the pricing of product families, whereas most of them are based on XML. The ontology is the central component of our system it encodes the domain knowledge and restrictions. The product family pricing calculator is divided in two distinct parts:

- The management of families architectures and packages which is done by an administrator using the package management tools. The package management tools support the domain engineering doing ontology management with an ontology editor, but it also supports the product engineering using a configurator.
- The usage of the knowledge encoded in the ontology. The ontology is used as operational data to make pricing offers. The web site component allows application users (i.e. sales people) to find, and price packages.

This approach enables to separate the product family architecture from its usage (for example a price calculation). Our system is ontology-driven and rules based system that allows to manage composition and to price packages. Next step is to evaluate and to analyze the results of the implementation to figure out if the system has fulfilled the requirements and needs and whether it helps to solve current issues in managing family of products.





## Chapter 6

---

# Evaluation of the approach and system

---

This chapter evaluates the results of running our system on different problem scenarios. This evaluation is done based on the unsolved problems by the existing solutions that support family of products (see section 3.2) and on the potential benefits to use semantic technologies (see section 4.1.2) and rules (see section 4.3.2). This chapter also analyzes those results. Analysis consists of both pointing out interesting result and trying to explain them.

### 6.1 Objectives

Our project is driven by the hypothesis that the semantic technologies can support the family of products. Our method was to find how the unsolved problems of the current solutions to support the family of products can be tackled by the usage of the semantic technologies. Based on a practical case we would like to verify this hypothesis by comparing it with the observed outcomes. We seek to figure out some best practices to:

- Improve the variability consideration within the products family architecture.
- Minimize redundancy of feature models when two family have a high level similarity.
- Ensure consistency of the feature model.

We consider at least in part our case study as an exploratory case study as it is based on a client demand and it might be lead to a large scale implemen-

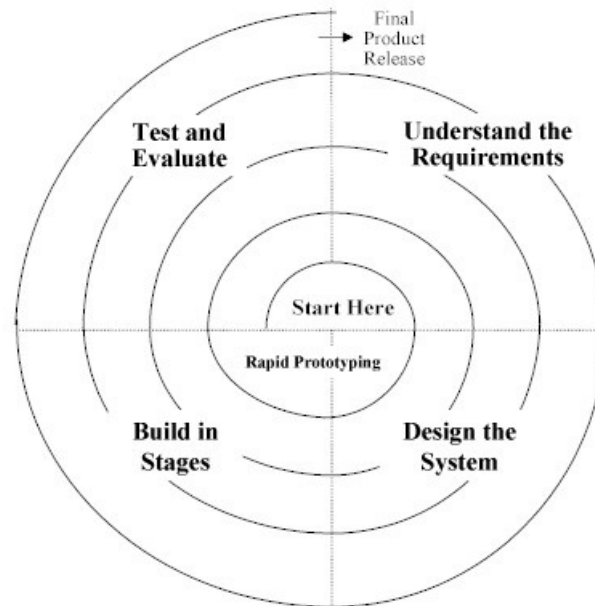


Figure 6.1: Spiral development method

tation. Exploratory case studies help to identify questions, select measurement constructs, and develop measures; they also serve to safeguard investment in larger studies. In order not to make premature or inappropriate conclusion we would like to use our project to identify potential future research directions.

## 6.2 Results

Goal of this section is to present the relevant users remarks done testing the prototype (see section 6.2.1) and attempts to analyze them.

### 6.2.1 Incremental development and evaluation

The project has been conducted using spiral development method [8] which is composed of four steps:

- Understanding the requirements.
- Prototype conception.
- Prototype building phase.
- Prototype test.

This method is very useful because it allows to take in account the customer remarks. In this method the requirements are not defined during the first phase

of the project. They are in an evolution process, in fact they become more and more precise while the time goes. Following this method we had bi-weekly meetings with the clients. During these meetings problems and issues together with possible solutions were discussed. Our evaluation aims to figure out the pro and cons of our approach. To achieve this goal we did a qualitative evaluation confronting users to the prototypes to allow to validate the concepts. Our evaluation method has been based on tests and questionnaires.

### 6.2.2 Final prototype validation

This section presents the qualitative evaluation of the final prototype made by the users, i.e. analyze and discussion of the feedback from the users (customer). During this evaluation we try to relate evaluation questionnaire and analysis to the requirements and needs (see section 2.3) and unsolved problems/current issues (see section 3.2). The evaluation has been done using test cases and questionnaires. This section will only present the relevant tests according to our research objectives. Detailed tests are available in the appendix F. In this section the test are presented using the template table (see table 6.1).

Test name	_____
Summary	_____
Assigned requirements	_____
Results	_____
Remarks	_____

Table 6.1: Test template

#### Package selection

This section presents the relevant test to evaluation the selection of one package that correspond to the use case UC04. This functionality is important for the system as it assigns the high requirement R01 and because it is used to create pricing offers. We did some tests to evaluate the reactions of the system to users actions in abnormal cases. Meanwhile in order to present the package selection search capability we will just present a general test. This test is described bellow.

Test name	Package selection test
Summary	This test is a search of one package based on a set of features.
Assigned requirement	R01
Results	The system helps users to find a convenient package.
Remarks	Some users think that it should be good to update the tree of features to reflect the current features selection (some features may exclude or require others features). In lot of case the sales people know at least partially what package they are looking for (i.e. from which family the package is a member) so it should be possible to select a family and then update the three of features according to the family architecture.

Table 6.2: Package selection test

### Price calculation

The price calculation is an important part of the project as it has been among the roots reasons that bring DNV to offer us the case study. Requirements concerning the pricing became more and more precise while the time goes. The first general requirements were have been defined in early stage of the project (see section 2.3). The validation of previous version of the prototype released new requirements such as "The system to allow to specify custom pricing options for each tool that compose a package". Our set of tests consequently changed. We tested the requirements related to the pricing in collaboration with the users. The most relevant tests are presented bellow.

### Package creation

The package creation corresponds to product engineering phase defined in the family of product engineering (see section 2.2.4). Our system support it using a configurator tool. The most relevant tests to verify that our system covers the requirements and needs that we figured out in the settings and background section 2.3.3 are presented bellow.

Test name	Normal pricing calculation
Summary	In this test we evaluate the ability of our system to create correct pricing offers
Assigned requirements	R02, R03, R04, R07, R09
Results	Only the price of tools include in the offer is computed. The prices are correct according to the rules.
Remarks	The execution speed can be slow some time.

Table 6.3: Normal pricing calculation test

Test name	Web site adaption to a change in pricing scheme
Summary	In this test we evaluate the ability of the web site to adapt to a change in the pricing scheme.
Assigned requirements	R02, R03, R04, R06, R07, R09
Assigned needs	N01, N02, N05
Result	The web site can adapt itself to a change in the pricing options. The prices computed reflect the changes made in the pricing rules.
Remarks	None.

Table 6.4: Web site adaption test

### Family creation

The family creation or modification corresponds to the domain engineering phase (see section 2.2.4). During our meetings the client expressed the wish that only one person could create new family of products. We did tests to create a family, to add restriction to a family, to test the consistency of families architecture and to test redundancy between two similar families. All the family architecture management corresponds to ontology management.

Test name	Customize pricing options test
Summary	This test enables to check if the system enables to have specific pricing options (i. e. customize options) for each tool within a package composition.
Assigned requirements	R02, R03, R04, R07
Result	The pricing is done according to the customize options if defined.
Remarks	None.

Table 6.5: Customize pricing options test

Test name	Package consistency test
Summary	The test verifies that the system insures the consistency of the packages
Assigned requirements	R05
Assigned needs	N01, N04
Results	The system supports the product engineering. The system insures the consistency of the packages.
Remarks	Discussions happened in order to define if the set of features describing the packages has to be included in the set of features supporting by the tools or not (i.e. Is a composition of tools superior in terms of features supported to the sum of the features supported by each tool?)

Table 6.6: Package consistency test

### Pricing rules definition

The pricing rules are defined using the DNV rule engine. The test of the rules editor has been done by the DNV BRIX development team. We based our evaluation on the experience of the DNV's engineers. It should be easy to define rules using the rule editor as we figured out that the system has to allow business knowledge to be maintained by domain experts without requiring programming skills. The

Test name	Variability support during the configuration
Summary	This test verifies if the family architecture is taken in account during the creation of a new package (i.e. variability during the product engineering)
Assigned requirements	R05
Assigned needs	N03
Results	The system supports the product engineering. The family architecture is respected but the variability support needs to be improved.
Remarks	The users have to know by advance the family architecture in order to be able to modify the preselected set of features as the configurator does not indicate the feature type. It implies that the people that create the architecture and the new packages have to be the same.

Table 6.7: Variability support during the configuration

main problem with the rules is not technical-aspect related, but mainly to define them. In fact it has been difficult to find domain experts that are able to define the rules.

### Results analysis

Goal of this section is to analyze the results from the final prototype evaluation, to create the links between the results, the requirements (see section 2.3), the needs (see section 2.3.3) and the unsolved problems (see section 3.2). This section will be the basis for the next chapter (i.e. conclusion and further work).

#### Fulfilled Requirements

Here the objective is to show that the prototype answers to the client's requirements. This part of the analysis presents a table (see table 6.12) that summarizes which requirements are supported by which functionalities. The requirements R06, R07 are partially fulfilled. Because only the web site component is web based, the package creation and family management are not. The requirement R08 is not supported as the system does not provide any interface with Salesforce.

#### Supported needs

Test name	Family creation test
Summary	The goal of this test is to find out if our system supports the domain engineering and if it is easy to manage the families.
Assigned requirements	R05
Assigned needs	N01
Results	The system supports the domain engineering. It is possible to create new family and to instantiate it.
Remarks	The users expressed difficulties to understand the logic in the ontology.

Table 6.8: Family creation test

Test name	Feature support test
Summary	The goal of this test is to check the ability of the system to support a change in the features within a family architecture.
Assigned requirements	R05
Assigned needs	N01, N03
Results	The system supports the domain engineering. It is possible to change a family architecture (i.e. add features).
Remarks	The users have to have knowledge about the patterns to use in the ontology to create new features.

Table 6.9: Family creation test

This section presents the links between the needs and the final version of the prototype. Our system:

- Supports the family of products engineering (see section 2.2.4). The product engineering is supported by our configuration tool whereas the domain engineering is supported using the ontology editor.
- Allows the business knowledge to be maintained by domain experts without requiring programming skills using the rule editor (see figure 2.4), and one



Test name	Family consistency test
Summary	This test aims to verify if the system enables to find inconsistency in the family architecture.
Assigned requirements Assigned requirements	R05 N01
Results	Our system helps to find inconsistencies.
Remarks	The users said that it is difficult to understand why a family is inconsistent. Sometime it is challenging to find the reason of the inconsistency error.

Table 6.10: Family consistency test

Test name	Redundancy test
Summary	The goal of this test is to verify how the system helps to handle redundancy between to family architectures.
Assigned requirements	None
Results	The system infers the best structure (i.e taxonomy) in order to minimize the redundancy between two very similar families
Remarks	The system helps to find redundancy but it does not correct it. The user has to correct the problem himself.

Table 6.11: Redundancy test

ontology editor.

- Enables to automatically propagate a change in the family architecture or in the pricing rules to all applications.
- Provides a rigorous pricing structure. But the client choose to have only one knowledge worker (i.e. the administrator), it implies that the rules are easy to trust if the administrator is trusted. Meanwhile it might be a problem to find one person that centralize all the knowledge of the domain, maybe the domain engineering should be supported by a collaborative task.

Requirement Id	Functionality that support the requirement
R01	Feature based search (see screen shot E.3).
R02	Pricing rules.
R03	Pricing calculation (runtime compilation and execution of the rules.)
R04	Brix rule engine is used for the pricing and the Brix web framework is used for the web site component.
R05	Role of administrator (knowledge worker) and sales person (web site user).

Table 6.12: Fulfilled requirements table

### Problems solved

Our approach helps to solve some current issues in supporting the family of products. The tests have shown that the consistency of the data is improved by using semantic technologies. The consistency of a family of products can be checked using a reasoner. Moreover the system forbids to create incoherent packages using our configurator tool.

Our prototype helps also to solve numerous problems of the feature models. It helps to reduce the redundancy between two very similar families by inferring the best family hierarchy structure. If this structure is adopted it will decrease the redundancies by using the inheritance mechanism.

The variability within the family of products concept is well supported even if it can be improved (see remarks in the tests). The usage of semantic technologies and rules open new possibilities without any cons.

### Functionalities improvement

The final prototype evaluation has shown that our approach supports all the high prioritized requirements and needs. However, some improvements of the functionalities are needed. First some remarks made during the tests lead us to think that the features based search offered by the web site component (see screen shot E.3) have to be modified. The feature based search should be improved by handling the events of selection of one feature or one family. In fact if the selection of a feature occurs then the feature tree will need to be updated to reflect this selection. For example if the user select the features diesel for a motor it should be impossible to select an other fuel type, so we have to remove the other fuel types from the features tree. In the same way it should be possible to select a family and automatically update the tree of features according to the features describing the family architecture.

Secondly the configurator tool helps to create new packages; the process to

create a new package is first to select a family architecture and then the system displays the features corresponding to the family architecture. The configurator could be improved by offering a better support of the different type of features. For example instead of selecting all the alternative features it should interact with the user and ask which optional features have to be selected. To achieve this improvement the configurator has to use more the semantics defined in the ontology.

Finally our system support the creation of new families with an ontology editor but it seems that there is a need to mask the concepts of the OWL language. In fact the creation of new family and new feature implies to use restrictions like *disjoint* or *SomeValueFrom* that are difficult to understand by non computer scientists. A plugging to the ontology editor to manage the architectures of the families could improve the domain engineering support. This plugging should help users to create all type of features and restrictions (see section 2.2.4).

### 6.3 Summary

The evaluation has been done using a spiral method (see section 6.2.1) to verify that our approach can help to support the product family. We did a qualitative evaluation of our work. By completing tests the client allowed us to find out which requirements and needs are supported by our system. Moreover based on the remarks made during the tests we also figured out which functionalities needed to be improved. Next step is to conclude our work and to express future work to be done.



## Chapter 7

---

# Conclusions and future work

---

This chapter concludes our work. Our hope is to help others who are interested in a similar problem, both by analyzing the successes and the failures of our approach. In the last part of the chapter figures out some directions and further work for the field of study.

### 7.1 Discussion and future work

Goal of this section is to highlight the pro and cons of our approach but also to find out some directions for further research.

Our approach highlights some direct benefits adopting semantic web technologies to manage and price product families. For example the lack of a formal semantics and reasoning support of feature models has hindered the development of solution to support family of products. Industrial experiences show that methods and tools that can support feature model analysis are badly appreciated. But by representing feature models as OWL axioms (see section 5.3.2), the consistency of the family architecture can be automatically checked. Moreover this enable the configurator tool to create only consistent packages. The problem of redundancy between highly similar families is solved using a reasoner to infer the best family taxonomy and using the inheritance mechanism. It seems that the semantic technologies really help to handle the family of products concept by the family architectures consistency and by improving the coherence of the new product created by the product engineering phase.

Our approach has shown some shortcomings of using semantic technology. First after the tests of the pricing of the packages it has been remarked that the execution speed can be low. It can be a problem in a system that interacts directly with components as DNV BRIX or DNV Rule Engine. Further work should investigate the impact of the system architecture on the system's performance to

find the source for the low execution speed.

Other drawbacks come directly from the OWL syntax. For example, OWL does not provide explicit declarations of ontology entities. This caused numerous misunderstandings in the past, and it makes defining an intuitive structural integrity check for OWL ontologies more difficult.

Working on this project we saw the difficulty to develop software using semantic technologies. There is a gap between "traditional" programming and knowledge engineering which is getting wider and wider. Traditional programmers and knowledge engineers often don't understand each other. We felt the need to change the traditional development method to adapt to the semantic web paradigm. In fact it has been difficult to find knowledge experts to define the concepts of the ontology and pricing rules. Moreover it has been quite difficult to find toolkit to integrate semantic technologies to classic programming.

There is a large amount of research concerning the way to derive the products family architecture. But there are very few works that study the impact of semantic technologies on this process. In our approach the semantic technology does not model the process to derivate the family architecture but influences it (see section 5.2.4). This area might need some further investigation.

## 7.2 Conclusion

The boundaries of the project did not include the evolution of the variability in the time. But the ontology evolution and versioning is a huge field of research. Moreover it is coupled with the product family architecture evolution. Therefore, one of the certain future works is to enable our system to manage co-evolution of ontology and architecture of product family, in order to find some best practices in this domain. This research project started out with the goal of developing a product family management using semantic technologies. Based on the evaluation of the final prototype, we can quite confidently say that we are on the right track. Because the flexibility in defining knowledge and in supporting family engineering is a very important criterion, the use of semantic technology and rules is a solution to this problem. The main research question of this project was to find technologies that can offer a better support for the family of product concept than the existing solutions. We came out with a method to find requirements, needs and unsolved problems in order to figure out which technologies are the most suitable for the management and pricing of product family. Our method has resulted in a prototype based on a case study provided by DNV Software. The prototype was the first step. We showed that we have had a working architecture to create such a system. The next step now would be to further develop the product family pricing calculator to make it become convenient for an industrial usage in DNV software. Our system encodes the knowledge in an ontology but

it also encodes the way to process this knowledge in rules. As a consequence our approach provides value to the products family enabling to improve the quality of features models and to improve flexibility in managing knowledge.





---

# Bibliography

---

- [1] *ESAPS project* <http://www.esi.es/esaps/>, 1999. [cited at p. 9]
- [2] *CAFE project*, <http://www.esi.es/en/projects/cafecafe.html>, 2001. [cited at p. 9]
- [3] *FAMILIES project*, <http://www.esi.es/en/projects/families/>, 2003. [cited at p. 9]
- [4] D. M. Anderson. *Build-to-Order Mass Customization, the Ultimate Supply Chain and Lean Manufacturing Strategy for Low-Cost On-Demand Production without Forecasts or Inventory*. 2004. [cited at p. 1]
- [5] G. Antoniou, E. Franconi, and F. van Harmelen. Introduction to semantic web ontology languages. In *Reasoning Web, Proceedings of the Summer School, Malta, 2005*, number 3564 in Lecture Notes in Computer Science, Berlin, Heidelberg, New York, Tokyo, 2005. Springer-Verlag. [cited at p. 26, 28, 29]
- [6] T. Asikainen, T. Mnnist, and T. Soininen. Kumbang: A domain ontology for modelling variability in software product families. *Adv. Eng. Inform.*, 21:23–40, 2007. [cited at p. 20]
- [7] S. Bechhofer, C. Goble, and I. Horrocks. *DAML+OIL is not enough*. 2001. Presented at First Semantic Web Working Symposium. [cited at p. 28]
- [8] B.W. Boehm. A spiral model of software development and enhancement. 1998. [cited at p. 60]
- [9] J. Bosch, G. Florijn, D Greefhorst, J. Kuusela, H. Obbink, and K. Pohl. Variability issues in software product lines. page 1119. Fourth workshop on Product Family Engineering, 2001. [cited at p. 10]
- [10] S. Buhne, K. Lauenroth, and K. Pohl. Why is it not sufficient to model requirements variability with feature models? Technical report, Software Systems Engineering Institute for Computer Science and Business Information Systems (ICB) University of Duisburg-Essen, 45117 Essen, Germany. [cited at p. 21]
- [11] V. Dedebean and D. Strasunskas. An ontology-centric approach for flexible configuration and pricing of product families. 2007. [cited at p. 2]
- [12] W.U. Eisenecker and Czarnecki. *Generative Programming : Methods, Tools, and Applications*. 2000. [cited at p. 11]

- [13] C. Fillies, G. Wood-Albrecht, and F. Weichardt. *A pragmatic application of the semantic Web using SemTalk, Proc.Intl World Wide Web Conf (WWW)*. 2002. [cited at p. 26]
- [14] J. Freeman-Hargis. Rule-based systems and identification trees: Introduction to rule-based systems. 2005. [cited at p. 31, 33, 37, 122]
- [15] N. Guarino. Formal ontology and information systems. In *Proceedings of FOIS98*, pages 3–15, June 1998. [cited at p. 27]
- [16] A. Gunter and C. Kuhn. *Knowledge-Based Configuration - Survey and Future Directions*. Germany, 1999. [cited at p. 13]
- [17] S. Haitham and Hamza. On the impact of domain dynamics on product-line development. Technical report, Department of Computer Science and Engineering University of Nebraska, Lincoln, NE 68588-0115. [cited at p. 20]
- [18] J. Hendler and D. L. McGuinness. *The DARPA Agent Markup Language*. IEEE Intelligent Systems, November 2000. [cited at p. 28]
- [19] I. Herman. Introduction to the semantic web (tutorial). Technical report, International Conference on Semantic Web Digital Libraries, Bangalore India, February 2007. [cited at p. 23, 26]
- [20] J-M. Jzquel. 2nd international conference on aspect-oriented software development. In *Model-driven engineering with contracts, patterns, and aspects*, March 2003. [cited at p. 10]
- [21] K. Kang and al. Feature-oriented domain analysis (foda) feasibility study. Technical report, Software Eng. Inst and Carnegie Mellon Univ Pittsburgh, 1990. [cited at p. 11, 12]
- [22] P. Lago and H. van Vliet. Observations from the recovery of a software product family. Technical report, Vrije Universiteit, Amsterdam, The Netherlands. [cited at p. 21]
- [23] Mitsuru, Kazuhisa, Osamu, and Riichiro. Task ontology: Ontology for building conceptual problem solving models. [cited at p. 28]
- [24] I. Nonaka and H. Takeushi. The knowledge-creating company. Technical report, 1995. [cited at p. 26]
- [25] N.F. Noy and M.A. Musen. Ontology versioning in an ontology management framework. *IEEE Intelligent Systems 19 (4)*, pages 6–13, July 2004. [cited at p. 46]
- [26] D.L Parnas. *On the design and development of program families*. IEEE Transactions on Software Engineering, March 1976. [cited at p. 9]
- [27] A. Rector. Representing specified values in owl: value partitions and value sets. Technical report, W3C Working Group, May 2005. [cited at p. 51]
- [28] T. Soinen, J. Tiihonen, T. Mnnist, and R. Sulonen. Towards a general ontology of configuration, artificial intelligence for engineering design, analysis and manufacturing. *Cambridge University Press*, pages 357–372, December 1998. [cited at p. 13]

- [29] M. Uschold and R. Jasper. A framework for understanding and classifying ontology applications. Technical report, The Boeing Company, 1999. [cited at p. 43]
- [30] I. Vanderfeesten, W. van der Aalst, and H. Reijers. Modelling a product based workflow system in cpn tools. Technical report, Technische Universiteit Eindhoven, Department of Technology Management, PO Box 513, 5600 MB Eindhoven, The Netherlands. [cited at p. 21]
- [31] H. Wang, L. Y. Fang, J. Sun, and H. Zhang. A semantic web approach to feature modeling and verification. In *1st Workshop on Semantic Web Enabled Software Engineering (SWESE'05)*, Galway, Ireland, Nov 2005. LNCS, Springer-Verlag. [cited at p. 35]
- [32] H. Wang, Y.F. Li, J. Sun, H. Zhang, and J. Pan. Verifying feature models using owl. *Journal of Web Semantics*. [cited at p. 35]
- [33] M.D. Weiss and C.T. Robert Lai. *Software Product-Line Engineering : A Family-Based Software Development Process*. Addison-Wisley, 1999. [cited at p. 9]
- [34] Y. Zhao and K. Sandahl. Potential advantages of semantic web for internet commerce. Technical report, Dept of Computer and Information Science, Linkping University, S-58183, Linkping, Sweden. [cited at p. 25]



# Appendices



## Appendix A

---

# Thesis resources and constraints

---

### A.1 Resources

To help with carrying out the project, the resources below are provided:

- Laptop with administrator privilege,
- Full access to all the source code in DNV software,
- Visual studio 2005 and the Perforce source control.
- Help provided by the engineers working for DNV.

### A.2 Economy

The economy of the project lies in terms of hour use. Based on an average number of hours per week and on the duration of the master thesis, 880 hours for the all task, sets the economic boundaries of the project.

### A.3 Thesis lifespan

The thesis starts the 9th of January and ends the 6th of June 2007. Deadline for project report printing is set to the 6th of June 2007.





## Appendix B

---

# Detailed use cases

---

This appendix presents the detailed use cases (UC) that have been used for the implementation phase. Each use case is described using a template. This template includes:

- *Use Case Name.* The use case name provides a unique identifier for the use case.
- *Summary.* The summary section is used to capture the essence of the use case.
- *Preconditions.* A preconditions section is used to convey any conditions that must be true when a user initiates a use case.
- *Triggers.* The Triggers section describes the starting condition(s) which cause a use case to be initiated.
- *Basic course of events.*
- *Alternative paths.* Exceptions, or what happens when things go wrong, are described in the alternative paths.
- *Postconditions.* The post-conditions section summarizes the state of affairs after the scenario is complete. A scenario guarantees that its post-conditions are true at the end of the scenario.

### B.1 UC:Create offer

**Use Case Name:** Create Offer.

**Summary:** This use case defines the process to create a new offer.

**Preconditions:** None

**Triggers:** The use case is trigger by the user.

**Basic course of events:**

1. The user enters a name for the offer to create.
2. The system creates the offer and adds it to the available open offers.

**Alternative paths:**

1. The system detects that the name entered by the user is not correct.
2. The system displays an alert to inform the user that the offer cannot be created.
3. Go to step 1 (Main Path)

**Postconditions:** A new offer is created and it is open and available to be selected.

## B.2 UC:Delete offer

**Use Case Name:** Delete Offer.

**Summary:** This use case defines the process to delete an offer.

**Preconditions:** The offer to delete has to be opened.

**Triggers:** The use case is trigger by the user.

**Basic course of events:**

1. The user chooses an offer to delete.
2. The system deletes the offer.

**Alternative paths:** None

**Postconditions:** The offer does not exist anymore.

## B.3 UC:Save offer

**Use Case Name:** Save Offer.

**Summary:** This use case defines the process to save an offer.

**Preconditions:** None

**Triggers:** The use case is trigger by the user.

**Basic course of events:**

1. The user chooses an offer to save.

2. The system saves the offer.

**Alternative paths:**

1. The system detects that an offer that has the same name already exists.
2. The system replaces the old offer by the new one.

**Postconditions:** The offer is saved on the server and it is available to be opened later.

## B.4 UC:Open offer

**Use Case Name:** Open Offer.

**Summary:** This use case defines the process to open an offer.

**Preconditions:** None

**Triggers:** The use case is trigger by the user.

**Basic course of events:**

1. The user click on a button that opens his file system.
2. The user chooses one offer in his file system.
3. The system opens the offer.

**Alternative paths:**

First alternative case:

1. The file chosen by the user is not an offer.
2. The system shows a pop up message that informs the user that the file does not correspond to an offer.
3. Go to step 1 (Main path).

Second alternative case:

1. The file chosen by the user is an offer and an offer that has the same name is already in the open offers list.
2. The system ask if the user wants to replace the open offer by the new offer.
3. If the response is yes then the system deletes the old offer.
4. Go to step 3 (Main path).

**Postconditions:** The offer is opened and available in the open offers list to be selected later.

## B.5 UC:Select offer

**Use Case Name:** Select offer.

**Summary:** This use case defines the process to select an open offer.

**Preconditions:** The offer to select is already in the open offers list

**Triggers:** The use case is trigger by the user.

**Basic course of events:**

1. The user click on the offer to select.
2. The system shows the details of the selected offer.

**Alternative paths:** None

**Postconditions:** The offer is selected.

## B.6 UC:Modify offer

**Use Case Name:** Modify Offer.

**Summary:** This use case defines the process to modify an offer.

**Preconditions:** the offer is selected.

**Triggers:** The use case is trigger by the user.

**Basic course of events:**

1. The user changes the client's name and address and or the offer name.
2. The user adds or changes the packages within the offer.
3. The user saves the changes made to the offer.
4. The system saves the offer.

**Alternative paths:**

1. The system detects that the name entered by for the offer the user is not correct.
2. The system displays an alert to inform the user that the offer cannot be created.
3. Go to step 1 (Main Path)

**Postconditions:** The modification made are saved and the modified offer is available.

## B.7 UC:Add package to offer

**Use Case Name:** Add package to Offer.

**Summary:** This use case defines the process to add a package to the selected offer.

**Preconditions:** the offer is selected.

**Triggers:** The use case is trigger by the user.

**Basic course of events:**

1. The user chooses to add a package to the offer.
2. The user computes the price for one package.
3. The system adds the package to the offer.

**Alternative paths:** None

**Postconditions:** The package is added to the selected offer.

## B.8 UC:Price package

**Use Case Name:** Price package.

**Summary:** This use case defines the process to compute the price of one package.

**Preconditions:** the package which price has to be computed must be displayed.

**Triggers:** The use case is trigger by the user.

**Basic course of events:**

1. The user selects the tools that he wants to include in the pricing calculation.
2. The user defines the pricing options to apply to calculate the price.
3. The user saves the changes made to the offer.
4. The system computes the price of the package and displays it.

**Alternative paths:**

1. The pricing options are not correct.
2. The system displays an alert to inform the user that the price cannot be computed.
3. Go to step 2 (Main Path)

**Postconditions:** The pricing options are defined and the price can be recompute when needed.

## B.9 UC:Select package

**Use Case Name:** Select package.

**Summary:** This use case defines the process to select a package in the branding structure.

**Preconditions:** none.

**Triggers:** The use case is trigger by the user.

**Basic course of events:**

1. The user chooses a set of features that the desired packages has to fulfill.
2. The system infers which packages can fulfill the set of features selected by the user.
3. The user selects one package.
4. The system displays the package composition.

**Alternative paths:**

First alternative path:

1. The user selects more than one package.
2. The system displays an alert to inform the user only one package at the time can be displayed.
3. Go to step 2 (Main Path)

Second alternative path:

1. The system infers that there is no package that corresponds to the selected features.
2. Go to step 1 (Main path).

**Postconditions:** The package composition is displayed to the user who can compute the price for the package.

## B.10 UC:Generate report

**Use Case Name:** Generate report.

**Summary:** This use case defines the process to generate a report from one offer.

**Preconditions:** the offer is selected.

**Triggers:** The use case is trigger by the user.

**Basic course of events:**

1. The user chooses to create a report.
2. The user chooses where in his file system he wants to save the file.
3. The user chooses a name for the file to save.
4. The system saves the report.

**Alternative paths:** None

**Postconditions:** The report is available in format .doc.

## B.11 UC:Manage Users

This use case included three use cases:

- Create a user.
- Delete a user.
- Manage the roles of one user.

### B.11.1 Create user

**Use Case Name:** Create User.

**Summary:** This use case defines the interactions between an administrator and the system to create a new user.

**Preconditions:** The user is an administrator.

**Triggers:** The use case is trigger by the administrator.

**Basic course of events:**

1. The administrator chooses to create a new user.
2. The administrator enters the informations concerning the user to be created (login, password,email ...).
3. The user chooses a set of role for the new user (administrator, salesperson).
4. The administrator creates the new user account.

**Alternative paths:**

1. The system cannot create the new user account.
2. The system displays a warning to the administrator.
3. Go to step 2 (Main path).

**Postconditions:** The user account is created and the new user can use the web site.

### **B.11.2 Delete user**

**Use Case Name:** Delete User.

**Summary:** This use case defines the interactions between an administrator and the system to delete one user.

**Preconditions:** The user is an administrator.

**Triggers:** The use case is triggered by the administrator.

**Basic course of events:**

1. The administrator chooses the user account to delete.
2. The system deletes the accounts.

**Alternative paths:** None

**Postconditions:** The user account is deleted.

### **B.11.3 Manage the roles of one user**

**Use Case Name:** Manage roles.

**Summary:** This use case defines the interactions between an administrator and the system to manage the roles of one user.

**Preconditions:** The user is an administrator.

**Triggers:** The use case is triggered by the administrator.

**Basic course of events:**

1. The administrator chooses the user account to modify.
2. The system displays a checkbox list of possible roles (the selected user's roles are checked).
3. The administrator chooses the set of roles to affect to the account.
4. The system modifies the roles of the selected user account.

**Alternative paths:** None

**Postconditions:** The roles of the user account are changed.



## B.12 UC:Create package

**Use Case Name:** Create package.

**Summary:** This use case defines how an administrator can create a new package.

**Preconditions:** The user is an administrator.

**Triggers:** The use case is trigger by the administrator.

**Basic course of events:**

1. The administrator chooses the place of the package to create in the branding structure.
2. The administrator enters the name of the package.
3. The administrator chooses a set of features to define the new package.
4. The system infers the tools that correspond to the feature and adds them to the package composition.
5. The administrator creates the pricing rules and options for the packages (or it will inherited the rules and the options of its parent package).
6. The system saves the new package.

**Alternative paths:**

1. The set of tools inferred by the system does not correspond to the administrator needs.
2. The administrator modifies the set of features and the set of tools of the package.
3. The system checks the concordance between the set of tools and the set of features.
4. If the previous step is ok then Go to step 5 (Main path).

**Postconditions:** The package is created and available to be used in the offers that the sales people will make.

## B.13 UC>Delete package

**Use Case Name:** Delete package.

**Summary:** This use case defines how an administrator can delete package.

**Preconditions:** The user is an administrator.

**Triggers:** The use case is trigger by the administrator.

**Basic course of events:**

1. The administrator chooses a package.
2. The administrator chooses if he wants to delete the sub packages of the package chosen package.
3. The system delete the package and the hierarchy of sub packages of the package and creates a new version of the ontology.

**Alternative paths:**

1. The administrator decides not to delete the sub packages of the package.
2. The system by a system of "bootstrap" attaches the sub packages of the selected package to the parent of the selected package. and create a new version of the ontology.
3. Go to step 3 (Main path).

**Postconditions:** The package is deleted.

## **B.14 UC:Update package**

This package is not detailed because the interactions between the administrator and the system are nearly the same than in the use cases Create package and Delete package. The use case update package is just a combination of the two previous use cases.

## Appendix C

---

# Components interaction

---

This appendix presents sequence diagrams (see figures C.1 and C.2) based on concrete scenarios. They show the interaction between objects over the progression of time.

### C.1 Price a package

This section presents a sequence diagram based on a usage scenario. Goal of this sequence diagram is to show the interaction between the sales people, the web site and the ontology. The scenario's steps from a sales person point of view are:

1. Login into the web site.
2. Select a set of features that describe the searched package.
3. Start the search.
4. Select one result of the search.
5. Enter pricing options and choose a composition for the pricing.
6. Click compute pricing.

### C.2 Create a package

This section presents a sequence diagram based on a usage scenario. Goal of this sequence diagram is to show the interaction between an administrator, the configuration and the ontology. The scenario's steps for an administrator are:

1. Star the configurator.

2. Select a family architecture basis.
3. Start the creation of a new package.
4. Click find corresponding tools.
5. Click create new package.

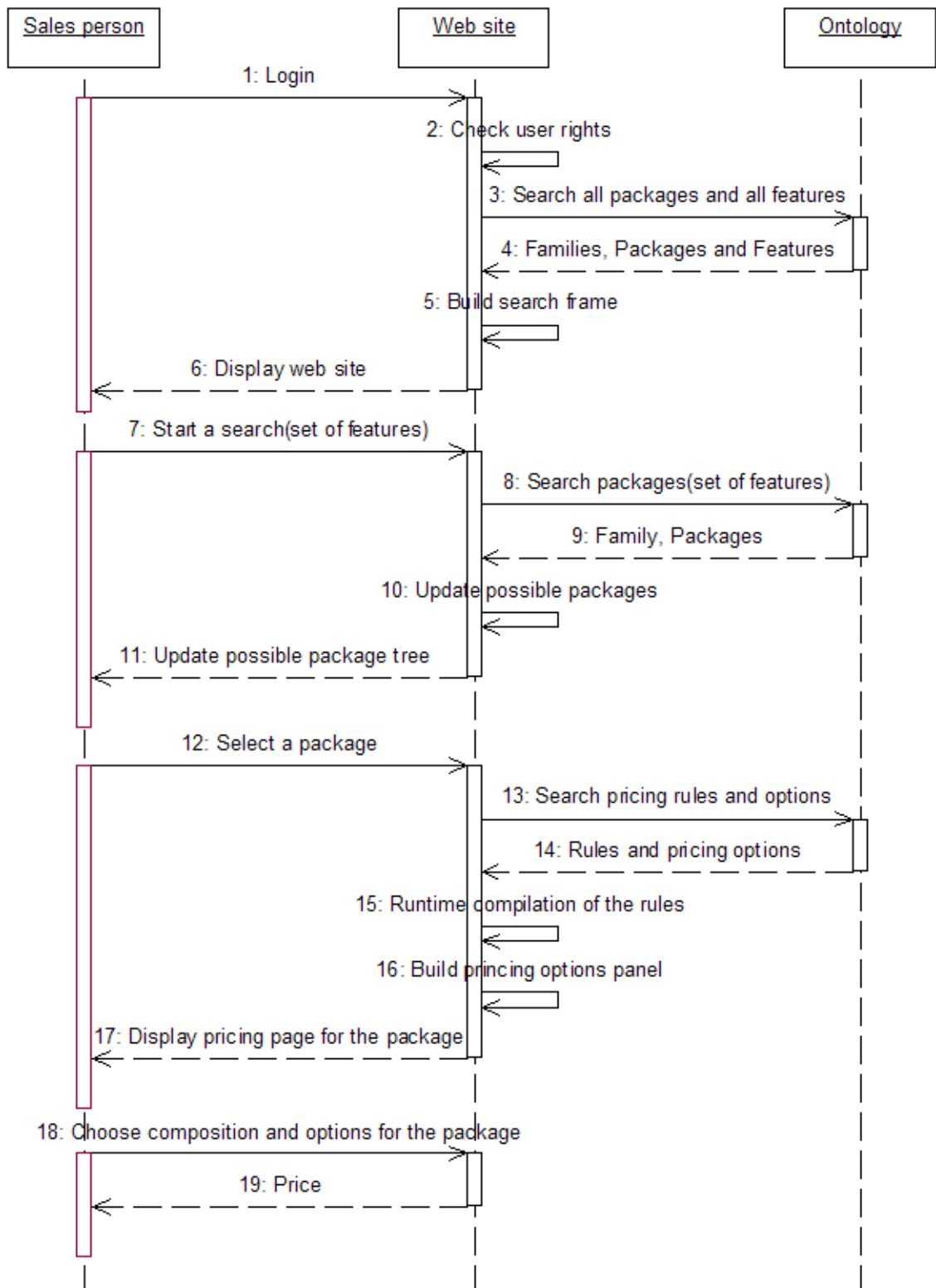


Figure C.1: Sequence diagram for pricing a package

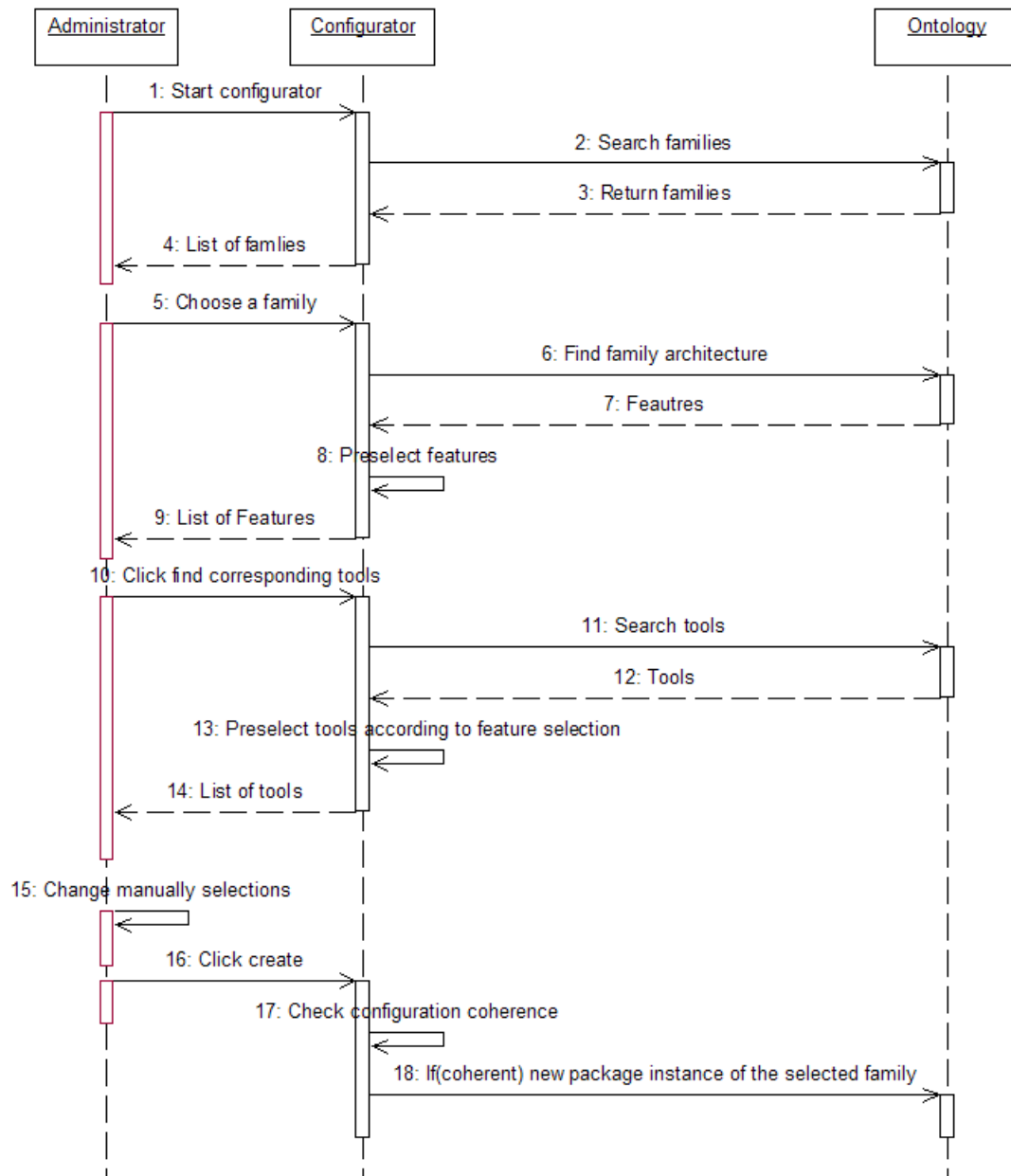


Figure C.2: Sequence diagram for creating a package

## Appendix D

---

# Guide to choose a Semantic Web Toolkit

---

This appendix presents some Semantic Web toolkits for the .NET platform and gives an overview about the features of each toolkit, the strength of the development effort and the toolkit's user community.

### D.1 Comparison Criteria

This chapter explains the comparison criteria.

#### D.1.1 License

License under which the toolkit is distributed.

#### D.1.2 API-Paradigm

This criteria describes which general API-Paradigms are offered for manipulating RDF data. The different paradigm are:

- Dataset- or Named Graphs-centric APIs allow you to represent and query RDF datasets.
- A model-centric API only allows you to load, save and delete whole RDF models. Through a statement-centric API, RDF data is manipulated as a set of RDF triples each consisting of a subject, predicate, and object. Usual methods are `model.addStatement()` or `find(S,P,O)`.

- A resource-centric API presents a RDF model as resources having properties. Usual methods are `resource.addProperty()` or `model.listResources()`. The ontology-centric view is an extension to the resource-centric-view by adding direct support for the kinds of objects expected to be in an ontology: classes (in a class hierarchy), properties (in a property hierarchy) and individuals. An ontology-centric API might offer methods to list its super- and sub-classes or instances of a class.

### D.1.3 Query-Languages

List of all RDF query languages supported by the toolkit. A comparison of different Semantic Web query languages is found here. Support for SPARQL means that the toolkit supports at least some features of the standard RDF query language currently developed by DAWG.

### D.1.4 Model Storage

List of storage alternatives for RDF data. Possible values are:

- Memory, if data is only stored in memory and has to be serialized to XML/RDF or N3 for persistence.
- DB, if the toolkit supports storing data in a standard relational or object-relational database.
- File, if the toolkit stores data in its own proprietary file format. A scalability report on triple store applications has been published by the MIT Simile Project. The SWAD-Europe has also published a report about Open Source RDF storage systems.

### D.1.5 Supported Databases

List of supported databases or name of the database abstraction layer used.  
**Supported Serialization Formats** Supported serialization formats for reading and writing RDF models. An overview about different RDF serialization formats is found here. Information about GRDDL is found here. A comparison how different toolkits handle the W3C RDF Test Cases is found here.

### D.1.6 Supported Serialization Formats

Supported serialization formats for reading and writing RDF models.

### D.1.7 Reasoning Support

Ontology-languages supported by the toolkit to infer implicit statements. A language is already listed if most of the inference rules of the language are supported



by the toolkit. A comparison how different toolkits handle the W3C OWL Test Cases is found here.

### **D.1.8 RDF Server**

Indicator if the toolkit contains some kind of RDF server, which allows models to be queried over the web using HTTP or SOAP. For more information about RDF servers and data access protocols see W3C DAWG work on protocols.

### **D.1.9 Other Features**

Additional features of the toolkit that aren't captured by other criteria.

## **D.2 Toolkit Comparison**

This section lists the toolkits for the .NET platform and makes a comparison of these toolkits according to the criteria defined in the previous chapter.

	Drive 3.3	EulerSharp 1.1.33	Spiral 0.34	SemWeb 0.751 0.751	Thea-VBA
License	LGPL	W3C License	MIT License	Creative Commons Attribution License	GNU GPL
API Paradigm	Statement centric	Model centric	Resource centric Ontology centric	Statement centric	Triple centric
Query Languages		N3QL		SPARQL , RSquery	
Model Storage	Memory	Memory	Memory , DB	Memory , DB	Memory
Supported Databases			MySQL	MySQL , PostgreSQL , SQLite	
Supported Serialization Formats	RDF/XML N3	N3	RDF/XML  N-Triples	RDF/XML  , N3 , Turtle , NTriples	RDF/XML
Reasoning Support		RDFS OWL	RDFS	RDFS	
RDF Server				SPARQL Protocol	
Other Features	RDF tool- bar for IE		Built-in support for FOAF , RSS and RdfCal	Supports quads	Native  MS-Excel interface

Table D.1: Toolkits comparison Table

### D.3 Conclusion

Regarding the table we chose SemWeb as the toolkit to use in the web site component (see section 5.3.1) to make the link with the ontology. Our choice has been done basing on the fact that SemWeb supports the SPARQL query language and also because it supports numerous databases and serialization formats.

## Appendix E

---

# Prototype user interface

---

Goal of this appendix is to give an overview of the user interface for each component of our solution.

### E.1 Web site component

This section presents some screen shots of the graphical user interface offered by the web site component. The user interface is the result of the collaboration with the sales people in DNV. Four screen shots are presented:

- Login page (see figure E.1).
- The main page (see figure E.2) of the web page where we can see the search frame on the left, the tabs on the top and the main frame where the packages are displayed.
- The search frame (see figure E.3).
- A pricing calculation. This screen shot is a zoom of the main frame of the figure E.2 after the pricing of one package.

### E.2 Package management tools

The package management tools support the creation of new packages using the package creation tool (also called configurator). The package management tools support the domain engineering using the protege editor. Consequently we will provide screen shots of both tool.

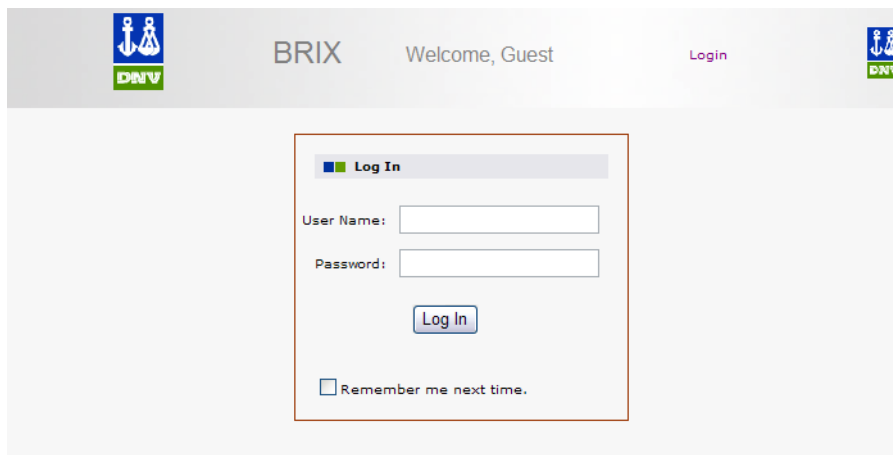


Figure E.1: Web site login page

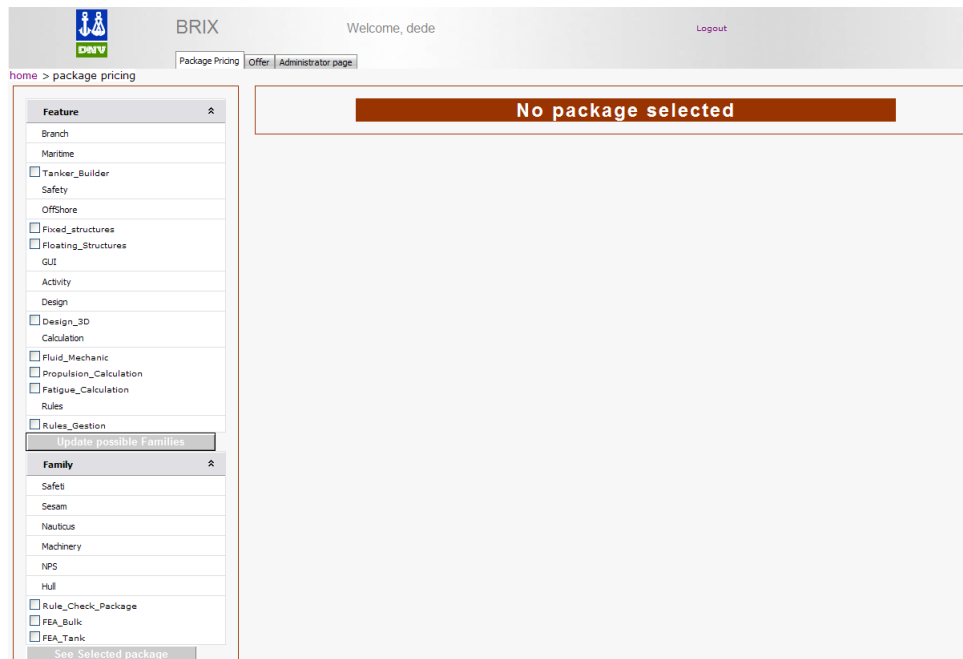


Figure E.2: Web site main page

## E.2.1 Package creation tool

This section presents some screen shots of the graphical user interface offered by the Package creation tool.

### **E.2.2 Ontology editor**

The Protg-OWL editor enables users to build ontologies for the Semantic Web, in particular in the W3C's Web Ontology Language (OWL). The editor offers numerous tabs (classes, properties, instance). This section present only one screen shot of the protege editor. More screen shoots are available on the protege web page <sup>1</sup>.

---

<sup>1</sup><http://protege.stanford.edu/overview/po-screenshots.html>

Feature	⤴
Branch	
Maritime	
<input type="checkbox"/> Tanker_Builder	
Safety	
OffShore	
<input type="checkbox"/> Fixed_structures	
<input type="checkbox"/> Floating_Structures	
GUI	
Activity	
Design	
<input type="checkbox"/> Design_3D	
Calculation	
<input type="checkbox"/> Fluid_Mechanic	
<input type="checkbox"/> Propulsion_Calculation	
<input type="checkbox"/> Fatigue_Calculation	
Rules	
<input checked="" type="checkbox"/> Rules_Gestion	
<b>Update possible Families</b>	
Family	⤴
Safeti	
Sesam	
Nauticus	
Machinery	
NPS	
Hull	
<input type="checkbox"/> Rule_Check_Package	
<input type="checkbox"/> FEA_Bulk	
<input type="checkbox"/> FEA_Tank	
<b>See Selected package</b>	

Figure E.3: Web site search frame

Package : FEA\_Bulk

Choose currency for the pricing : NOK Change currency

PRESELECTED TOOLS

Tool Name	Description	Price in NOK	Include in offer	Custom pricing options	Computed price
Xtract	Post-processing of FE results, includes animation.	120000	<input checked="" type="checkbox"/>	<span style="border: 1px solid gray; padding: 2px;">Define</span> <span style="border: 1px solid gray; padding: 2px; margin-left: 10px;">Show</span> <span style="border: 1px solid gray; padding: 2px; margin-left: 10px;">Delete</span>	118950
Seatra_Basic_Dynamics	Analysis tool.	100000	<input checked="" type="checkbox"/>	<span style="border: 1px solid gray; padding: 2px;">Define</span> <span style="border: 1px solid gray; padding: 2px; margin-left: 10px;">Show</span> <span style="border: 1px solid gray; padding: 2px; margin-left: 10px;">Delete</span>	99125
Genie_ext._CCPL	Code checking of plates (all codes).	0	<input checked="" type="checkbox"/>	<span style="border: 1px solid gray; padding: 2px;">Define</span> <span style="border: 1px solid gray; padding: 2px; margin-left: 10px;">Show</span> <span style="border: 1px solid gray; padding: 2px; margin-left: 10px;">Delete</span>	0
Genie_Basic	Basic equipments and concept results.	100000	<input checked="" type="checkbox"/>	<span style="border: 1px solid gray; padding: 2px;">Define</span> <span style="border: 1px solid gray; padding: 2px; margin-left: 10px;">Show</span> <span style="border: 1px solid gray; padding: 2px; margin-left: 10px;">Delete</span>	99125
Genie_Ship_Rules	No description.	0	<input checked="" type="checkbox"/>	<span style="border: 1px solid gray; padding: 2px;">Define</span> <span style="border: 1px solid gray; padding: 2px; margin-left: 10px;">Show</span> <span style="border: 1px solid gray; padding: 2px; margin-left: 10px;">Delete</span>	0
Genie_ext._CGEO	Curved geometry.	50000	<input type="checkbox"/>	<span style="border: 1px solid gray; padding: 2px;">Define</span> <span style="border: 1px solid gray; padding: 2px; margin-left: 10px;">Show</span> <span style="border: 1px solid gray; padding: 2px; margin-left: 10px;">Delete</span>	
<b>Total Price</b>		<b>317200</b>			

OPTIONAL TOOLS

Tool Name	Description	Price in NOK	Included in offer	Custom pricing options	Computed price
Cutras	No description.	0	<input type="checkbox"/>	<span style="border: 1px solid gray; padding: 2px;">Define</span> <span style="border: 1px solid gray; padding: 2px; margin-left: 10px;">Show</span> <span style="border: 1px solid gray; padding: 2px; margin-left: 10px;">Delete</span>	
Presel	Superelement modelling - gives also access to superelement analysis.	75000	<input checked="" type="checkbox"/>	<span style="border: 1px solid gray; padding: 2px;">Define</span> <span style="border: 1px solid gray; padding: 2px; margin-left: 10px;">Show</span> <span style="border: 1px solid gray; padding: 2px; margin-left: 10px;">Delete</span>	74343,75
Submod	Sub-modelling.	60000	<input type="checkbox"/>	<span style="border: 1px solid gray; padding: 2px;">Define</span> <span style="border: 1px solid gray; padding: 2px; margin-left: 10px;">Show</span> <span style="border: 1px solid gray; padding: 2px; margin-left: 10px;">Delete</span>	
<b>Total Price</b>		<b>74343,75</b>			

Total price = 391543,75

Pricing options

Number of users: 5
 Offer type: base\_purchase
 Network installation type: WAN
 License type: network
 Discount: 0,5

Figure E.4: Screen shot of a pricing calculation

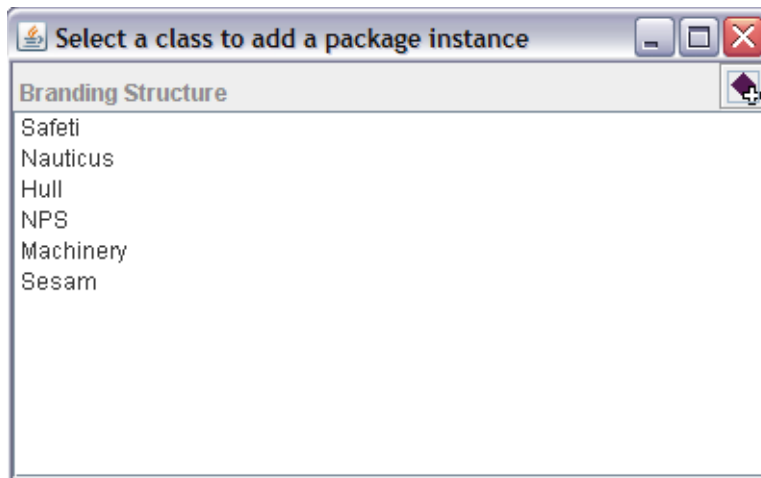


Figure E.5: Family selection

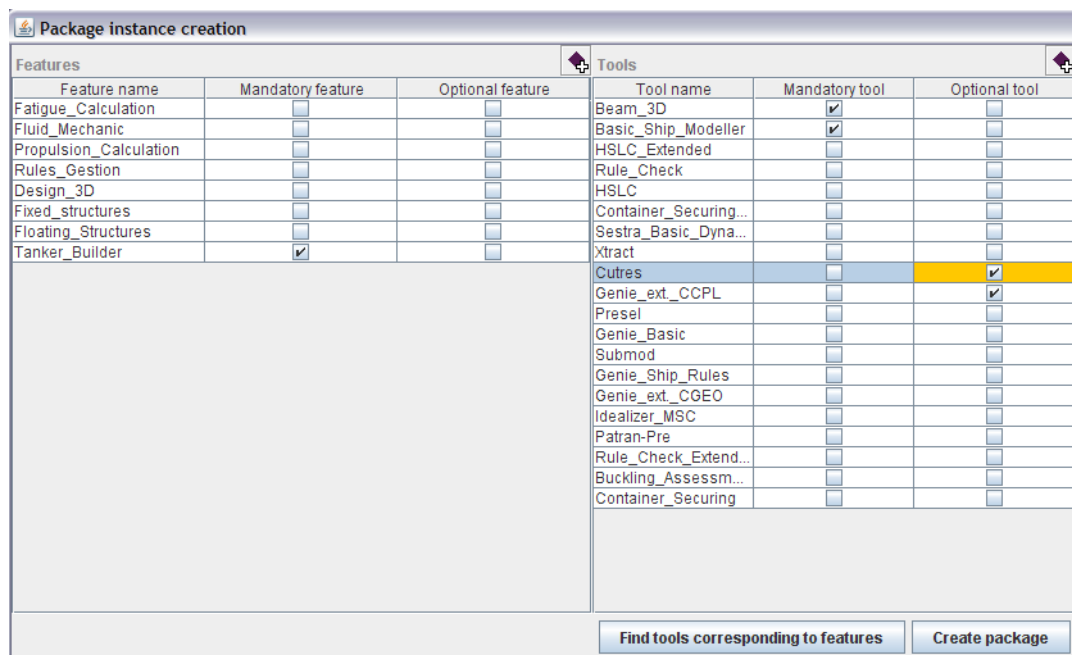


Figure E.6: Package creation



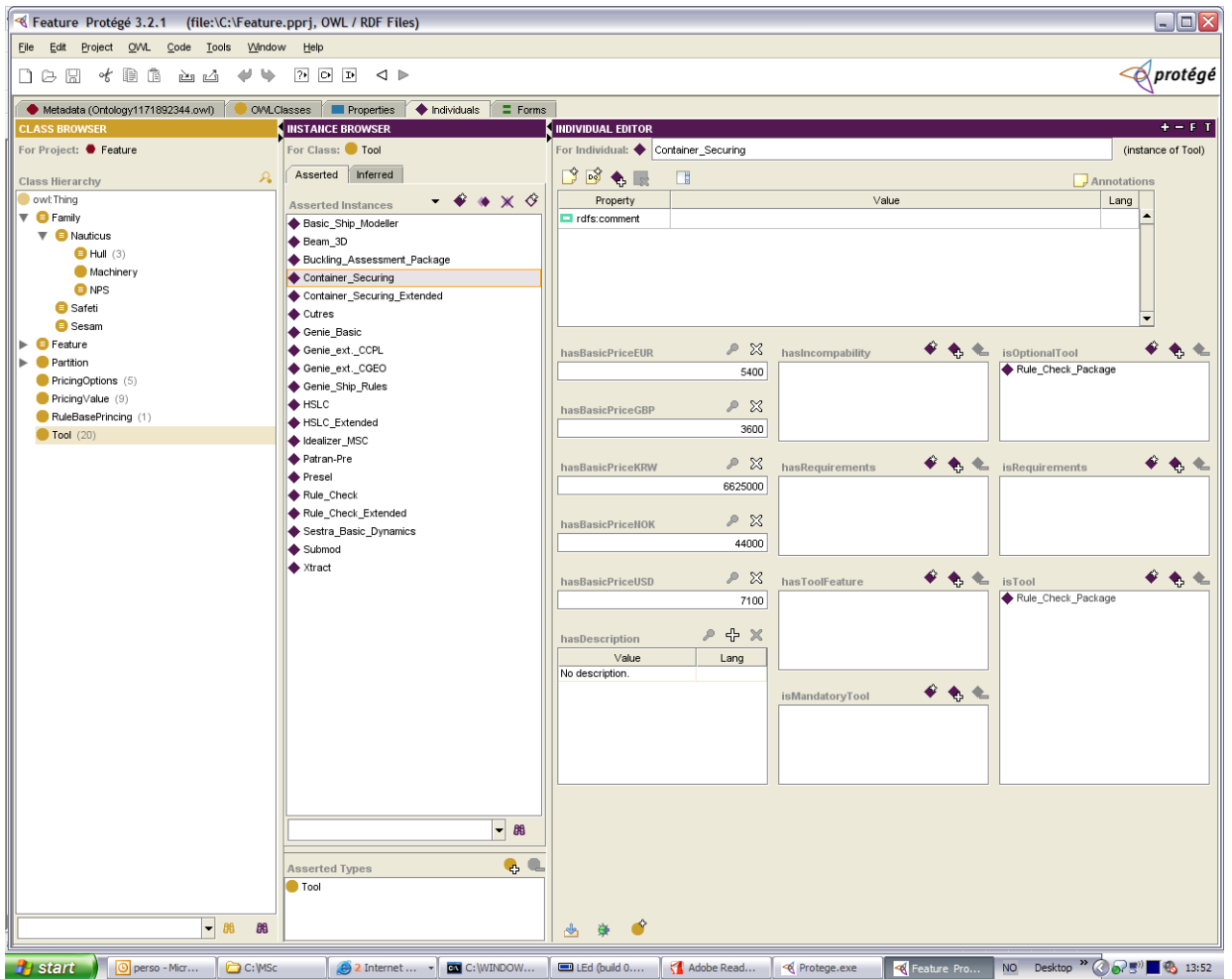


Figure E.7: Ontology of Product Family in Protege editor



## Appendix F

---

# Tests

---

Goal of this appendix is to present the detailed tests done for the evaluation of the final prototype. Each test is described using the template bellow.

- *Information*
  - Name of test case that allows to quickly understand test case purpose and scope.
  - Requirement which is covered by the test case.
  - Purpose contains short description of test purpose, what functionality it checks.
- *Test case activity*
  - Initialization describes actions, which must be performed before test case execution is started. For example, we should open some file.
  - Actions step by step to be done to complete test.
  - Input data description
- *Results*
  - Expected results contains description of what tester should see after all test steps has been completed.
  - Actual results contains a brief description of what the tester saw after the test steps has been completed.
  - Remarks.

## F.1 Package selection

- *Information*
  - Name. Package selection test.
  - The test case cover the requirements "The system has to help the salesperson to select the offer corresponding to the client needs".
  - Purpose. Goal of the test is to figure out if the features search provide by the web site offer a good help to the sales people. The users can search for packages using the left frame of the default page of the web site ( see figure E.3).
- *Test case activity*
  - Initialization. We have to create some packages that are define by features in order to provided the information for the search.
  - Actions step by step to be done to complete test.
    1. Open the web site.
    2. Login as a sales person.
    3. Select a set of features.
    4. Click on the button that update the list of packages that correspond to the set of features selected.
    5. Select a package and display it.
- *Results*
  - The only package corresponding to this feature should be Rule Check Package.
  - Actual results are the same as the expected results.
  - Remarks. The users affirmed that the system helped them to find a convenient package. Meanwhile some users think that it should be good to update the tree of features to reflect the current features selection (some features may exclude or require other features). Another remark was that in lot of case the sales people know at least partially what package they are looking for (i.e. from which family the package is a member) so it should be possible to select a family and then update the three of features according to the family architecture.

## F.2 Price calculation

### F.2.1 Normal pricing calculation

- *Information*
  - Name. Normal price calculation test.
  - The test validates three requirements:
    - \* The system has to integrate the rules described in the pricing scheme
    - \* The system has to create the pricing offers to customers
    - \* The system has to use the Brix technology
  - Purpose. Goal of the test is to figure out if the rules defined using the rules engine give the right result according to the pricing options. This test tests the rules and how the web site understands the information about the pricing options that is encoded in the ontology.
- *Test case activity*
  - Initialization. In order to start the test the package *FEA Bulk* has to be selected and displayed.
  - Actions step by step to be done to complete test.
    1. Select the optional tool *Presel* and remove the preselected tool *Genie ext CGEO*.
    2. Enter the following pricing options (number of users=5, offer type= lease purchase, installation type = WAN, License type= network, discount = 0,5). The pricing options are visible at the bottom of the screen shot [E.4](#).
    3. Click on the button to compute the prices of the tools within the packages.
- *Results*
  - Expected results:
    - \* Only the price of tools include in the offer have to be computed.
    - \* The prices have to be right according to the rules.
  - Actual results are the same as the expected results.
  - Remarks. Performing this test the sales people concluded that the results are correct but it has been observed that the response time to give a price can be quite long.

## F.2.2 Web site adaption to a change in pricing scheme

- *Information*
  - Name. Change pricing options test.
  - The test tests if the two needs listed bellow are supported:
    - \* The future system has to allow business knowledge to be maintained by domain experts without requiring programming skills.
    - \* A change in the family architecture or in the rules for the pricing of this family must be propagated in all applications using it.
  - Purpose. Goal of the test is to figure out if a change in the ontology concerning the pricing options will be taken in account by the web site component.
- *Test case activity*
  - Initialization. Open the ontology with ontology editor and select the select the tab instances.
  - Actions step by step to be done to complete test.
    1. Select the instance Rule check package and then find the rule base attached to this package.
    2. Select the corresponding rule base
    3. Remove the value Discount from the list named *hasPricingOptions*.
    4. Select the instance License Type of the class *PricingOptions*.
    5. Change the value of *hasDisplayType* to *TextBox*.
    6. Open the web site.
    7. Login.
    8. Display the package rule check package.
    9. Do a normal price calculation.
- *Results*
  - Expected results:
    - \* The pricing option License type has to be enter in a text box.
    - \* There is no pricing option discount.
    - \* Only the price of tools include in the offer have to be computed.
    - \* The prices have to be right according to the rules.
  - Actual results are the same than the expected results.
  - Remarks. None.

### F.2.3 Customize pricing options test

- *Information*
  - Name. Customize pricing options test.
  - The test tests three requirements:
    - \* The system has to integrate the rules described in the pricing scheme.
    - \* The system has to create the pricing offers to customers.
    - \* The system has to use the Brix technology.
  - Purpose. Goal of the test is to figure out if it is possible to define custom pricing options for one tool.
- *Test case activity*
  - Initialization. In order to start the test the package FEA Bulk has to be selected and displayed.
  - Actions step by step to be done to complete test.
    1. Select the optional tool *Presel* and remove the preselected tool *Genie ext CGEO*.
    2. Enter the following pricing options (number of users=5, offer type= lease purchase, installation type = WAN, License type= network, discount = 0,5).
    3. Click on the button to define custom pricing options for the tool *Xtract*.
    4. Enter the options (number of users=1, offer type= purchase, installation type = LAN, License type= network, discount = 0).
    5. Click on the button to compute the prices of the tools within the packages.
- *Results*
  - Expected result: The price compute for the tool *xtract* has to be equal to the basic price of this tool.
  - Actual results are the same than the expected results.
  - Remarks. None.

## F.3 Package creation

### F.3.1 Consistency of a new package

- Information

- Name. Package consistency test.
- The test verifies if the system allows to produce consistent packages (i.e. the set of features that is supported by the tools within the composition of a package is compatible with the set of features that describe this package).
- Test case activity
  - Initialization.
  - Actions step by step to be done to complete test:
    1. Open the package management tool.
    2. Select the family of product *Nauticus* to create a new package.
    3. Select the feature *Rules gestion* as optional feature.
    4. Define the tool *HSLC* has mandatory for the new package.
    5. Select a rule based.
    6. Click on the button create new package.
- Results
  - Expected results:
    - \* The system has to preselect the tools *Rule check* as optional, *3D bean* and *basic ship modeler* as mandatory.
    - \* When the user will click on the button create new package the system has to explain that because the tool *HSLC* is mandatory tool for the new package then the package has to be described by the feature *fatigue calculation*.
  - Actual results. The system displays the good message. It is impossible to create a package which tools support more features than it has to describe itself (see figure F.1).
  - Remark. This test was a success because the system does not allows to create inconsistent packages. But some discussions happened in order to define if the set of feature describing the packages has to be included in the set of features supporting by the tools or not. This remark is equivalent to ask the question: Is a composition of tools superior in terms of features supported to the sum of the features supported by each tool?



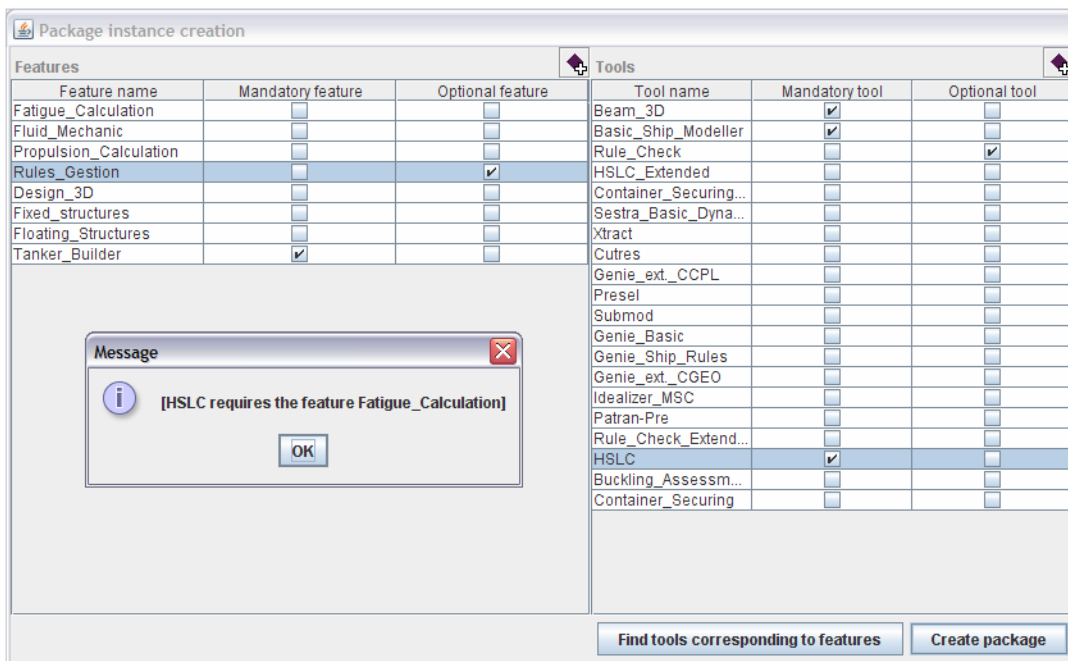


Figure F.1: Error when creating a new package

### F.3.2 Variability support during the configuration

- *Information*
  - Name. Product family variability in package creation test.
  - The test case verifies if the system supports the variability inherent to the family of products concept and if it supports the product engineering phase (see section 2.2.4).
  - Purpose. Goal of this test to figure out if the variability contained in the features (alternative, multiple or mandatory) is well supported by the package management tool.
- *Test case activity*
  - Initialization. None.
    1. Open the tool.
    2. Select the family named *Sesam*. This family is a family of products that support the offshore industry.
    3. Click on the button to create an new package in this family.
    4. Look the preselected features for the new packages.
- *Results*
  - The expected results are:
    - \* Only features defined in the family architecture have to be preselected.
    - \* The mandatory features have to be preselected.
    - \* No alternative features have to be selected at the same time.
  - Actual results:
    - \* The features preselected correspond to the family architecture.
    - \* The multiple features are all preselected without asking without asking to the user if he wants to include them in the new package.
    - \* The optional features are all preselected as optional for the new package without asking to the user if he wants to include them in the new package.
  - Remarks. The users have to know by advance the structure of the features in order to be able to modify the preselected set of features. it implies that the people that will create the architecture and the new packages have to be the same.

## F.4 Family creation

### F.4.1 Family creation test

- *Information*
  - Name. Family creation.
  - This test verifies if the system support the domain engineering and if the knowledge is easy to maintain.
  - Purpose. Goal of this test is to check if it is easy to create new family using the ontology editor. It verifies also that the inheritance between family architecture works properly. We would like to verify also that the web site is updated when a new family is created.
- *Test case activity*
  - Initialization. Open the ontology with protege.
  - Input data description
    1. Open the class tab in protege.
    2. Create a sub class to the class *Nauticus*.
    3. Enter a name to the class.
    4. Add the restriction "*hasMandatoryFeature some Rules*".
    5. Save the ontology.
    6. Create an instance of the Class using the package creation tool.
    7. Login to the web site.
    8. Verify that the new family is represented in the family three in the search frame (see screen shot E.3).
- *Results*
  - Expected results. We expect that the new family inherit the restriction from *Nauticus*, that the rules feature is preselected when using the package creation tool and that the web site reflect the changes made to the ontology.
  - Actual results are concordant with the expected results.
  - Remarks. The administrator expressed that the concepts of the ontology language used (i.e. OWL) are not easy to understand.

### F.4.2 Feature type support

This test needed a pre formation to explain the users the patterns to create features (see section 5.3.2).

- *Information*
  - Name. Feature creation.
  - This test verifies if the system support the creation of the different types of features.
  - Purpose of this test is to collect the remarks of the users.
- *Test case activity*
  - Initialization. Open the ontology with protege.
  - Input data description
    1. Open the class tab in Protege.
    2. Create a new sub class to the class Family.
    3. Create a new mandatory feature for the new Family.
    4. Create a new optional features for the new Family.
    5. Create a new multiple features for the new Family.
- *Results*
  - Expected results. We expect the users to be able to create the new family.
  - Actual results it is possible to create all the features types but we observed that it was not easy for the users to understand the steps to add a feature to one family.
  - Remarks. The administrator expressed that the patterns to use for the different types of features are difficult to understand.

### F.4.3 Family architecture consistency

- *Information*
  - Name. Family consistency test.
  - This test verifies if the system enable to detect inconsistency in a family architecture.
- *Test case activity*
  - Initialization. Open the ontology with protege. Run the reasoner.
  - Input data description
    1. Open the class tab in protege.
    2. Select a class.
    3. Add inconsistent restrictions.
    4. Run the consistency.
- *Results*
  - Expected results. The consistency test has to detect the error.
  - Actual results it is possible to find that there is an error in the ontology.
  - Remarks. The users said that it is not easy to understand why a family is or is not consistent and where to find the consistency error.

#### F.4.4 Redundancy test

- *Information*
  - Name. Redundancy in family architecture.
  - This test verifies how the system helps to handle redundancy in feature model.
- *Test case activity*
  - Initialization. Open the ontology with protege. Run the reasoner.
  - Input data description
    1. Open the class tab in protege.
    2. Create a sub class of family.
    3. Add the same restrictions than the *Nauticus* class.
    4. Add others restrictions.
    5. Run the consistency.
    6. Click on Classify taxonomy.
- *Results*
  - Expected results. The computed taxonomy has to classify the new class under the class *Nauticus*. In other words the reasoner has to infer the right parent for the new family in order to inherit part of its architecture.
  - Actual results correspond to the expected ones.
  - Remarks. The system helps to find redundancy but it does not correct it. The user has to correct the problem himself.

---

# List of Symbols and Abbreviations

---

Abbreviation	Description	Definition
FODA	Feature-Oriented Domain Analysis	page 11
DNV	Det Norske Veritas	page 2
XML	eXtensible Markup Language	page 24
RDF	Resource Description Framework	page 24
URI	Uniform Resource Identifier	page 24
OWL	Ontology Web Language	page 29
DAML	DARPA Agent Markup Language	page 28
OIL	Ontology Inference Layer or Ontology Interchange Language	page 28
DL	Description Logic	page 29
AI	Artificial Intelligence	page 32
SWESE	Semantic Web Enabled Software Engineering	page 35
MDA	Model Driven Architecture	page 36
UML	Unified Modeling Language	page 36
W3C	World Wide Web Consortium	page 34
UC	Use Cases	page 40
API	Application Programming Interface	page 50

---

# List of Figures

---

2.1	DNV's layered architecture . . . . .	6
2.2	Brix framework organization . . . . .	7
2.3	Business process for BRIX Workflow . . . . .	8
2.4	Screen shot of BRIX Rule Editor . . . . .	8
2.5	Dimension of variability in Product Family . . . . .	10
2.6	Example of feature model with FODA . . . . .	12
4.1	Semantic Web layer cake . . . . .	24
4.2	Ontology languages evolution. . . . .	30
4.3	Forward chaining process (source [14]) . . . . .	33
4.4	Bacward chaining process (source [14]) . . . . .	37
5.1	Use cases for a sales person . . . . .	40
5.2	Use cases for an administrator . . . . .	41
5.3	Application's components use . . . . .	43
5.4	Pricing model . . . . .	44
5.5	Configurator model . . . . .	45
5.6	Package creation process . . . . .	47
5.7	Example of alternative feature. . . . .	52
6.1	Spiral development method . . . . .	60
C.1	Sequence diagram for pricing a package . . . . .	95
C.2	Sequence diagram for creating a package . . . . .	96
E.1	Web site login page . . . . .	102
E.2	Web site main page . . . . .	102
E.3	Web site search frame . . . . .	104
E.4	Screen shot of a pricing calculation . . . . .	105
E.5	Family selection . . . . .	106
E.6	Package creation . . . . .	106



E.7	Ontology of Product Family in Protege editor . . . . .	107
F.1	Error when creating a new package . . . . .	115

---

# List of Tables

---

2.1	Requirements and needs Table . . . . .	16
5.1	Administrator use cases related to requirements and needs . . . . .	42
5.2	Factor A table . . . . .	53
5.3	Factor B for network license . . . . .	54
5.4	Factor B for non network license . . . . .	54
5.5	Factor C table . . . . .	54
6.1	Test template . . . . .	61
6.2	Package selection test . . . . .	62
6.3	Normal pricing calculation test . . . . .	63
6.4	Web site adaption test . . . . .	63
6.5	Customize pricing options test . . . . .	64
6.6	Package consistency test . . . . .	64
6.7	Variability support during the configuration . . . . .	65
6.8	Family creation test . . . . .	66
6.9	Family creation test . . . . .	66
6.10	Family consistency test . . . . .	67
6.11	Redundancy test . . . . .	67
6.12	Fulfilled requirements table . . . . .	68
D.1	Toolkits comparison Table . . . . .	100