Méthode d'évolution de modèles produits dans les sytèmes PLM

Seyed Hamedreza Izadpanah

UNIVERSITÉ DE GRENOBLE

THÈSE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ DE GRENOBLE

Spécialité : **Génie Industriel**

Arrêté ministériel : 7 août 2006

Présentée par

« Seyed Hamedreza IZADPANAH »

Thèse dirigée par **« Michel TOLLENAERE »** et
codirigée par **« Lilia GZARA »**

Préparée au sein du **Laboratoire G-SCOP**
dans **l'École Doctorale IMEP2**

Méthodologie d'évolution de modèles produit dans les systèmes PLM

Date de la soutenance : **28 septembre 2011**,
devant le jury composé de :

Monsieur Laurent TABOUROT
Professeur à l'Université de Savoie, Examinateur
Monsieur Christophe MERLO
Maître de conférences HDR à l'ESTIA de Bayonne, Rapporteur
Monsieur Samuel GOMES
Professeur à l'UTBM, Rapporteur
Monsieur Philippe PERNELLE
Maître de Conférence à l'Université Lyon 1, Examinateur
Monsieur Michel TOLLENAERE
Professeur à Grenoble-INP, Directeur de thèse
Madame, Lilia GZARA
Maître de Conférence à Grenoble-INP, Co-encadrante

Insanity means doing the same thing over and over again and expecting different results.

Albert Einstein

Remerciements

Un travail de thèse est le résultat d'une forte collaboration. Je tiens à remercier ici toutes les personnes m'ayant aidé tout au long de mon parcours académique.

J'exprime mes profonds remerciements à mes encadrants, madame la docteur Lilia Gzara et le directeur de thèse, le professeur Michel Tollenaere, pour l'aide compétente qu'ils m'ont apportée, pour leur patience et leur encouragement.

Je remercie également Monsieur le professeur Yannick Frien, le directeur du laboratoire G-SCOP pour son aide et son soutien

Je remercie en premier lieu les membres du Jury pour avoir accepté de faire partie du Jury de cette thèse ainsi que pour le diplôme qu'ils m'ont accordé.

J'exprime aussi ma gratitude à tous mes amis et camarades au sein du laboratoire GSCOP pour créer cette superbe ambiance. Je tiens plus particulièrement à remercier mes collègues du bureau, Charlotte, Sandra, Valérie, Safa et Helene, elles sont toutes formidables.

Enfin, je remercie ma famille et mes amis pour l'aide et le soutien qu'ils m'ont apportés au cours de ces années de thèse.

Table of Content

Table of Figures

Table of Tables

Abbreviations

ATL	Atlas Transformation Language
BOM	Bill of Materials
CAD	Computer Aided Design
(CAO)	(Conception Assistée par l'Ordinateur)
DSM	Dependency Structure Matrix
DSML	Domain Specific Modeling Language
EMF	Eclipse Modeling Framework
MDA	Model Driven Architecture
MDE	Model Driven Engineering
(IDM)	(Ingénierie dirigée par les modèles)
MM	Meta-Model
MOF	Meta-Object Facilities
MT	Model Transformation
PDM	Product Data Management
PIM	Platform Independent Model
PLM	Product Lifecycle Management
PSM	Platform Specific Model
UML	Unified Modeling Language
XMI	Extensible Modeling Interchange
XML	Extensible Modeling Language

Chapter 1 Introduction and Context

PLM system has a very important role in whole enterprise information system. This chapter tries to explain this importance by presenting its mission in the roadmap of enterprise's strategy. Product model's position in the whole PLM system is the other subject to be discussed in this chapter. We aim to show the importance of product model evolution in PLM systems within industrial information management systems. This chapter will finish with a brief representation of scientific problematic as well as its context.

1. PLM Systems

Product information management is a huge challenge for industries nowadays. Connecting to a secure stream of the exact part of data in the proper time and appropriate place is essential and vital for an enterprise in this globalized business world. In order to fulfil this necessity, enterprises employ information systems like PLM (Product Lifecycle Management), which help them to organise and share their technical data (Terzi S. 2007)

A commonly accepted definition of product life-cycle management, proposed by CMIData (CMIData 2003) is "a strategic business approach that applies a consistent set of business solution in support of the collaborative creation, management, dissemination, and use of product definition information across the extended enterprise from concept to end of life-integrating people, processes, business, systems, and information". Therefore, PLM systems not only offer a document management system and a collaborative design platform, but also present a strategic approach through design and manufacturing until marketing and EOL (End Of Life) stages in order to retrieve information and improve business activities. So PLM tools manage product's information across its complete lifecycle from early phase of design until recycle phase, and allow the exchange of related data between these phases. Figure 1 shows the PLM implication in different phases of product life, extracted from (Terzi S. 2007)

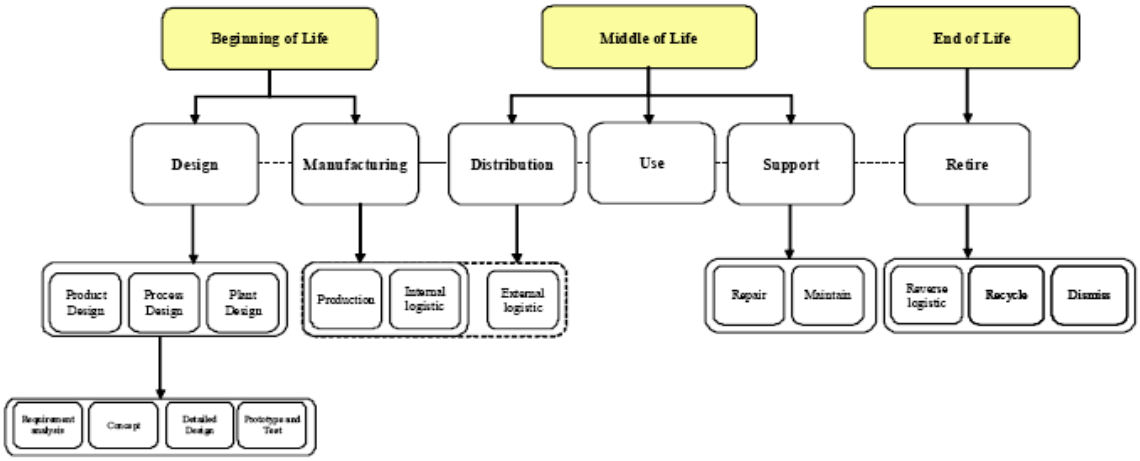


Figure 1-1, Product Lifecycle , (Terzi S. 2007)

Information systems that manage product data were limited to product design and development phases. Product Data Management (PDM) tools deal with product information structure and concepts during the design process. The objective of PLM systems is to break this limitation by managing product information within its lifecycle. However, nowadays PLM systems are yet focused on design phase of product lifecycle, which may be due to their age. By the time, the implication of PLM systems during the other phases may be done. It is undeniable that the huge actual tendency to PLM systems comes from their capacity in information management during design phase. In any case, PLM tools proposed by editors are claimed to be able to manage product during its whole lifecycle, which needs to be proved by several industrial cases.

1.1. Functionalities of PLM systems

The most important functionalities required from PLM systems in industries are:

- Organising, classifying, archiving and creating a reliable referential system of product information, product breakdown structure, and all technical data related to product. It should be capable to manage the product structure and its multiple views (C. Sung & S. Park 2007)
- Information sharing and diffusion between actors or stakeholders who use different levels of details or representation of product information, work with various types of computer systems and software, are located in different sites, during all phases of product design, production, after sale etc. (G. Thimm 2005). It means that PLM system assures the presence of appropriate information in requested time for proper actor.
- Globalization: Worldwide production demonstrates the necessity of using different types of distant collaborative design and manufacturing, etc. systems within the enterprise as well as with its partners in order to share necessary information in real time. Also globalized business situation, increases the competition between different enterprises, which forces them to improve their reactivity to market change (and various other vagaries) and encourage them to use marketing or end-of-life related information of product to improve their product design or fabrication (Golovatchev & Budde 2007).

Moreover, product innovation activities accompanied by strong concurrence, push enterprises to profit from PLM systems in order to set up collaborative frameworks, which help them to work in teams, design in real-time and reduce their costs (Schuh et al. 2008)(Eynard et al. 2006).

- Security of information: in globalized prospect of product design, precious information circulates the globe. This information should be accessible for everybody needs it, but the sharing system must guaranty the security.
- Maintaining the coherence of product information. The product information organisation changes during its life cycle. PLM system dues to manage this organisation changing. (Terzi S. 2007). Moreover, PLM system can assist to manage the product information exchange between different information systems like ERP, etc. within enterprise.
- Information traceability: With PLM systems, information of a part or a product can be followed and traced, during its lifecycle. This traceability contains information of the part, modifications done through time, products whose it belongs to and users created or changed this piece. These data may be crucial throughout the time. For example, a product component, which is a source of problems on the last phases of product life can be found and redesigned in the early phases of product development.
- Workflow and process standardisation: In each enterprise, there are lots of processes for treatment of product information during different phases of product lifecycle. PLM tools formalise, manage, keep trace and execute these processes by workflow functions. This facilitates the process tracking by its regularisation and autoimmunization (Golovatchev & Budde 2007).
- PLM systems help to integrate functionalities done by other design and engineering systems, such as visualisation, communication, calculation, and task execution. (A. A. Yassine et al. 2004)
- PLM systems can be used to solve the design problems related to environmental challenges. The capacity to manage the EOL information and integrate this information within the first phases of design is a good solution for sustainable product development. With this tool, designers can be informed about the environmental and ecological aspects of the product and its components in different phases of product lifecycle, and improve their design process by considering these demands.

1.2. PLM and its around

Several information systems handle data management in an enterprise, such as ERP, PLM, CIM, etc. They manage different information relating to different phases of product development, or different functionalities. Coherence of existing information in these systems is a challenge for industry. (Svensson & Malmqvist 2002) stated that in an efficient product information management system, ERP, PLM and other similar systems, which offer different product structure, should be adopted to each other. The challenge has been encountered by considering the different viewpoint of product structure related to different domains of activities of these systems. (Schuh et al. 2008) claimed that integration of Computer integrated

manufacturing core fundaments in the PLM system can improve the process management in enterprise.

This integration or making coherence, which is the subject of lots of researches, needs some modification in information systems to be integrated. Eventual evolution of PLM may be conceivable.

1.3. PLM Tools Architecture

Architecture of an information system is an important subject. Analyzing the impacts of architecture on the process of evolution as well as the influence of evolution on the IS architecture helps to 1) design IS architecture more optimally in order to make it more adaptable for eventual evolutions during its lifecycle and 2) improve the pertinence of evolution process since evolution process phases may be determined by the section of IS to be evolved. (Nieva T. 2001) defines the architecture of IS as a concept that originally comes from the creation of building structures. This structure is enriched by determination of its components and its relationships.

Before entering to debate, it should be noticed that, in academic research, there are two approaches for architectural analysis of IS. First approach focalises more on informatics aspect of tools used to implement an IS. In this approach, IS deduces to an informatics tool and then different parts of this tool are studied. Databases, interfaces, modelling infrastructures, processes workflow diagrams are some different parts of IS with software decomposition viewpoint. Research on IS architecture focus more on this approach. The other approach deals with functionalities proposed by a tool. In this approach, tool is divided into some business activity fragments, like different services, processes and portfolios. Industrial information's systems like PLM are studied more with functionality-based viewpoint.

(Hong-Bae Jun et al. 2006) defines architecture of a PLM system on three different viewpoints: Business architecture, Software architecture and Hardware architecture. Their business architecture is based on different BOL, MOL and EOL modelling. (Beginning of life, Middle of life, End of life). Their approach is developed for closed-loop PLM, in which an embedded chip in the product contains its information. Proposed software architecture in (Hong-Bae Jun et al. 2006) for their closed-loop PLM is illustrated in figure 2:

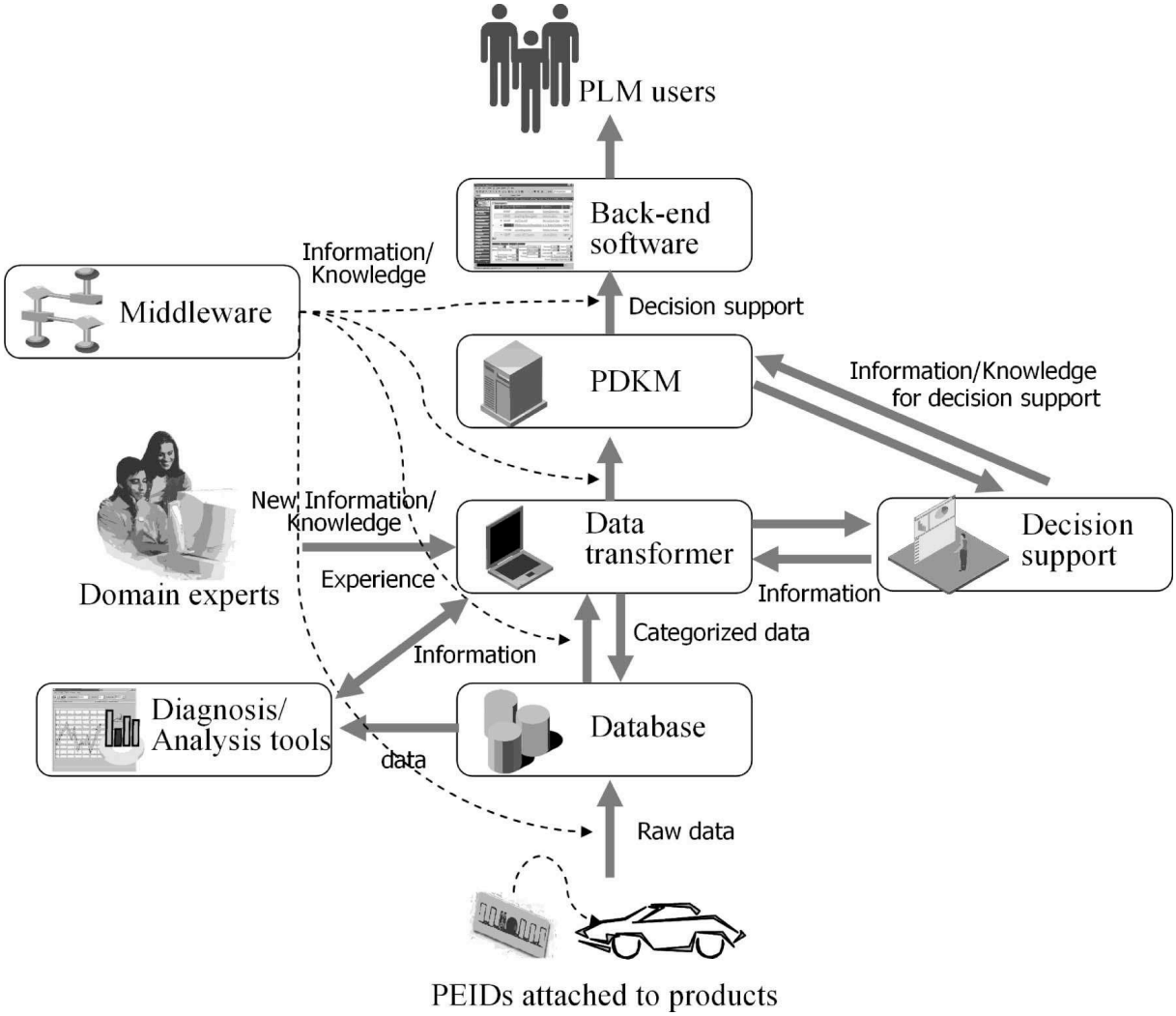


Figure 1-2, PLM software architecture presented in (Hong-Bae Jun et al. 2006)

(Golovatchev & Budde 2007) in their research on the integrated approach to PLM, explain IT architecture of PLM as one of the four principal components of integrated PLM-approach. This structure must maintain 3 axes of PLM process: 1) decision support, 2) operational support and 3) integration of supplemental business applications. These components consist of different modules of PLM functionalities; already existing separately in enterprise. Therefore, they have proposed a method of reuse of existing IT components (modules) for their integrated PLM. These modules are presented in figure 3.

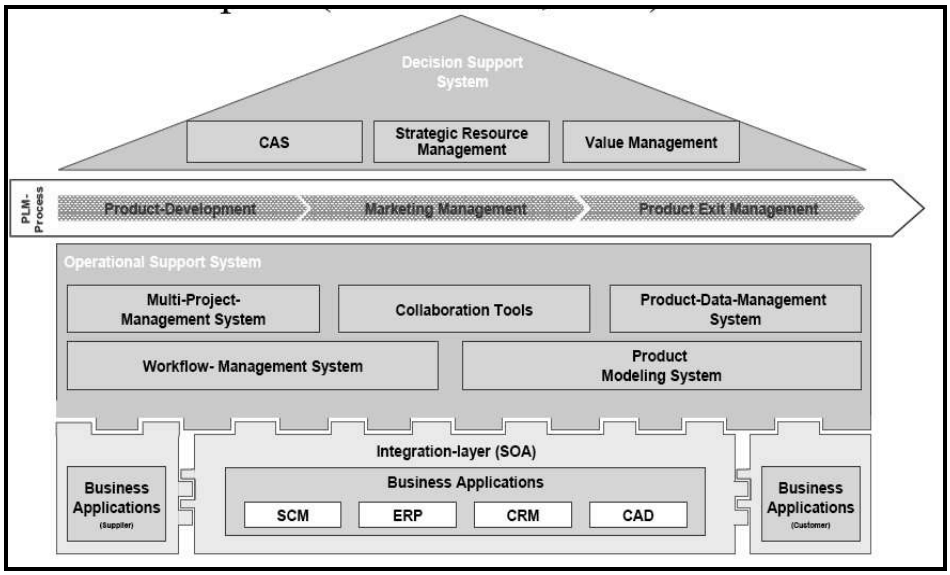


Figure 1-3, IT architecture of (Golovatchev & Budde 2007)

The approach of PLM tool architecture, used in this thesis, is based on its informatics and business characteristics. A PLM tool, which is supposed to be integrated in enterprise, consists of two separated sections: (1) Core section, which is the set of principle functionalities exiting in the PLM tool sold to the enterprise (functionalities like modeling framework, databases, workflow engines as well as interfaces). (2) Customization section, which is added in order to integrate enterprise-required functionalities and adapt the PLM to enterprise specificities and requisites of future deployment. For example, product models, as well as product meta-models should be developed and built, based on core framework facilities. Therefore, the core section is a PLM offer of developers and customization layer is the consequence of deployment process.

The advantages of choosing this approach of PLM architecture are:

- Separating commercial tool from integration-adding section may be useful while evolution caused and consequences are studied. Because the causes related to the tool modification or replacement have different consequences than customization layer changing. Furthermore, the other modularization of principle functionalities like product and process modeling will be useful for classification and analysis of PLM evolution.
- It is based on integration process of PLM.

In the 3rd chapter, the new representation of this architecture, which is more adaptable to our solutions' methodology (MDE), will be presented.

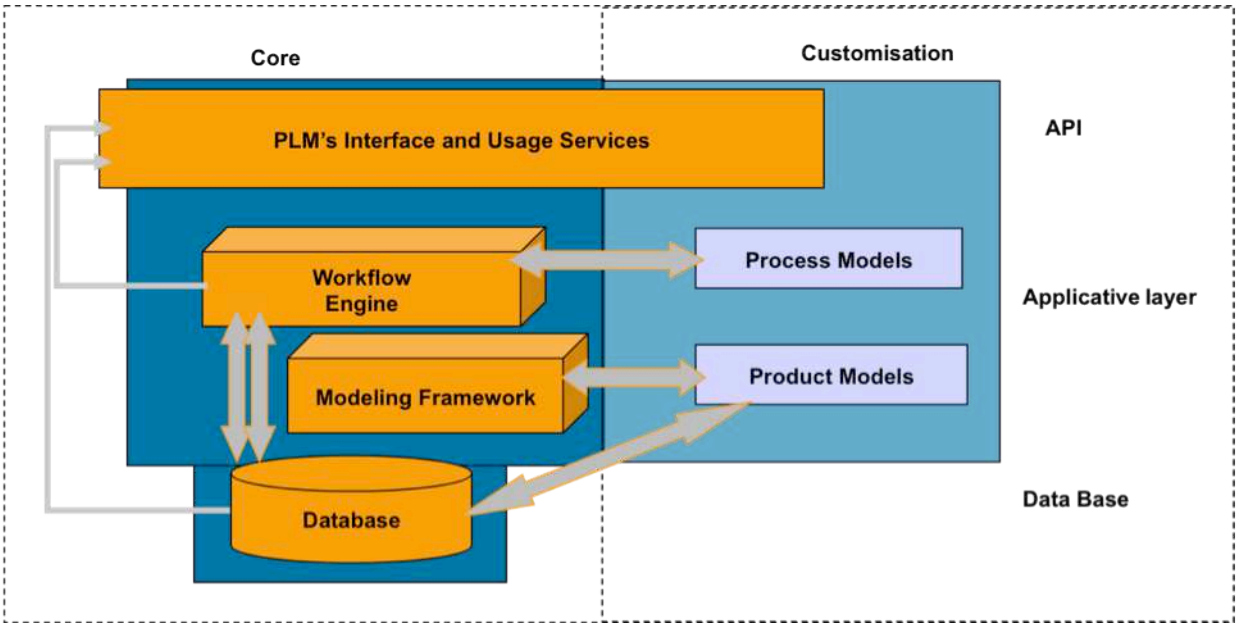


Figure 1-4, PLM architecture

1.4. Product Model, Definition and its role in PLM systems

Product model, product structure, product configuration, product breakdown and Bill of Materials are the most common terms used to explain product constitution. Different in their definitions, they are commonly employed for diverse functionalities in product lifecycle. For example, BOM shows the product's physical components. So it's more effective for assembly or manufacturing purposes. Regardless to exact definition of "Product model", it's the most principle structure to manage product information in PLM systems, since all information and technical data must be organized and stored within this structure. All eventual evolution or changing of PLM may influence on this structure. In other words, product model can be considered as the core of information organization in PLM system, which has a strong association with evolution subjects. This will be elucidated in coming sections.

Based on the definition of Zeng et Jin (Zeng & Jin 2007), "Product Model" is an abstract of a product series, which represents product. In the classical tree, components form the nodes. Product model in this form, which contains only physical product sub-components, represents assembly decomposition. In (Svensson & Malmqvist 2002), they clarified the different types of product structure of a single product, based on business viewpoint, such as design or manufacturing and counted the different usage of these viewpoints. In their research, they claimed that one of the challenges of collaborative frameworks is the lack of coherence between these product structure viewpoints of a single product.

Garwood defined BOM as "items or raw materials that go in to product" (D. Garwood 1988). Essential tree aspects of BOM, regarding to Jiao (Jiao et al. 2000), are items, relationships and employment. Items are the product subparts, which may be purchased parts, intermediate parts or even finished product. Relationship defines the parent-component; and finally employment explains the different viewpoints of a singular product structure via different perspective business functions. BOM is commonly used for production management.(Jiao et al. 2000)

2. PLM Evolution, Industrial context

PLM systems manage and organize all of product-relating information within enterprise. Moreover, all workflows which should be followed in order to receive, collect, stock, modify, approve, and share this information are organized by PLM. This structure of information and workflow undergoes lots of modification or changing due to all types of evolution, which occur in the product configuration, organizational structure of enterprise, or informatics modification of the system.

Evolution in product design and fabrication occurs frequently in a complex and multi-actors context. Changing of product definition in order to extend it, modification of product structure, or shifting modeling framework, which are probable, can originate PLM evolution. Adaptation of PLM to other information systems of enterprise or, systems of partners, customers, or suppliers may obligate the need to evolve PLM. This might be happen while the enterprise wants to joint or merge to other enterprise.

2.1.1. Problematic, Model Evolution

As discussed PLM systems are one strategic components of information management in enterprises. Their role in creating and maintaining enterprise's collaborative framework is fundamental and crucial. Furthermore, They are strongly connected to almost all business activities of enterprise. It means that product design, manufacturing, marketing, maintenance, recycling, which contain almost all activities of the enterprise, are dependent to PLM system. Therefore, a PLM system, which is adapted accurately to enterprise requirements and specifications, and additionally is flexible and evolvable with different possible types of changing or modification, is vital.

In the other hand, the risk of evolution in PLM system is high. It is because of this strong dependence of PLM system and all activities of enterprise. Changing enterprise activities, modification of workflows, product evolution, system replacement and other reasons argued in the last section, which are enough probable, may influence on organization of information and obligate the PLM evolution.

Product model, as elaborated in section 1.3, is the most important information structure in PLM systems. Therefore, it may be the first PLM component that suffers from these probable evolutions. Moreover, any evolution of product model can affect other components of PLM. In this approach our problematic returns with the evolution of product model within PLM system.

Our approach in order to analyze this problematic consists of two wings:

- Analyze the cause of evolution of PLM, with business viewpoint. These causes are industrial problematic, which force the enterprise to modify their PLM. They are discussed in the next chapter.
- Analyze the component of PLM system that is influenced by evolution. This approach helps us to find the appropriate solution for each accurate situation of evolution. This analyze is based on the architectural study done in section 1.2, in which the different informatics and business components of PLM system are identified.

The research question supposed to be answered during this research work may be formulated:

> Which procedure should be followed for evolving a product model from one condition to another, within PLM systems, when the evolution originates from an external business change?

To clarify this problematic, it should be noticed that in our problematic changing condition refers to all types of modification can influence the product model, such as product redefinition, reorganization of enterprise, or system replacement. These types of source of evolution will be discussed in the next chapter.

The associated questions to the principal problematic, which may be useful to be analyzed preliminarily are:

- What are the different causes of product model evolution?
- Which component of PLM system may be undergone the consequences of each type of evolution?
- How to manage evolution?

These questions will be discussed in the next chapters.

3. Conclusion

In this chapter the global context of PLM systems and the importance of analyzing its evolution was discussed. The different functionalities of PLM systems were identified. These functionalities may accentuate the evolution problematic, because they make PLM system as a strategic tool in enterprise, which should be properly adapted to enterprise specifications.

Then PLM architecture was introduced. This architecture may help the enterprise to 1) improve the flexibility of its PLM for probable evolution in the pre-installation of system, 2) manage the evolution by identifying which component of system will be evolved and this evolution can affect which other components.

Next topic treated in this chapter is product model, one of the components of architectural structure of PLM systems. Product model has been chosen for this research because it is an important organization of data in PLM systems. PLM is due to manage product information. With this perspective, we assume that analyzing product model evolution within PLM may be useful to studying other type of evolution.

The importance and high probability of PLM evolution were discussed in the end of this chapter. PLM is relatively a new component of information managing framework of enterprise, comparing to other tools like ERP. These components like ERP or CMI have been employed more previously, so 1) they are now more optimal and adapted to requirement of enterprise, 2) problematic of evolution may be more treated in their context. PLM evolution may be initiated

by several reasons. These reasons were discussed in this chapter and will be detailed in the following one.

4. Map of this thesis

1st part of this thesis is about the problem definition. 2nd chapter introduces a detailed analysis of the problematic of this research. The different type of product model evolution, modification, changing and reformulation will be discussed. 3rd chapter devotes to explain our methodology of solution and approach that was chosen to propose the solution mechanism.

2nd part is about the solution. It contains the proposed mechanism to enable evolution as well as an industrial example, which is treated by the proposed method. Industrial example is evolution of product model from a system that is only capable to deal with specific product model to one, which manages the genericity of product family.

Chapter 2 Product model evolution within PLM systems, Problematic

1. Beginning discussion

Various components of PLM systems may undergo an evolution. Workflows defined in PLM systems, organizational parts (users, groups, authorizations…) and the system interface are supposed to evolve. It is important to identify the part of PLM system to be evolved following a business change. It should be noted that evolution of product model usually launches the evolution of other parts of PLM systems but rarely the reverse. Evolution of other parts of PLM is not considered in this study. We focus on *product model evolution*.

As discussed in the previous chapter, product model determines how a product is structured, what are its components and what is other information to be related to product's structure. Therefore, product model encompasses product related information and its organization within an information system like PLM. Each product model has a pattern, which defines how the product model can be built by providing the various entities that should exist in the product model as well as the relationships between these entities. In some cases, this pattern provides the ontology of the product domain. Then, evolution of product model means any type of change in this organized structure. When dealing with product model evolution, we consider all types of modification in the product model as well as in its associated pattern. This is the subject of this chapter.

Evolution of product model was a subject of several research works. These researches can be classified on two directions.

- First direction analyzes the evolution problematic from a business and organizational viewpoint. In this approach, the evolution of information content, which is stored in the product model, is a subject of considered researches. In this viewpoint, the probable industrial changes and their effects on product model in PLM systems are studied. Researches in this axis focus on identification of changes in the product model semantics and main contributions aim to

preserve the coherence between the different product models (before / after change) from semantic point of view (so to not lose information on product). In this thesis, this axis is called "product model evolution from informational viewpoint".

- Second direction analyses the evolution problematic from an informatics point of view. In this approach, the evolution of information container, which stores product information, is the subject of considered researches. In this viewpoint, product model evolution problematic is treated like any other model evolution problematic. In this axis, the way that product model, or more generally all types of models, is represented and constructed in different modeling frameworks is discussed and the translation between these frameworks are analyzed and treated regarding to their structure and format. Research works in this axis propose a variety of model driven engineering based approaches. This axis is called in the rest of the chapter "model evolution from informatics viewpoint".

These two viewpoints of product model evolution will be detailed in this chapter. But before broaching the subject, some terminologies are defined in next section.

2. Modification, Evolution and Transformation

The terms modification, evolution and transformation are usually considered as synonyms. However, it's better to define them in order to avoid ambiguity regarding existent research works.

Evolution of an entity signifies its change during time, at different stages of its lifecycle. This changing is progressive and closely related to time axe. It can be considered as a sequence of small changes within the entity (changes of some entity's parts).

In the other hand, transformation of entity is a kind of change, which takes place in a moment of time, between two contexts. In this situation, changing may not be considered as a sequence of small changes within the entity, because of its brutal and fundamental nature. Contrary to evolution, transformation takes place between two systems, situations or languages. During the process of changing, all parts of the entity are simultaneously transformed.

Modification concerns changes within a part of entity. This modification doesn't occur during the lifecycle of this entity or even because its evolution. The result of modification is the entity with a changed part. It happens in a moment of entity's lifecycle. Moreover, the entity is not transformed to a new situation by a modification

As mentioned earlier, these 3 terms are often considered as synonyms but taking into account their variations may help the reader to understand more properly the differences between researches. In this report, our problematic is to change a product model between systems, which lies more on the concept of transformation. In our viewpoint, a product model may be evolved during its lifecycle, and transformed between different systems or situations.

3. Research review on product model evolution from informational viewpoint

In this section, we focus on research works dealing with evolution of information stored in PLM systems. These works' approach is based on the "content" of information stored in the system, rather than on its "container". Container refers to modeling languages and format, i.e.

information syntax, whereas content refers to the concepts included in the model to describe and specify the product, i.e. information semantic.

3.1.1. Evolution of information through different phases of product lifecycle

Researches in this category are about connecting, within a single model, product information issued from different representations. This category of researches is not about product model evolution problems; however mechanisms proposed in related works are interesting for our purpose. In this axe, product model restructuring during the different phases of product lifecycle is studied. As mentioned in the first chapter, the mission of PLM systems is to structure and manage the information and knowledge about a product during its lifecycle. Obviously, the structure and nature of product related information varies during different phases of this lifecycle, based on the specific requirements of each phase of design, industrialization, manufacturing, usage and end of life. For example, in product design, different options for a given product component are authorized, which are not allowed in the manufacturing (Männistö et al. 2001). Therefore the structure of product model varies during its lifecycle. The product model may have different viewpoints, which represent the product according to specific business requirements (Eynard et al. 2006) (Svensson & Malmqvist 2002). This variety in product model is generated by several actors with different business viewpoints such as manufacturing and design (with different fields such as electric, mechanic, and thermal). In this case, different models co-exist and represent different aspects of the same objects. Therefore the mechanism of transformation among the various models used in the enterprise during the product lifecycle may be interesting for our research issue.

The mission therefore is to manage links between various viewpoints. Researches in this topic propose two approaches:

1) Determining a modeling methodology that generates a very general model which is able to represent the different viewpoints,

2) Managing the transformation of product models and their synchronization through the different phases.

Sudarsan et al. proposed a generic framework for the design of product information modeling architecture. This research aims to maintain coherence between the various product configurations during lifecycle (Sudarsan et al. 2005). Eynard et al. have paid attention to the different breakdown business-viewpoints such as assembly or manufacturing and tried to avoid duplicating the data stored in PLM by recommending the mixed modeling structure (Eynard et al. 2006)(Eynard et al. 2004). Sevensson discussed the basic technical requirements for the Product Structure Management strategy. He highlighted the capabilities to create and manage the product models, keep the history of all of the activities around the product and ensuring the traceability and consistency of information. He proposed a master structure from which all of the views can be derived (Svensson & Malmqvist 2002). Gzara et al. proposed a method for building a product model by reuse of patterns. Based on a set of rules and criteria, the product model is built progressively by assembling different model fragments obtained by reuse of patterns (Gzara et al. 2003). Eastman et al. analyzed the evolutionary product model development with the viewpoint of database languages. They choose a suitable type of model

mapping which enables following and handling the evolution of product model during its lifecycle. Then they proposed a language able to facilitate the mapping, derivation of views, etc. (Charles Eastman & Jeng 1999). The PPO (product-process-organization) project aimed to propose a generic model, which describes all of aspects of information system (Noel & Roucoules 2008). They introduced a general meta-model on which all models in an information system can conform.

Zina et al. also worked on generic modeling of product. They divided the approaches used to take into account actors' viewpoints to two types: multi-view and multi-model. The multi view approach is based on the development of a single model starting from different views. Therefore there is only one unique generic model, and the derived models are the writable view of it (it is possible to change or modify the views, and this modification is also made in the generic model). The multi-model approach, meanwhile, proposes many models; each one is conformed to one business viewpoint. In this optics, the coherence rule is unavoidable (Zina et al. 2006).

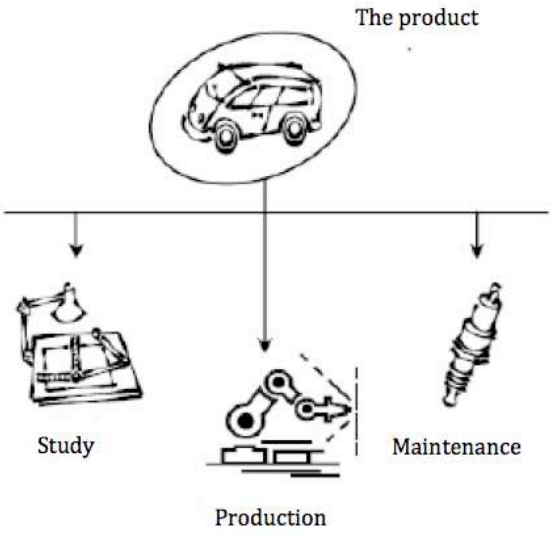


Figure 2-1, Product structure viewpoints (Zina et al. 2006)

In more specific way, Demoly et al propose a methodology based on assembly constraints in order to manage the evolution of product model between design and assembly phases (Demoly F. & Gomes 2009) (Demoly F. 2010). In their approach, they use assembly rules in design process in order to construct product structure, which contains sub-assemblies and parts.

To conclude, it can be noted that the main propositions in quoted researches are about developing a generic model, which manages the evolution during different phases of product lifecycle. This generic model contains all of necessary information on product. Therefore, different methods have been developed to construct specific models via this generic model. Evolution between models can be projected to this generic model and solved in simple manner. But these generic models are complicated and models should be designed according to them. The most part of these propositions don't cover already existing models. In order to use these methods, models should be designed and implemented regarding to methods' standards. This

shows the need to propose a methodology for treating product model evolution in general way that is not dependent to their designs.

3.1.2. Evolution of information within a single phase of product lifecycle

Previous section is devoted to analyze research works dealing with product model evolution through different viewpoints. In this section, we focus on the case where the product model evolves within a same phase of its lifecycle. As mentioned before, there are many reasons that lead to evolve or modify existing product models. Reorganization of the enterprise, change of the products' offer or evolution of the enterprise' information system can be sources of product model evolution. This latter case corresponds to the case where the product model migrates between 2 PLM systems; two sub-cases may be distinguished: (1) the enterprise uses the same PLM software (the core model is then unchanged) but migrates from one product generic model to a new one, (2) the enterprise uses a different PLM software (replacement) and then the core model is different.

It should be noted that the type of evolution considered in this section is different from managing simple product versions (here we assumed that the difference between product versions isn't so radical that the new version can be considered as a new offer). Product versions can be managed in lots of existing systems, but our problematic concerns managing a radical model evolution in PLM systems, which are not treated with features of existing systems.

Comparing to previous category (product viewpoints), few research works dealing with this type of evolution are observed. This lack may be relied on the complexity of the subject. This type of evolution hugely depends on the industrial context and user's requirements. Our industrial observations have proved the importance of this subject as well as the varieties of approaches adopted in literature to approach the problem (for example (Martin Gwyther 2008), which specifies the lack of existing PLM systems on this subject and shows its industrial needs). As mentioned before in this chapter, Eastman et all described the different types of product model evolution and proposed a methodology to handle this evolution (Charles Eastman & Jeng 1999). Their research, which didn't consider the product model complexity, tried to propose a new modeling language in order to support the modification of model's elements and support the synergies between different derived views of them. Therefore Their approach is simple and primitive, based on design of database (Eastman, C. M. & Bond A. 1991).

In this section two axes of product evolution within a design phase are discussed. Product model evolution may relate to context evolution. Product offer extension or product modeling framework modification can cause a context-oriented product model evolution. In this axe, modeling concepts evolve and the new product model must be constructed by using new concepts. The other axe is structure-based evolution. In this axe, the structure or configuration of product data evolves. In other words, in this axis, the motivation of evolution is the liaison and relationships between concepts in model. For example, when the evolution is about a passage from specific model to generic model for a product line, the evolution is structure-based. This structure represents the relationship between concepts existing in product modeling facilities of PLM system.

4. Research review on informatics evolution viewpoint

In this axe, we consider the evolution between different modeling and implementations frameworks. Models can be described and implemented in different environments. This axe focuses on transformation of models between these environments. We approach this axe according two considerations: Evolution of implementation framework (§4.1) and Evolution of modeling framework (§4.2) Software evolution

Lots of researches are done in order to study software evolution during time. They study how software's components change during different versions of software and how this evolution may be managed by optimizing this process and decreasing system or software eventual changes. This subject seems to be far from this thesis's problematic and explicitly not related to product modeling in PLM systems. But it should be noted that the evolution in software is key part of modeling-driven engineering initiatives. Therefore, it may help to understand the different parts of the proposed methodology, which is introduced in the third chapter. On the other hand, since that the core part of PLM systems is an application, then studying software evolution helps understanding PLM evolution problematic

Dhungana studies the potential types of evolution of models in the cases of software engineering (Dhungana et al. s. d.). In his approach, he uses two sections of meta-model evolution and model spacing to manage the product (software)-line evolution. Model fragments, which are managed independently, and meta-models may undergo an evolution, but by these strategies, evolution may be handled separately in these sections.

Evolution of data-centered information systems is another type of evolution discussed in software evolution field. Hink et al proposed a methodology to support database evolution caused by context evolution (Hick & Hainaut 2006). In their approach they used modern database modeling methods to handle the evolution in content and structure of databases.

4.1. Modeling language evolution

The other aspect of informatics evolution is the transformation of models between modeling languages. Modeling languages are used to build models, which represent real systems (Bézivin 2005). Modeling languages are used to conceptualize model's elements. Therefore, they have a significant effect on modeling processes. The modeling capabilities and features offered by these languages may influence on the resulting models and therefore the concepts and their structure in models may be different in different languages.

On the other hand, there are a variety of modeling languages used in information systems, software applications and frameworks. Besides, these frameworks and systems should be interoperable and a coherent flow of information must be set up between them. Transformation between modeling languages should guarantee an appropriate exchange of information between these systems. Moreover, transformation between languages may be useful to install a communication between systems that are fed with different modeling languages.

Several studies have dealt with evolution between modeling languages. They mainly focus on transformation between UML and other modeling languages developed in different contexts. Favre et al. worked on the transformation between SQL and UML(J. M. Favre & T. NGuyen 2005). In their research, they found the different elements existing in SQL and tried to determine their correspondence among UML concepts. As seen in this example, the problem is not the domain-related concepts (i.e. CRM, PLM, purchasing, planning, inventory systems etc.) but it is about the modeling elements, which are used to represent these concepts. In other words, the transformation does not deal with evolution of one system into another, but it s about evolution of this system described in a modeling language to the same system described in another modeling language. This is shown in figure 2.3. Figure 2.3.a illustrates an evolution process where source and target modeling framework are conform to a same modeling language. In this case, modeling frameworks differ upon to the concepts they held. Figure 2.3.b illustrates an evolution caused by migration between modeling languages. Modeling framework is the same in the source and target systems but the modeling language with which they are constructed are different. Obviously, with a proper methodology of this transformation and a suitable procedure to execute this type of evolution, the real-time exchange of data and information between different systems with different modeling languages is possible.

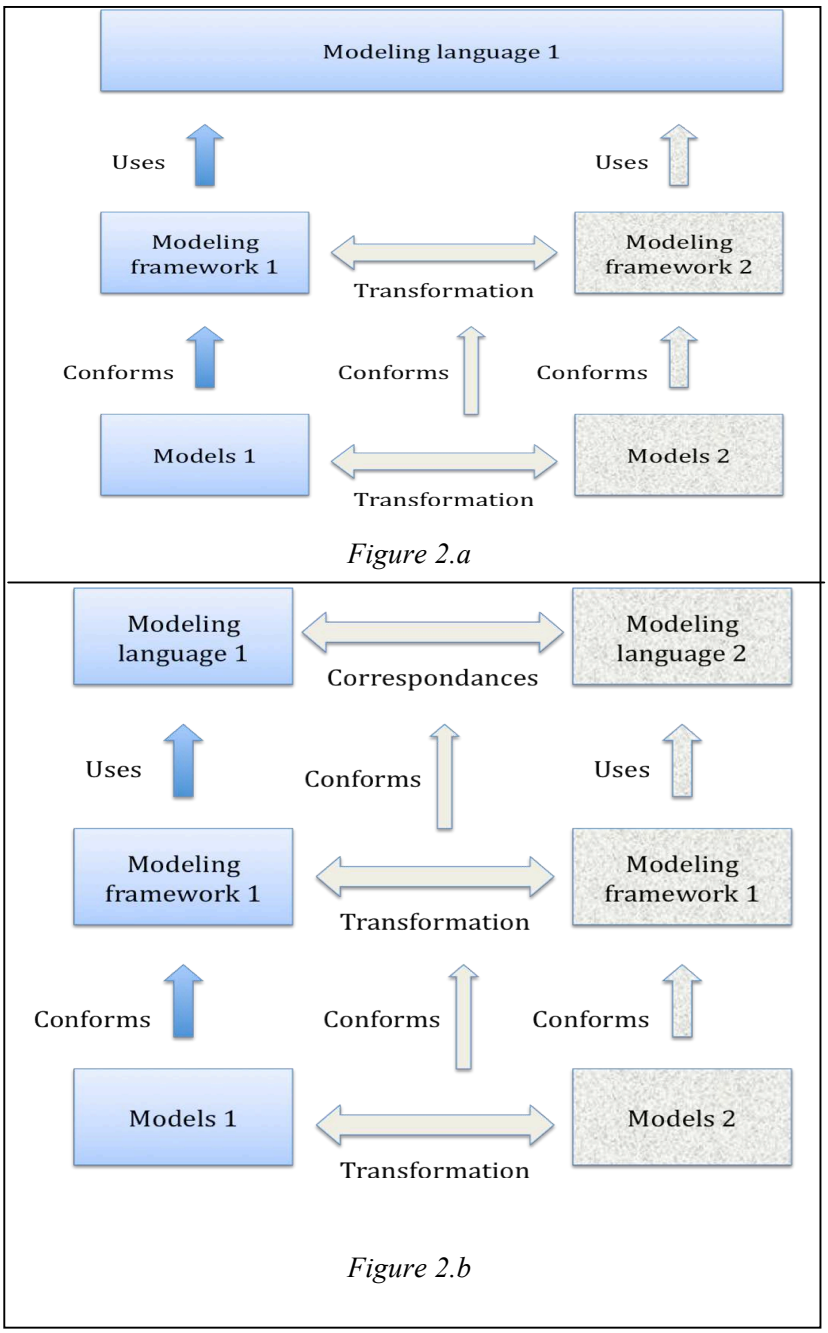


Figure 2-2, Evolution of models and modeling languages

Some research works have addressed the same problem (transformation between modeling languages) in the particular case of product data management. They focused on evolution among different product modeling languages by studying the process of data (models' instances) migration and standardizing the rules of data exchange between different languages, like STEP initiative (Männistö et al. 1998) (S. Rachuri et al. 2008). For example, some efforts have been done to do mapping between the models in different systems, such as PDM, CAD and CAM (Oh et al. 2001), or mapping among diverse representation language or schema of models and meta-models (Krause & Kaufmann 2007). In these cases, the process of MDE's model transformation is not always considered, even though the specification of mapping between models is a question of model transformation. It should be noted that the process of

standardization is useful as the systems are under design and implementation. Lots of problems happen when systems are designed and implemented previously, without proper specific standards.

This type of model evolution problem is interesting for the context of this thesis since product model evolution sometimes concerns the transformation of a product model through two PLM systems, which are using different modeling languages. In this case, the product model evolution follows the same procedure as common model evolution methodology described in the chapter 4, because the domain concepts are not different in the source and target models but the languages used to build models are different. The problematic is how to construct the same concepts with different modeling elements. Besides, for several modeling languages, some translators are designed and implemented in order to transform models between languages. Indeed, in product model evolution within PLM systems, a migration of data between these systems must be done. This migration may require these methods of translation since probably different systems use different modeling languages for data (instances of models which are stored in databases).

5. Conclusion of state of art

As argued earlier, product model evolution can be considered as a type of model evolution in different information systems. Subject of model evolution/transformation in informatics field has been discussed and treated in the last ten years. These researches have been enforced and directed by introduction of model driven engineering and model driven architecture in software engineering and development. Model driven engineering (MDE) which will be discussed more precisely in the next chapter, proposes some methods and regulations in order to manage model evolution.

In industrial information systems, the problematic of model evolution is more focused on the necessities related to industrial requirements. Based on previous discussions, the transformation of product model in PLM system may be caused by several types of industrial problematic.

As it was discussed in chapter one, analysis of product model evolution may be based on the classification of the business reasons that have initiated it. It should be noted that it is possible to classify the product model evolution regarding the PLM system's part, which undergoes the evolution (i.e. the architecture of PLM system). But, here we are interested by classification regarding the reason leading to the evolution and then the proposed solution follows according to the impacted part in the PLM system. Moreover, in this PhD thesis, the evolution problematic is treated only in a single part of PLM system that related to product modeling that consists of two main parts, product models and product modeling facilities. Therefore, classification of PLM reasons seems more suitable here.

There are different types of reasons, which can cause product model evolution. They are listed in the table 4.

Related to product:

Physical product	Product offer extension (product definition)
	Modification of product components
Product information to be saved in the system	Adding or deleting the product information
	Modification of product modelling framework

Related to organisation and decision:

Inter-Organisational modifications	Adaptation of PLM to partner's systems
	Merging PLM to partner's systems
Inter-Organisational modifications	Adaptation of PLM to other enterprise's systems

Related to informatics tool:

Modification/Update/Upgrade
Integration/Merging
Replacement

Related to process :

Modification of process related to product configuration

Tableau 1, Reasons of PLM evolution

As described in this table, PLM evolution reasons are classified on four main groups. Reasons related to product, to process, to organization, and to informatics technologies. As discussed earlier, this classification doesn't reflect the part of PLM systems that undergoes the evolution. It means that for example, the reasons, which are related to organization or process, may effect product model, and cause product model evolution. It depends on the specific context of evolution. In some case, the adaption of PLM system to other enterprise systems may initiate product model evolution, if the adaption aims to integrate modeling framework and construct a common models, usable for all systems. Therefore it stays upon the case and context of evolution. Moreover, even for a specific industrial reasons, there are different alternative solutions with different PLM system's parts to be evolved. All of these choices should be done according to the context properties and user's requirements.

6. Problematic

Product configuration, i.e. the set of functional and physical features of a product, is the principal concept in PLM systems. PLM systems offer a set of mechanisms in order to support

product configuration management. Product configuration includes the documents and BOMs (Bill of Materials). BOMs are instantiations of one or more product models in a database. Therefore the role of product model within the PLM is to structure all product-related information and their connections (Schuh et al. 2008).

In this context, any modification in the product model plays a fundamental role in the evolution of the whole PLM application. Product model evolution means modification of the model architecture, its elements and their relations. In the object-oriented modeling approach, it means the modification of classes, attributes, associations, and methods. These items show the importance of the product model evolution analysis in relation to the PLM evolution. In fact, during the procedure of PLM evolution, the most delicate subject that can affect the whole procedure is how product model should be modified. Moreover, as mentioned before, some of PLM evolution causes are related to product model modification itself.

In the other hand, PLM implementation is costly and time-consuming effort for an enterprise. The enterprise may works for couple of years to set up a PLM system. This PLM system is fit to all of enterprise's properties and fulfills all of its requirements. Additionally, it formalizes some of its procedures and commands. But as described in the first chapter, evolution is probable in this changing world. Therefore treating the evolution problematic is inevitable to help enterprise to manage evolutions in their information systems in the most proper and optimized way. This helps them to transform properly their knowledge, skills, and requirements to the new system with a minimum of cost, time and effort, and probably, in the automatic technique. Our problematic is a crucial subject in PLM system management in an enterprise.

But, as shown in this chapter, previous researches don't cover all of different types of product model evolution. Several of these researches concerns the evolution of product model during its lifecycle, which means the modification of model caused by the different process of design, manufacturing, etc. It can be cited that this modification may be done on the instantiation of product model. But the approach under the study in this thesis regards to modification of product model in one instant of time. In this case, the product model is changed, not because of the design and manufacturing process requirements, but in the reason of evolution in the information system or the enterprise organization. This classification described in figure 2.3.

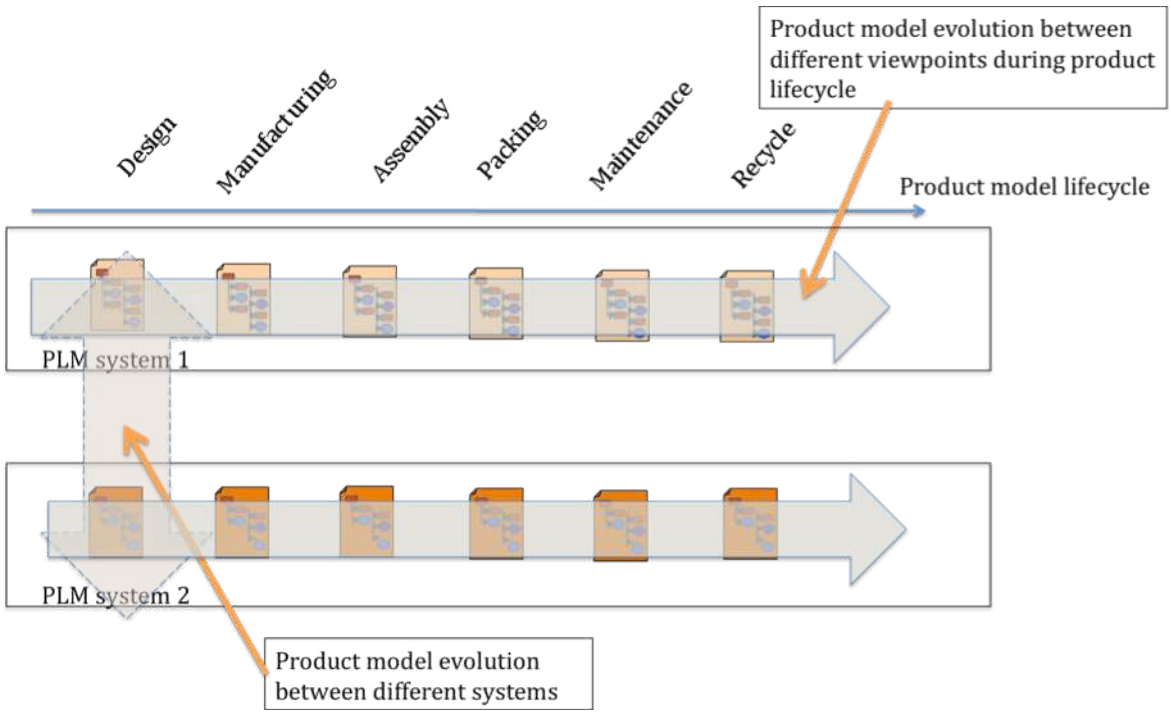


Figure 2-3, Product model evolution between systems/viewpoints

As shown in figure 3, product model evolution, which is interesting in this thesis, is the evolution that takes place between systems. Evidently, proposed method may help the users to preserve the consistency between different viewpoints.

Several other researches cover the technical and informatics subjects of model evolution. They proposed methods and tools for software or application evolution, as well as translation between different modeling languages. They didn't treat the subject of conceptual modeling evolution within systems.

Enterprises need a proper general procedure for evolution of their product models, between different systems, probably with different modeling languages, which can preserve the consistency of the coherence between different phases of product models during its lifecycle. Moreover, this procedure should be able to treat the problematic of evolution of whole system, evolution, which cause the general modification of system by changing the core of PLM systems, or evolution with the less complication such as product offer renovation.

6.1. Functions of proposed procedure

As mentioned before, product model evolution returns to the problem of model transformation. In this context, one of the major questions is to determine if the implementation of model transformation should be started in the meta-model level or if the transformation is not related to the upper (meta-model). Each PLM system uses its own structure to design the product model, i.e. the meta-model allowing to build the various product models are different in the PLM system. In the case where the PLM system itself is changed, it should be firstly ensured that the meta-model transformation is necessary or not. In the other hand, in the case of model

transformation with the same meta-model, the procedure of model transformation probably conforms to the meta-model. Furthermore, it must determine if the target systems meta-models are defined previously or they should be constructed during model evolution. In other words, the new system's properties and specification are known in the beginning of model evolution or not. For example, in the case of adaption a PLM system to the system of the other bigger partners, generally, the new system's specifications are known.

Proposed procedure must be able to tackle all types of product model evolution between different systems, even if product meta-model evolves. Moreover, it must define some methods to determine the new system's specifications. It consists of its design or enrichment, depending on the context of problematic. Forth chapter tries to propose a structured procedure to fulfill these requirements.

Chapter 3 Model Evolution, Our Approach

1. Introduction

Problematic of product model evolution in industrial information system seems to be a crucial subject in industries which deal with complex and very composed products, but the solutions proposed in the academic media may not be structured on a very standard approach in information science and technology. In other words, propositions were made for a specific situation or context or didn't relay on a general application. As a result, these propositions didn't study product model evolution as a generic problematic but in a very specific way to solve a local problem.

Approaches aiming to handle and solve product model evolution may be classified in two directions:

(1) Approaches, based on product conceptual modeling, utilize more the product model concepts and try to re-configure these concepts so that the resulting product model conforms to the target specifications. For example they use the concept of modular modeling and try to reconstruct the models from these modular parts. These approaches are more oriented towards industrial context of problematic. The specification of product design and its concepts normally don't let to be used in the other application cases.

(2) The other approach, which is ours, depends on the model itself. The principals and high generic level of our approach don't depend on the specification of product models. Product modeling framework requirements are used during evolution process and helped the system to find the similarities and divergences. In other words, our approach observes the product only as an element of model with some specifications, which specify it from other elements of model. There for, our methodology is not limited to a specific context, but in the more general level, it may solve the product model evolution. More in detail, we preferred to observe concepts, parts,

configurations, links and specification of a product by a modeling-oriented view, for example UML diagrams. With this "transformation" of product model evolution problematic, a new problematic may be formed that is more oriented on the general model evolution.

In our methodology, we profit from the rules and concepts of MDE, Model Driven Engineering. This is new approach that is pioneer in design and implement informatics systems as well as information systems, is described in this chapter. Methods based on MDE deal with models as a general concepts. In their specifications, model has a large definition. However, the general process and proposition should be customized in order to be applicable for a specific problematic, but these customization are also possible by their methods. Therefore, MDE has a almost complete framework for model management and engineering, which prepare a very huge and vast collection of tools and methods.

At the end of this chapter we will justify the utility and importance of MDE implication in product model evolution.

2. Model Driven Engineering: principals

Challenges faced in software development field during its huge advancement in recent years, justified the introduction of new methods and technologies, which reduce the related complexity and set up an efficient and fast development process. These challenges are:

- Complexity: during these past years, software industry has suffered from a huge increase of problem and solution complexity (van Sinderen & Ferreira Pires 2004). Requirements against computer systems are becoming more and more complicated and technological solutions developed to fulfill these problems are complex. This complexity comes form size of software's, number of modules and services provided by software, number of users, possibility of running on web, different platforms and operating systems, etc. Lots of programming languages, development platforms and technologies (such as middleware and Service-Oriented Architecture) necessitate a modern global method of software development able to reformulate the industrial problematic in a very general way which is comprehensive for several technological platforms and also helps the solution developers to understand and simplify their propositions.
- Time to market: huge rapidity of software development demands a very fast response to needs in order to protect own market. So the problem identification and solution development processes should be fast and efficient. One of the most important methods that aimed to decrease development time is the opportunity of reusing different components; and setting up a library. But the complexity related to platform technologies make this reusing so difficult. Therefore a new method should be proposed in order to facilitate this reusing for more complex situations.
- Collaboration on software development: complexity and rapidity of software development, force enterprises to collaborate and develop them mutually. This collaboration requires setting up a common framework of software development by introducing more integrated methods of design and implementation. These methods

should be capable of keeping the particularities of each collaborator, and simultaneously guarantee a permanent comprehensive communication.

In order to fulfill the necessities mentioned above, a new paradigm of software development, based on abstraction, was introduced, Model driven engineering (MDE). MDE separates design from implementation and reduce the ambiguity of development (J. M. Favre 2004b). Firstly, any computer system should be designed, independently from platform on which it may be implemented. Then, implementation is done by *transformation* of designed concepts to implementation concepts by entering platform's requirements.

In design phase of software development, the design models are constructed, assembled, and refined. This phase is done interdependently from platform. This model is called Platform Independent Model (PIM). All of further probable evolutions are then done on this level of models. The independent models are then transformed to models, which are executable on different platforms. These models are called Platform Specific Models (PSM). The requirements and particularities of platform should be taken into account. These particularities form the "platform model". The primitive task of MDE was to establish this transformation between PIM and PSM. In order to facilitate and automate this transformation, the concepts of meta-model and model are developed and accustomed to MDE. This procedure is shown in figure 3.1

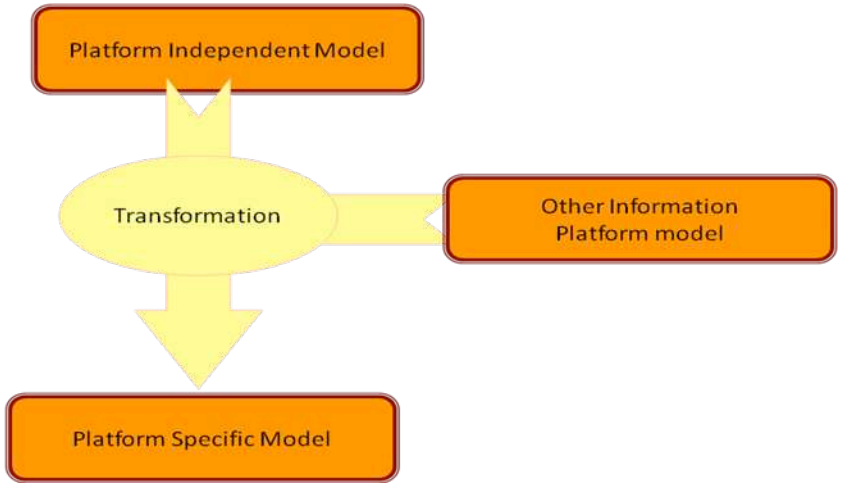


Figure 3-1, PIM, PSM and transformation (OMG 2001)

This brief history shows the needs, which led to invention of MDE and standardize it into MDA (Model driven Architecture, standard of OMG). But this new paradigm of software development wasn't limited to PIM/PSM transformation. It means that the new areas proposed and developed in MDE isn't limited to model implementation. All other types of model transformation and evolution between different systems, even if both of them are platform independent can be treated by MDE mechanisms. The new concepts and methods proposed and developed in MDE are used in other areas of computer engineering model engineering. The definition of model and meta-model and model transformation, which are the interesting parts in this research are presented in the following parts.

2.1. Model and meta-model concepts

The trends in previous works of software development, especially those on object-oriented programming are to increase the abstraction of information, by constructing models. Concept models are illustrated by class diagram. Then other model diagrams use these concept's models in order to specify different aspects of concepts, like their transition. The object level, which represents the physical objects, is the instances of models. Therefore the relationship between object level and model level is "instantiation".

Creating models should follow some rules, which specify the concepts and their relationships in models. These rules form a modeling language. Modeling language determines which concept can be present in model, with which kind of specifications, and which constraints should be respected. This is called "meta-model" of model (J. M. Favre 2004a). The relationship between a model and its meta-model is called "conformation". Conformation means that the model respects the specifications imposed by meta-model.

A meta-model may define the representation language of a model. For example, the meta-model of UML defines how we can construct a model with this language. It describes the concepts of UML diagrams and their relationship. This meta-model is called MOF (MetaObject Facility). Because of the hierarchical relationship between models and meta-models, this meta-model of UML is the Meta-Meta-Model of all models defined in UML.

The relationships between objects, model, meta-model and meta-meta-models concepts are illustrated in figure 3.2. A simplified view of the MOF is presented in figure 3.3.

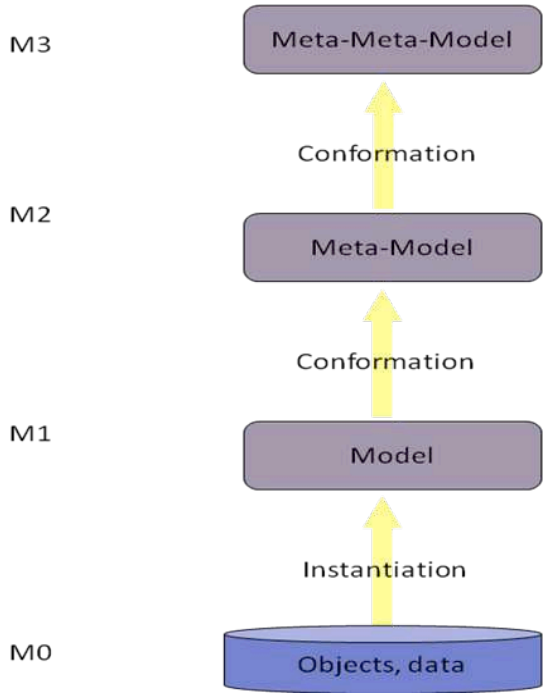


Figure 3-2, object/model/meta-model/meta-meta-model

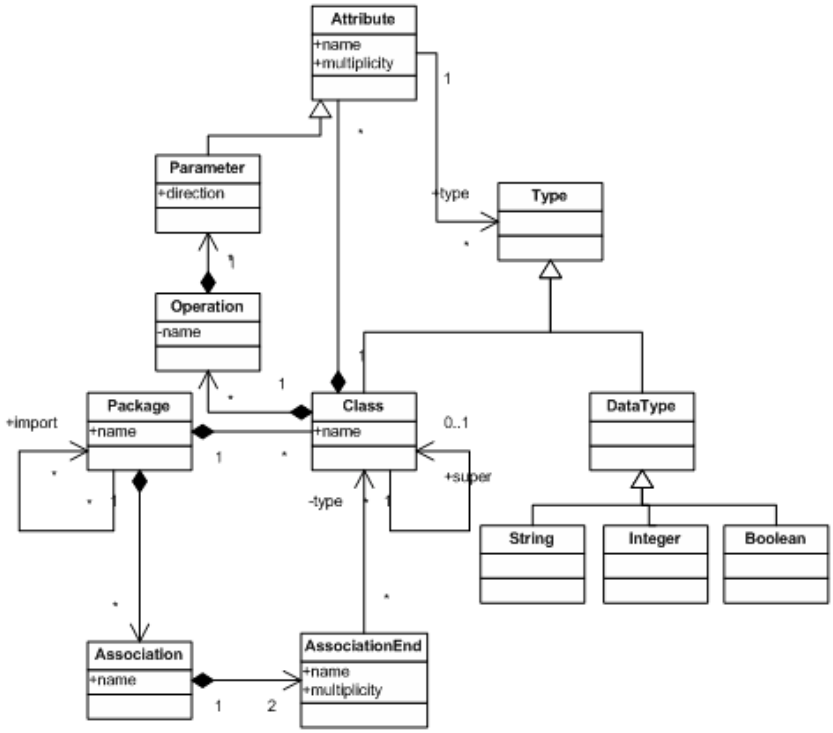


Figure 3-3, Simplified UML meta-model (MOF)

Figure 3.4 shows the relationship between models of UML and MOF. This structure is standardized by OMG in the MDA. UML in this figure signifies the language, which contains the concepts of class, associations, etc. In this structure MOF conforms to itself (Kurtev & K. G. van den Berg 2004) but in other approaches of MDE, this hierarchy can be continued. It means that we may have meta-meta-meta model which describes the meta-meta-model concepts. In other words, in general approach to MDE, which is the general extension of MDA standards, all of "meta" terms are relatives and the based on the user's needs.

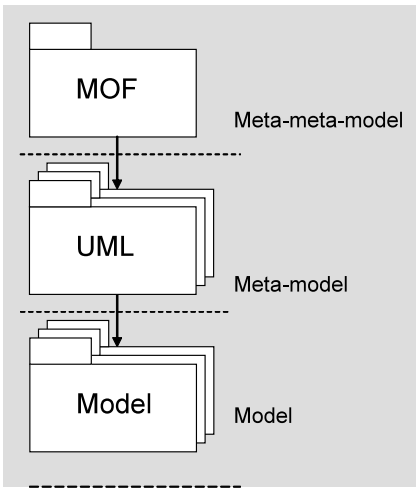


Figure 3-4, MOF and Models in UML, example standard de figure 3.2

The meta-model may identify the specifications of a model nature. The concepts that can be used to construct a model, the allowed relationship between these concepts and the constraints are predefined in meta-model. All of these specifications depend on specific purpose of modeling. This is the domain modeling, that shape a meta-model and contains modeling requirements. Each concept in models, as well as relationships should be "conformed" to an entity in meta-model. In another viewpoint, meta-models are the accurate and formal result of ontology engineering(Goknil & Topaloglu 2005). Figure 3.5 shows a simple example of two models and their meta-model.

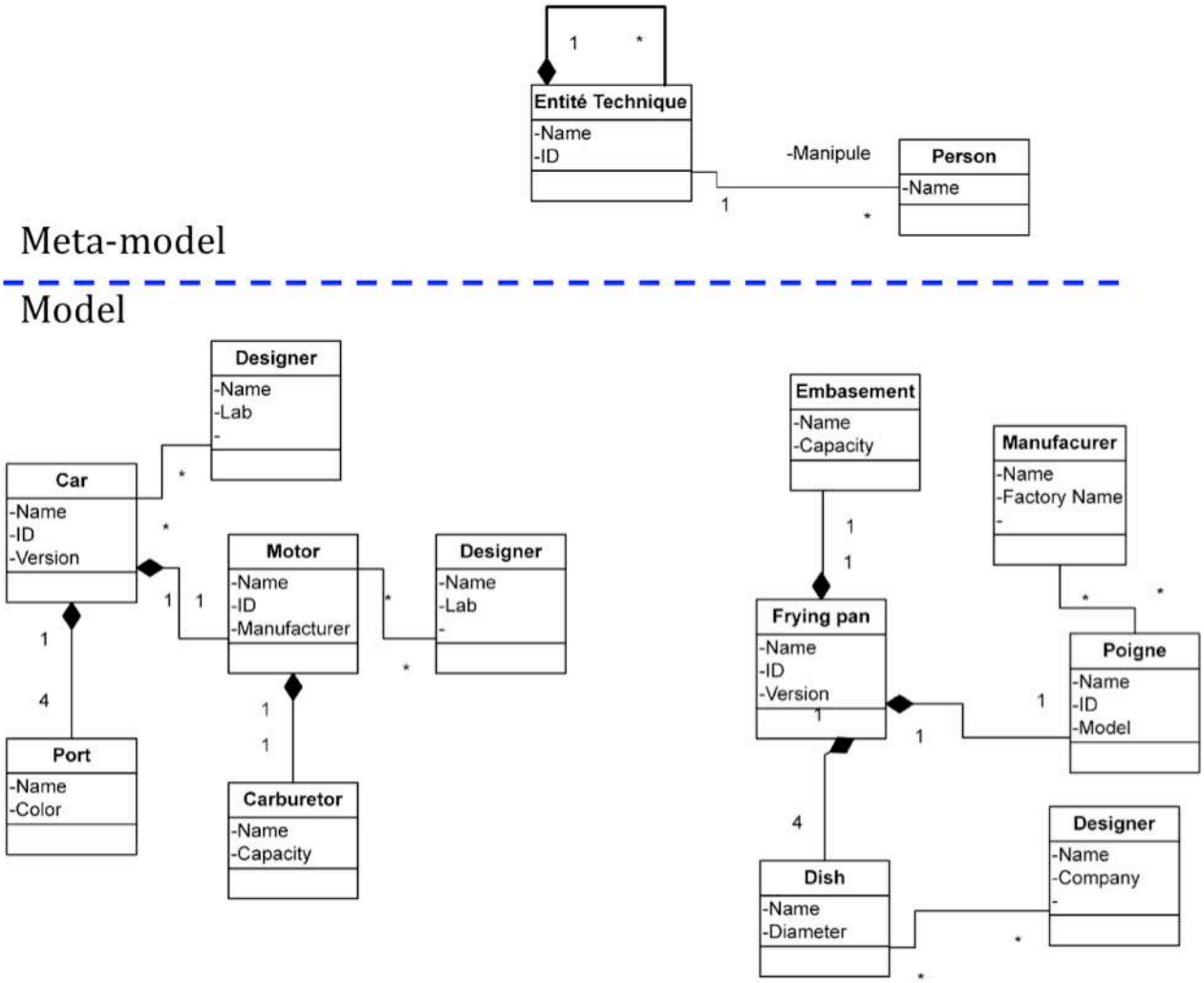


Figure 3-5, Two models and their meta-model

2.2. Model Transformation

Model transformation is the most powerful tool in MDE that is the core of its usability and advantages(M. D. Del Fabro & P. Valduriez 2009),(Pierantonio et al. 2007). As mentioned above, model transformation was invented to ensure the migration between PIM and PSM(OMG 2001). But, the proposed methods may be applicable for other types of transformation between models, even in the same level of dependence on platform. Many methods and languages are

proposed in research and software industries for formulating model transformation, in a general form, especially on PSM/PIM transformation methods (Gardner & Griffin 2003)(Frédéric Jouault et al. 2008) or more advanced model transformation used in translation between different concept frameworks(Willink 2003)(Varró & Balogh 2007)(Kurtev & K. G. van den Berg 2005)(Frédéric Jouault et al. 2008)(Frédéric Jouault et al. 2008)(Frédéric Jouault et al. 2008)(Frédéric Jouault et al. 2008)(Frédéric Jouault et al. 2008)(Frédéric Jouault et al. 2008)(Frédéric Jouault et al. 2008). The result of these approaches, which will be discussed later in this chapter, shows the capacity of model transformation to solve the problematic beyond to PSM/PIM transformation for one, it was invented.

Mechanism of model transformation is based on the meta-models. The overall mechanism is presented in figure 3.6. The term of transformation model in this figure contains the comparison results between meta-models. Actually if it is not a model in some ways, but some technical media chose this name for this part of transformation motor.

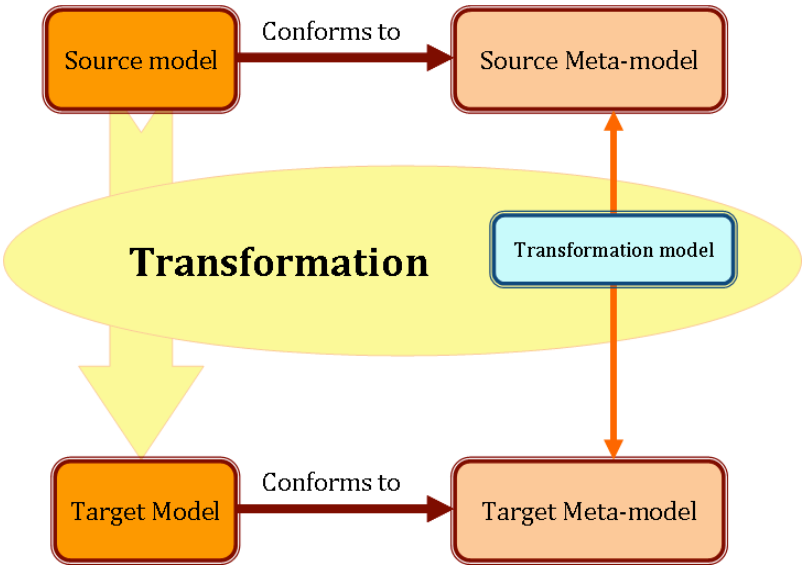


Figure 3-6, Model Transformation mechanism

In this approach, meta-models are compared in order to find the similarities between them. According to these similarities, for each concept in source meta-model, a correspondent in target meta-model may be found. The most important task in model transformation is finding these correspondents. When these correspondences are set up, the transformation will be done by creating target model from source model. Each concept in source model conforms to an entity in source meta-model. Its correspondent in target model will be a concept that conforms to the entity in target meta-model correspondent to this entity in source meta-model. This is illustrated in figure 3.7.

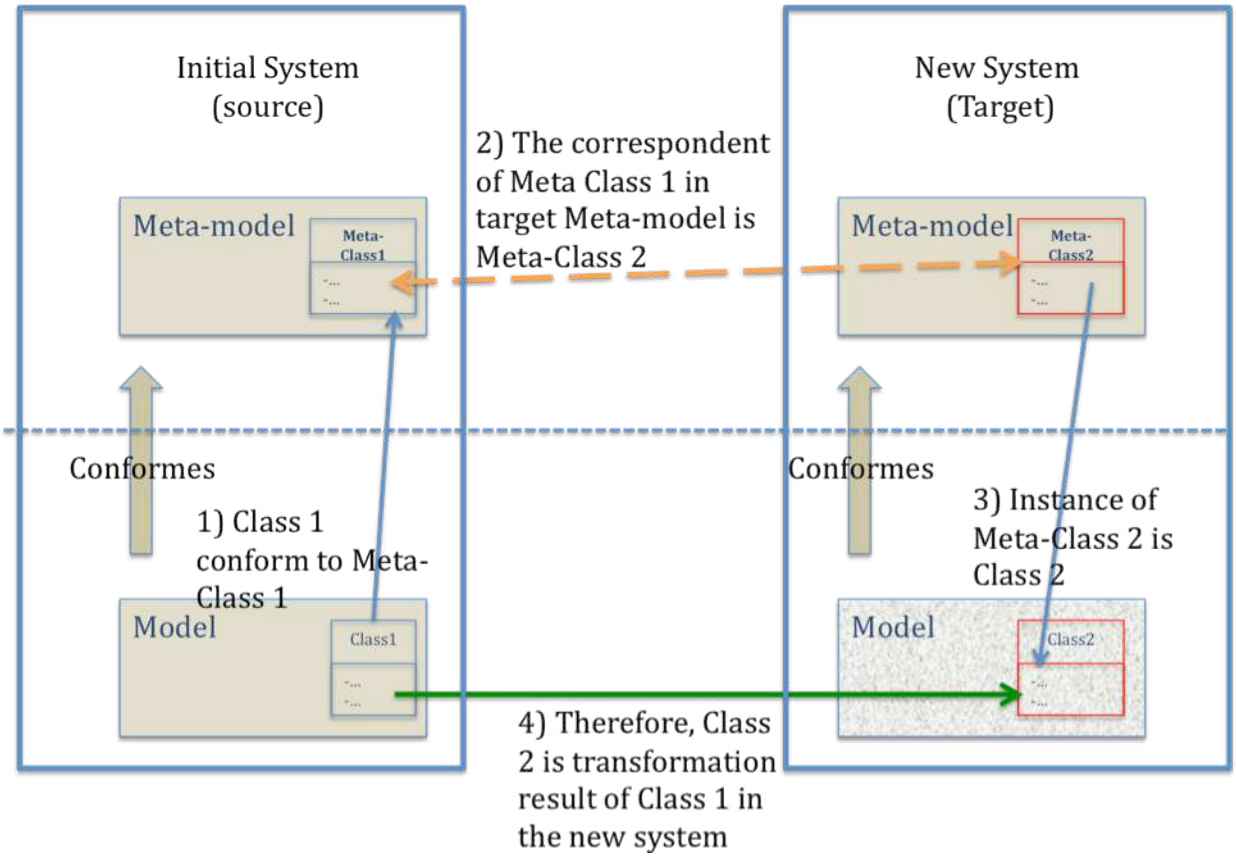


Figure 3-7, Model Transformation

Lots of researchers, especially in software development, work on different transformation languages and automated model transformations (Frédéric Jouault & Kurtev 2007). Their researches focus on different types of rules, which are used in model transformation. Tools that are under MDA standards are related to this part(Kurtev et al. 2007). It comes from the importance of this part in whole organization of MDE.

Process of establishing correspondences was addressed in a huge number of researches in MDE domain. This process enormously depends on the models' domain. Model differentiation or comparison aims to compare models and identify similarities between them. It's clear that it depends on the model semantics, which are represented by modeling language. In other words, this is the modeling language that imposes and defines these semantics. Therefore proposed methods for model differentiation are normally specific to a particular modeling language. The most common and general language is UML. Several researches devoted to present a generic model comparison of UML models: by using graphs as a structure of meta-model (Winkelmann et al. 2008), model regression which emphasis on the recognition of modification on a meta-model during time (Muccini 2007), matching transformation as one of the pioneer structured activities to construct a semi-automated model matching, but for some simple examples in (M. D. Del Fabro & P. Valduriez 2009), model differentiation with meta-model-independent algorithms(Lin et al. 2007), etc.

As an example, (Y. Lin 2007)'s definition of model comparison goes from textual differentiation to more generic model comparison based on graph isomorphism. But she tried to introduce a new method of model comparison, which is not dependant on modeling language, facing the new trend to language invention through domain modeling.

Del Fabro has categorized many tools and methods for model mapping and analyzed them (M. D. Del Fabro 2007). In his analysis, he considered some criteria for model comparison, like simplicity, as being powerful by letting express complex rules in a simple way, traceability of transformation, interoperability, etc. He concluded that in mapping management, a trade-off should be done between the simplicity and genericity of proposed methods.

As a conclusion to these methods and approaches to the early phase of model transformation, it should be noted that these researches are focalized on structure comparison or concepts' language comparison. It means that in their informatics and software viewpoint, the differences between meta-models are more related to their terminology or graph representation. But in this research, the semantic comparison between entities existing in meta-models is crucial. As a consequence, our approach in model comparison, presented in detail in the next chapter, is developed in a way more adapted to product model context. In this approach, we emphasizes on the definition of concepts and relationships existing in product design and manufacturing and their evolution with a modeling look. The more informatics technical parts of models, which are treated in model transformation is out of the scope of this research. The method used in this research is an element-to-element method (M. D. Del Fabro & P. Valduriez 2009), which tries to compare the meta-models' entities, one by one, and regarding to pre-defined rules and list of concepts in the product-modeling domain, find the corresponding between meta-model elements. This approach is more reliable outside of informatics viewpoint because of the importance of semantics definition in these applications. Our methodology is followed by a simple structure similarity method.

3. MDE and PLM

As it was discussed in the previous chapter, the problematic of model evolution in PLM systems has not yet been treated properly. The methods generally used on the subject of product model evolution are more based on the business-viewpoint; the proposed solutions almost relate to operational issues. But we propose that the problematic of product model evolution can be treated and explained by informatics-based approaches. This approach increases the rapidity of solution as well as its genericity, and also it reduces the complexity of industrial problematic by an appropriate problematic formulation. With this viewpoint, the problematic, what may be so vast and complex, can be represented by a formal and comprehensive standard language, used in informatics media. Moreover, evolution has an exact semantic in MDE. Therefore, treating evolution of models in information systems lies properly in this semantics and can be done by tools and methods of MDE.

Some of facts that prove this appropriateness are listed here:

- The process of product model development in PLM systems is similar to the process of model creation from its associated meta-model. The PLM core section defines the entities that may be used in order to construct product

models. Their relationships are explained and the constraints, which must be respected, are listed. The different classes and their allowed attributes as well as the description of different types of relationships between future-constructed elements of product models must be predefined. Therefore, close comparable to model development procedure of MDE; in PLM systems, all product models must conform to these descriptions, called Product meta-model. This tight similarity between the overall MDE and PLM context is a reason to use MDE methods in our problematic.

- MDE main problematic is model transformation. Lots of researches have focalized on design and improvement of the model transformation process. In the other hand, our problematic can be reformulated to a model transformation problem. This reformulation will be discussed in chapter 4. It should be noted that the transformation procedure should be adjusted for each type of context. Especially, similarity framework highly depends on the domain in which the evolution takes place.
- MDE methods are widely used in software development. Increasing tendency to use MDE makes it a standard approach to solve problems related to transformation of models. This attention makes MDE related methods more valid, structured, standard and accepted in information systems and information technology areas. (Brambilla et al. 2009)
- Model transformation proposes some tools and automated methods, which can be useful during the research. These tools are normally validated by the research community and are reliable.
- Procedures developed for product model evolution can be reused for other similar problematic, because they follow a standard form. This use of developed procedures is useful for other types of evolution, which takes place in PLM systems.
- Using MDE methods helps to implement the solution more properly in PLM or other information systems. Since, even if the initial problem of evolution is described in business viewpoint, the solution has an informatics nature and should be formulized with informatics concepts, which makes using MDE approach more credible.

Architecture of PLM system has been presented in first chapter. This architecture consists of two fundamental sections: (i) a core section that contains all PLM facilities proposed by PLM editors and, (ii) a customization layer that is added to PLM during its implementation in an enterprise and envelops all specific functionalities. In order to implement MDE viewpoint, it's tried to represent the structure and concepts existing in the PLM system with MDE principal semantics and concepts. The division discussed earlier is conformed to this approach.

By mixing MDE and PLM architecture, the core section may contain the product meta-models. This depends on the architecture of the considered PLM commercial tool. All PLM tools have a predefined meta-model for product's data. This meta-model, may be so primitive and PLM tool lets user constructing a meta-model, which is adapted to its requirements and

particularities. In other cases, the PLM tool proposes a rich library of predefined concepts for product model construction. It's clear that predefined meta-models of a PLM tool are not let to be changed or modified, unless for one tool in a specific context and problematic allows the user to define and propose its own meta-model. This aspect, important in the procedure of model evolution, will be presented in the next chapter.

Product models are then created by using concepts proposed in product meta-model. It is done in product modeling section. This section offers a toolbox of all allowed concepts and controls the constraints imposed by meta-model, that modelers should respect during product modeling.

Therefore PLM architecture with Model Driven Engineering approach keeps the previous division.

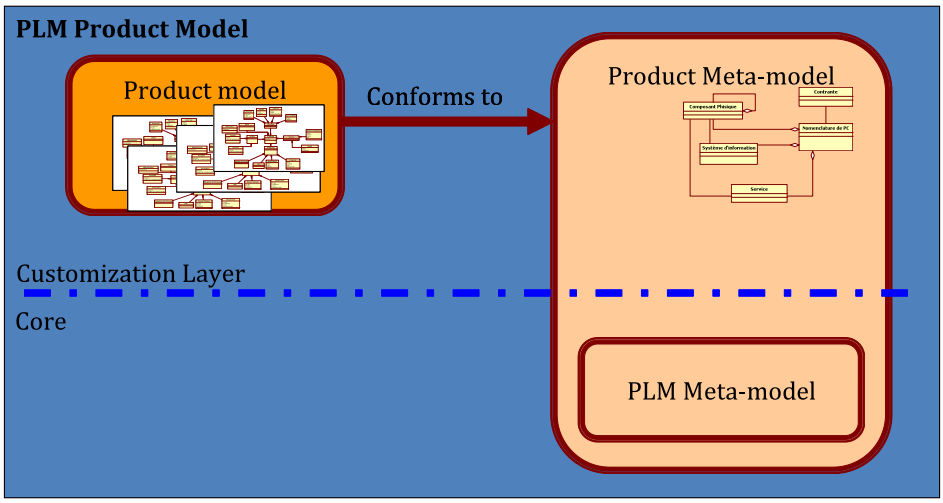


Figure 3-8, PLM product modeling facilities

4. Conclusion

Model Driven Engineering, which is a standard approach, was proposed to facilitate software development processes, but its principals seem to be suitable for use in other modeling areas, and not only software engineering. Lots of other modeling frameworks, which use formal modeling languages can profit from this approach and its related tools and techniques. The core of MDE is model transformation. Researches on model transformation aim to propose proper languages and methods automate transformation processes and set up a framework for model comparison.

MDE is an interesting approach to govern evolution of product model in PLM systems. It's a reliable and accepted method for models evolution. Methods that are designed under standards of model driven engineering are à priori reusable. These advantages make MDE and associated technologies more interesting for product model evolution problematic. MDE has not been used widely in PLM design and evolution. Some research have been done to solve interoperability problems between PLM and other design support systems (such as CAD systems) or communication problems between different standards (Sadeghi et al. 2009)(Krause & Kaufmann 2007)(Panetto s. d.), but for product model evolution, a gap is identified.

In order to profit from the MDE approach for our research, some task must be fulfilled. The industrial problematic should be reformulated as a MDE-type problem. The model comparison should be adapted to the context of product modeling. Therefore, the ontology of product model should be defined within PLM systems. Moreover all possible and probable scenarios of product model evolution should be identified and treated. Next chapter consists of proposing a new methodology for answering product model evolution by treating these challenges.

Chapter 4 Product Model Evolution

1. Beginning of discussion

As discussed in the previous chapters, product model evolution is a critical subject of product information management in PLM systems, due to the central role of Product models and BOMs. Product models consist of a hierarchical decomposition of product information and structure during different phases of product lifecycle

As discussed in chapter 3, the retained approach for managing product model evolution is model driven engineering (MDE). One important feature of MDE is "Model transformation"; a semi-standard approach for solving problems related to model evolution in software engineering: we call it semi-standard because event model transformation presents a generic approach to solve all kinds of model evolution, but the tools and methods that have been developed to support this approach do not already have implemented a general algorithm for any application, therefore, various methods must be applied. Anyway, the increasing usage and development of this approach in computer science may be sufficient incitation for an experiment in the domain of product models and bill of material transformation. However, as it is a brand new approach, its integration in industrial information systems engineering process is restricted to the implementation phase. Analysis, design and evolution of these systems are not already treated entirely and are restricted to academic research. But with the growth of complexity in industrial information systems and the needs for their rapid development, the use of MDE approach will become more and more unavoidable.

This chapter aims at the description of a solution regarding this approach. In this proposition, product model evolution is treated by using model transformation techniques. A general description of the proposition is presented below in figure 4.1. The different phases are developed and detailed in the rest of the chapter. These phases construct a general roadmap to solve the evolution question. Let's notice that the proposed roadmap starts with analyzing industrial problematic. This is mandatory since the following phases depend on the classification

of the industrial motivations behind PLM evolution. Then the industrial problematic should be translated to a computer-supported language. It is essential because the solution proposed is based on the computer-supported part of MDE.

The next steps are about finding the particularity of the under-study problematic and specifying on which scenario it relies. Depending on the appropriate scenario, the target meta-model can be identified or constructed. Due to source and target meta-models, the similarity framework should be set up by their comparison, and the proper relationships will be established. The model transformation can then be executed according to these relationships. These steps will be discussed in detail in the following section, but it must be noted here that the identification/construction of meta-models and their comparison has a significant role in all the model evolution processes. All the process of model evolution depends on the way that the meta-models have been constructed and compared. This assumption is discussed later.

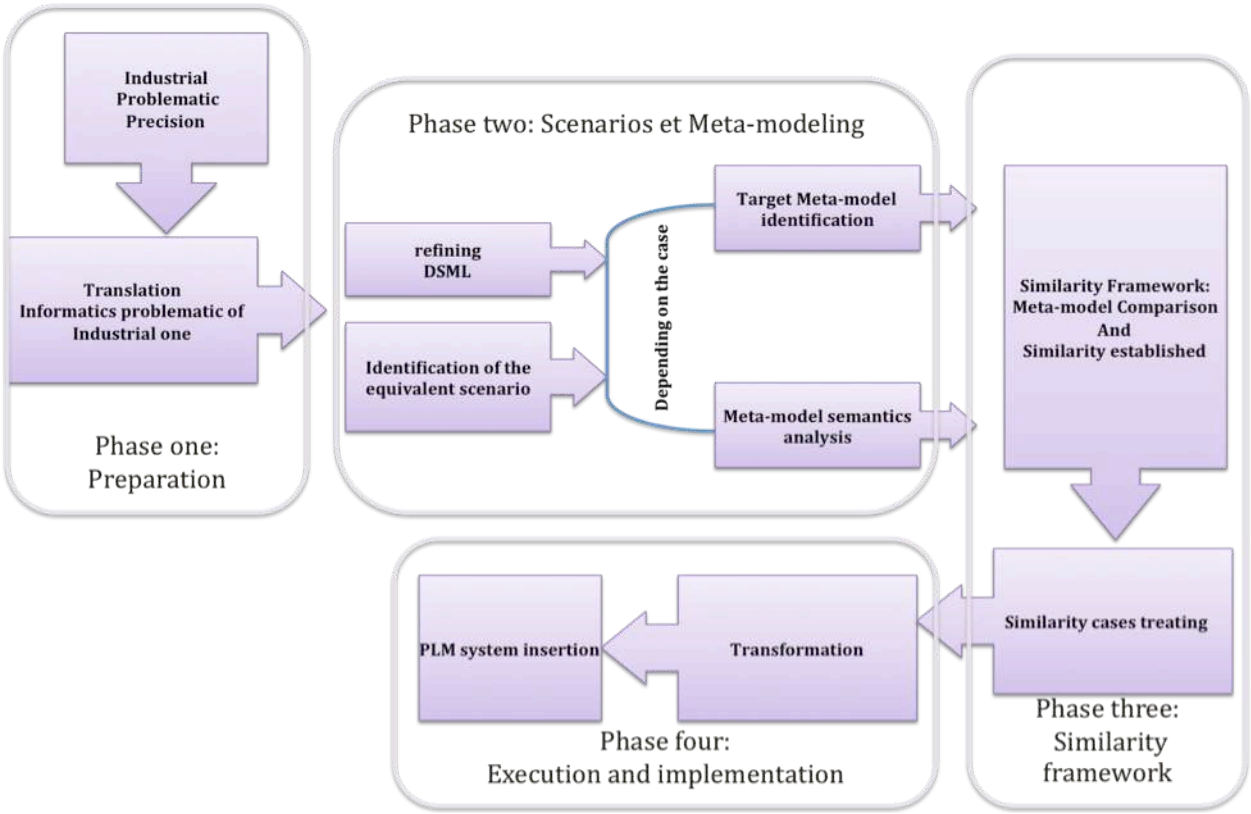


Figure 4-1, Overall view of the proposition

2. Phase one: Preparation: modeling industrial problematic

Product model in PLM systems may change due to an industrial initiative. This industrial initiative, which was described in the second chapter, is usually expressed with a business definition. Situations such as product configuration modification or PLM system replacement may cause lead to product model evolution. But they are generally not enough precise and accurate to be treated with the computer-supported methods, because they are described with a specific business terminology. The first stage of the evolution treatment is thus to identify the detailed problematic and find its corresponding solution in terms of MDE classification. Figure 4.2 illustrates the position this first phase.

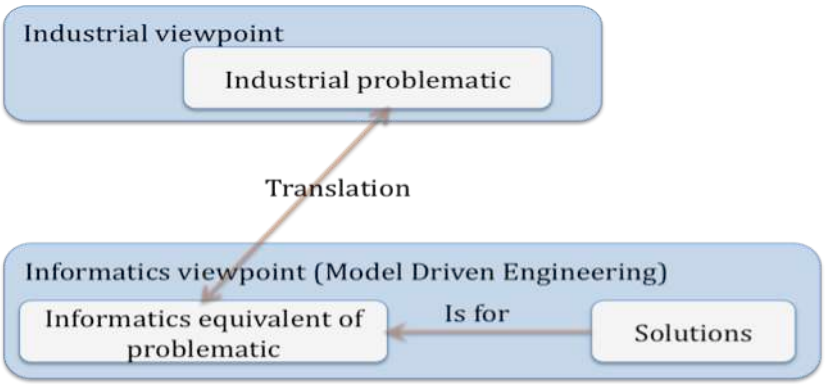


Figure 4-2, Translation of evolution problematic

Since the aimed solution to resolve an industrial problematic is of informatics nature. the translation of the industrial viewpoint into an informatics viewpoint is necessary to find the appropriate solution and take benefits from the MDE approach.

During this translation, the nature of product model evolution within PLM system is identified. The industrial problematic may take place within one of these situations:

- In some industrial evolution problematic, product meta-model is not modified or evolved. It means that the target models are conforming to the same meta-model as the source models. This problematic can be solved only by models evolution. For example, product's component re-arranging or new product development whose models are conforming to current product meta-model. In these cases, all concepts or structures additions are compatible with existing meta-models. Let's develop an example: the information system of an enterprise manages information about a product with two composed components. Followed by some internal modifications of a production line, the enterprise wishes to modify the product components and reorganized them in the way more suitable to the production constraints. Therefore three new-composed components are created to demonstrate the decomposition of product, but the new components conform to the previous meta-model. The other case is adding a new component to product. If this new component complies with the existing meta-model then any change on meta-model is eliminated. In these cases the meta-model of the system doesn't change. Obviously, other products' offer evolutions, needing the addition of new concepts or structures, which do not exist in the current product meta-model, may not be included in this category.
- The industrial evolution problematic is about replacement of the PLM tool by a new release. In this case, the PLM system may be changed and replaced by a new PLM system. Each commercial PLM system has some pre-defined concepts that are used to define and construct the product meta-model. The replacement of PLM system will change theses concepts. The predefined concepts in the source PLM system can be slightly different from those of the new PLM system, inducing difficulties in data transfer and adaptation. As a consequence, the product meta-model entities should be redefined and the meta-model must be reconstructed in

order to conform to the new system. In some industrial PLM systems, the product meta-model also is predefined and integrated within the system. In these cases, the PLM replacement will result in the modification of product meta-model. As a general conclusion, any modification, which necessitates the change of product meta-model, may be the source of this type of problematic.

- The industrial problematic is not about the PLM tool replacement, but its adaptation to changes in organizational context. For example, integration of a new information system within the enterprise systems may require the modification of product-modeling facilities. In this case, the objective is to change the structure of data and information within different information systems in enterprise in the way that these systems become interoperable. In some cases, this integration does not necessitate any product meta-model modification for PLM system (as one of the integrated systems). Therefore the integration may be done by designing an interface between systems. The real problem in PLM tools adaptation and integration is in case of incompatibility due to different meta-data structure. Therefore, PLM tools adaptation and integration may induce product meta-model evolution. In this case, during the integration, the product meta-model in PLM system should be modified and then a model evolution between systems should be done to transfer the product models in PLM system to the new adapted PLM system.
- In some cases, the enterprise has not used a commercial PLM system to manage its product-related information. However it uses the other type of information systems to do it. Sometime, these systems are self-developed by enterprise. The deployment and integration of a new PLM system needs the evolution of product models, which are adapted and stored in previous legacy systems. In this case the previous information system is most of the time more adapted and customized to the enterprise requirements but the new PLM system is more standard and provides communication facilities with its customers, partners and suppliers. In this case, the product meta-model in new system should be defined in the way that the requirements of enterprise as well as the particularities of previous system should be preserved.

These situations show in which situations this research may find industrial applications.

3. Phase two: Identification of the equivalent Evolution Scenario

The four business situations described in previous section define four scenarios of evolution from an informatics point of view. It means that the translation of any industrial problematic around product evolution must be laid within one of these four scenarios.

The first scenario, illustrated in figure 4.3, is about the case when product meta-model is not modified or totally replaced. In these cases, source and target models conform to the same meta-model.

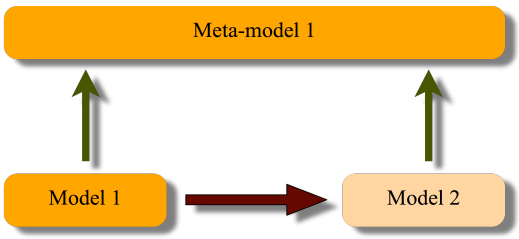


Figure 4-3, First scenario

This scenario takes place when the PLM system remains in place and only some models are evolved or changed. Another possible case is while PLM system is entirely modified or replaced, but the new PLM system relies on the same meta-models and product meta-models as the old one. Generally solution doesn't require an entire transformation process.

In the second scenario, the product meta-model in the target system is unknown. It means that final conditions are partly ambiguous and undefined. This type of situation may be favorable, since even the target meta-model can be adapted and determined in the way that is suitable for the future application of the information system. On the other hand, it may add another phase to product model evolution procedure. This type of informatics problematic is possible when an enterprise desires to change its PLM system or modify it dramatically, but there is no restriction on the future system basic meta-models. It means that the enterprise desires to create the best suitable product meta-model.

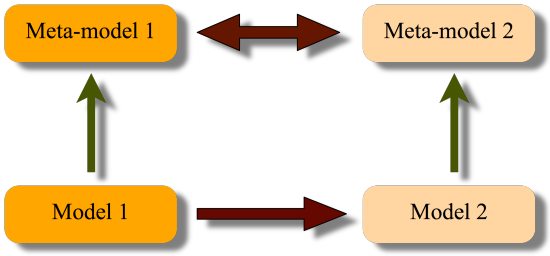


Figure 4-4, Second scenario

In order to perform this type of evolution, the target meta-model must firstly be designed and implemented. This may be completed by using DSM (Domain Specific Modeling) methods, which are used to construct or identify the concepts and grammars for a specific domain. This will be presented in the following sections. The source meta-model may be taken into consideration in the process of construct final meta-model. This implication on one hand helps to more enriched construct meta-model compared to source meta-model, and on the other hand can reduce the difficulties in finding correspondent concepts. It means that in this scenario, a proper and accurate definition of the final meta-model may improve the process of transformation and decrease the possible lack of information and adaptability between meta-models. While the product meta-model is defined, the transformation of related models are similar to the third scenario that will be presented subsequently.

The third scenario, which is the most likely, is a very typical type of model evolution and transformation. In this case, the source and final meta-models are different and predefined.

The possible industrial cases that lead to this type of evolution are all PLM system replacement in which the new system is defined. For example, an enterprise, which is joined with another one and must integrate or even replace its PLM system with its mother company. In some conditions, adapting PLM systems with the other information systems of enterprise or information systems of partners (clients, suppliers, etc.) may cause this type of transformation.

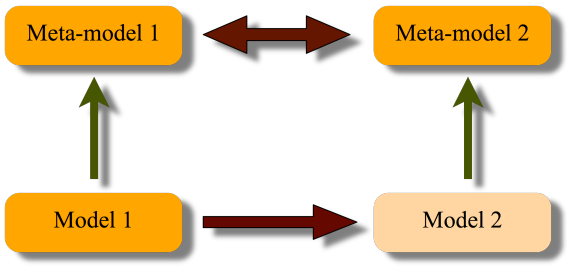


Figure 4-5, Third scenario

The procedure to precede this type of evolution will be presented in the next section, but it should be noted that in the second scenario, after identifying target meta-model, it becomes approximately similar to this scenario.

The fourth scenario is a very specific case. In the informatics viewpoint, it may never occur because, in informatics viewpoint, a model has always a meta-model. In other terms, for constructing a model, a language (meta-model) must be used. Therefore this idea of having a model without knowing its meta-model is quite unrealistic. But, it may happen occasionally in industrial context based on legacy systems. It is about the cases that the source meta-model and models are recognized, but for the final situation, only target models are known. It means that, in the industrial viewpoint, the final requirements are identified, but the corresponding meta-models to be inserted in the new system is not properly identified.

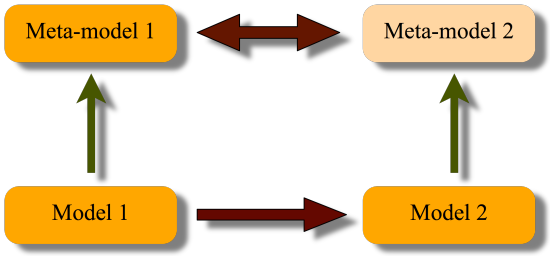


Figure 4-6, Fourth scenario

No structured and on-going method has been identified to solve this type of problematic, however, some instructions relating to meta-modeling can be found in the following sections.

Processing the first scenario is straightforward and practicable. For the second scenario, initially, the final meta-model should be defined and constructed. Afterward, the other stages for both the second and third scenarios are the same, and transformation should be executed. This is the subject of the second part of this chapter. For the fourth scenario, the corresponding meta-model should be “identified”. It means that regarding to the requirements in the target model,

the concepts of the meta-model should be found and then the whole meta-model be constructed. In other words, it's not a real transformation problematic. It's about meta-modeling.

3.1. Meta-Modeling, DSML

Meta-models can be considered as a modeling language (J. M. Favre 2004c). Some modeling languages define the concepts in a general way; they can be used for modeling all types of systems. But on the other hand, in order to preserve the semantics of a specific domain, some modeling languages are developed for specific domains. They provide not only the modeling elements to represent a system, but also the semantics and concepts of the language. Therefore they are more rich and detailed than a simple modeling language. One approach to define and develop a specific domain modeling is meta-models. In other words, the meta-models are the structured representation in DSML, Domain Specific Modeling Language. In DSML, the ontology of a context is designed and a framework for modeling the concepts is proposed. This DSML covers also the constraints and different types of rules which must be satisfied or followed during modeling process. DSML may result in a textual description of concepts and semantics of the domain. But meta-models can describe and explain these concepts in a more detailed and clear manner.

In model transformation, the meta-models of both the source and target systems should be identified before execution of the transformation. In one scenario, the third, this meta-model should be constructed by experts of the domain. In this case, if the DSML of the current domain is known, the expert can construct its meta-model via the semantics described in DMSL. But if even the DSML of the domain is not previously defined, the expert should define it at first by using the methods and procedures. In the second scenario, the input meta-model of the target system may be modified in some cases. This might happen when during the model comparison, lots of concepts or constraints in source meta-model have no correspondent or equivalent in the target meta-model. Besides, the target meta-model may guide the user when modifying based on evolutionary requirements. In both cases, DSML has an important position during models comparison.

DSML plays a crucial role in Model Driven Engineering and its related standards as MDA (OMG 2001). Lots of research has been devoted to DSML formalization and methods related to construct DSML, in software engineering (Jackson & Sztipanovits 2009) (Lin et al. 2007) as well as in industrial modeling context (C.-H. Chen et al. 2003) (Yang et al. 2008). In computer science, there is a major trend to propose semi-automated process to construct this DSML. The main idea in this approach is to collect almost all presented concepts in a context of software engineering with their relationships and associated rules, and then, by proposing an intelligent interface to users, to help them to make their choices and construct an appropriate DSML, based on their needs and limits(Santos et al. 2010).

But in industrial engineering domain, constructing DSML requires a capitalization of the expertise in the considered business domain. Several technical meetings with domain experts may be necessary. The concepts and particularities of a domain are found during these meetings and then they are formalized in UML models (T. Asikainen & T. Männistö 2009). In product model context, each enterprise has its own way to describe its product meta-model. But the list of probable concepts in previous section may be useful in order to initiate the job.

In many companies, this phase of meta-modeling is not established in a structured way. Modeling product in M0 layer may attract huge time and effort investment, but its prior phase of meta-modeling is done with insufficient attention. Lots of problems in PLM product modeling problematic could be solved by a proper meta-modeling, if performed in the early phases of PLM implementation.

One of the most important challenges in product model management is the consistency and interoperability between different viewpoints of product models. For example, product configuration as-designed, which is used in enterprise design department, should be simultaneously coherent with product configuration as-built in manufacturing department. A proper and structured meta-modeling may help to construct a meta-model, which has the ability to manage this relationship. This meta-model has all concepts needed to construct these different models of a product and guarantee their consistency.

4. Phase three: Similarity Framework

As mentioned in the previous chapter, in order to do a transformation process, final and initial meta-models should be compared and the similarity points between them identified.

The result of this comparison concludes in how much a meta-model is similar to another one. In other words, for each concept in a model or a meta-model, what is the corresponding concept in the other model or meta-model?

4.1. Model comparison

In model engineering, lots of researches, presented in the next section, have been devoted to the model comparison and model differentiation. In meta-model comparison for transformation, all of these methods and tools of model comparison can be used to compare meta-models. It means that in the domain the subject of similarities, model and meta-models make no difference. Model comparison is a process of comparing two models in what, for each modeling entity in a model, a correspondent or similar concept in the other model is determined. Moreover, model differentiation, which is about finding differences between models are a way to establish the correspondences between models. The difference between the term of model differentiation and model comparison comes from the process of model evolution. In model comparison, generally the subject is to compare two models, whereas in model differentiation the subject is to identify the differences of a model during its evolution.

Researches on model differentiation enclose a variety of domain and methods. From the most simple model differentiation of text comparison to most complicated methods of graph and structure evaluation. Researchers on model differentiation have generally taken two approaches: one approach deals more with the differentiation of models during their lifecycle (Kolovos et al. 2009) (Lin et al. 2007) (Sadeghi et al. 2009) (Rivera & Vallecillo 2008) (Y. Lin 2007). In this approach, the differences between a model in one state of its lifecycle and itself in the other state of lifecycle are studied. It means that in this approach, the influence of evolution on a single model is studied. The principal idea of the differentiation in this approach is to identify the created, deleted and updated concepts during this evolution, so one of the elements of model differentiation is to find when the comparison must take place.

The other approach is about comparing two different models (A. Cicchetti 2008) (A. Cicchetti et al. 2007) (M. D. Del Fabro & P. Valduriez 2009). In this approach, which is much newer even in software engineering, two different models are compared in order to find their similarities or variations. Common methods are textual comparison, which is based on XML schemas of models (Deridder et al. 2009) or using graph theory to compare more accurately. Some of model information and knowledge is about its structure and the way that concepts are related. Some methods (A. Cicchetti et al. 2008) studied that aspect.

4.2. Proposed framework of similarity framework

In the present research the procedure of model comparison relies on syntax comparison as well as on semantic comparison for product models. This kind of comparison is useful in this research, because of the importance of semantics in industrial information system modeling. The modeling language, which is a representation method of these concepts, can play an important role in this comparison. A language is a way that defines what semantics are needed and how they are organized between concepts. As discussed earlier, UML language has been chosen as the reference standard modeling language because of its large dissemination in industrial domains modeling. Moreover; in lots of researches and practices on product modeling, UML is used as a modeling language. The modeling language has an important and fundamental role in model transformation, particularly in model similarity framework, since the entities of models that should be compared, are facing the entities described in modeling language meta-models. For example, in UML language, the entities or modeling concepts are described in MOF ; thus, every UML model uses entities and concepts characterized in UML meta-models, and with a description and constraints predefined in MOF. As a result, any product meta-model entity should be "conform" to this meta-model, and using these concepts like attributes and associations. In other words, the semantics of model concepts should be defined by language proposed entities (attributes, classes, associations etc in UML), and organized in the way described and allowed by a modeling language. In conclusion, for proposing a model transformation, and especially model comparison, modeling language has a central role.

Besides, defining the similarities between models relies on modeling language. This semantics, which is described in details in the following, may be enriched or modified, based on the semantics of a specific domain, but its granularity remains intact. It means that the overall process of similarity identification is based on the domain language and requirements related to a system, but the entities to be compared are represented in the modeling language. The granularity of semantic definition specifies that for a certain concept, which element in modeling language should be chosen. For example, in a very simple case, the concept of product material may be specified in a model by a class or an attribute in a product model. Obviously it depends on the desire and need of the model designer, but it can be important in model comparison and subsequently model transformation. On the other hand, the granularity depends on the modeling language facilities.

Since they are both expressed in the same modeling language, namely UML, model comparison is not to find the correspondences between modeling concepts and simple conformity or agreement between models. As mentioned above, it is about to find their semantic

and syntactic similarities. Therefore, first of all, the concepts and entities that are the objects of comparison should be defined as well as their granularity.

The other important subject to be determined is to construct a criteria framework for comparing models and specify the similarity. These criteria are used as a mean to find the similarities between model concepts. Each pair of concepts (one from source model and the other from target model) should be compared by these criteria. They consist of syntactic based criteria as well as semantic based ones. Syntactic based criteria analyze the similarity between two concepts from form and structure of concept, but semantic-based criteria try to find the similarities related to characteristics of concepts. They are enriched according to the domain characteristics. Besides, there are criteria related to the structure of models or concepts. In other terms, the domain specifications and business related semantics have more influence on the similarity criteria, but the modeling language shapes the granularity of models and entities, which must be compared according to these criteria since the modeling language defines the blocks in a model to be compared.

And in the last but not least, other relating subject is the calculation framework of similarity. In this framework, the similarity between the elements of model is calculated by assigning a degree to each element for a specific criterion. These degrees should be assigned based on expertise and requirements of studied domain. Then, the similarity of each concept is calculated in function of degrees and the suitable weight allocated to each criterion.

As an example, let's define two elements of **"a"** from the source meta-model and **"b"** from the target meta-model. For each criterion of comparison, based on the domain expertise, a number that represents the degree of similarity is assigned to this pair of elements. In order to compute the total similarity between these elements, the similarity degree of all of the criteria should be added. But during this addition, the weight of each criterion must be taken into account, because each criterion has its own importance on similarity. This formulation will be presented later. This calculation process is shown in figure 4.7

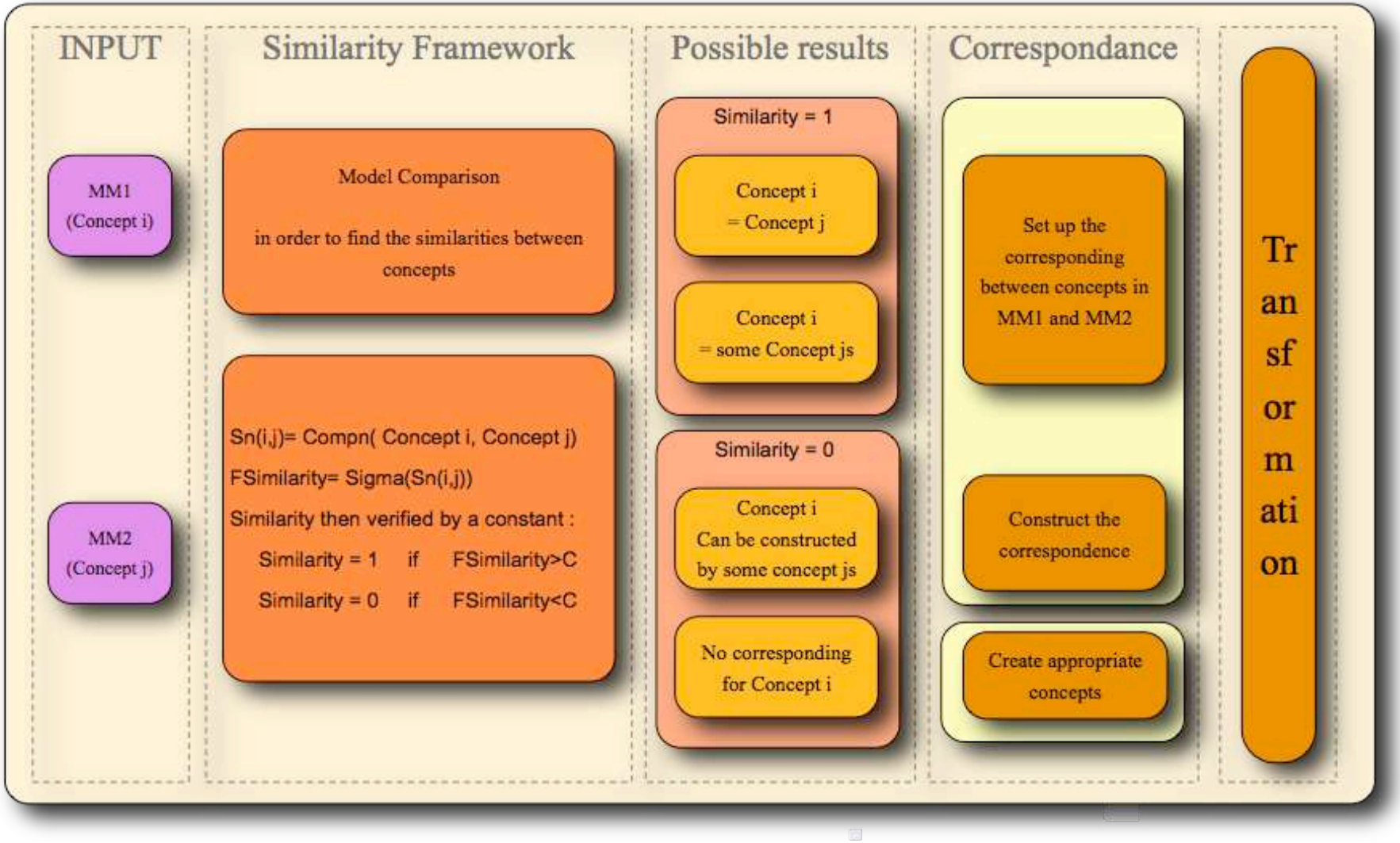


Figure 4-7, Similarity calculation framework

In this research, the model entities, based on the modeling language, are:

- Attributes
- Classes (as a set of attributes)
- Relationships:
    - Relation inter-class (relation between the concepts existing in a same class, like between the attributes of a class. This type of relation is not presented explicitly in UML classification. But when we present two concepts in a same class, it means that there is the "inter-class relationship" between them)
    - Relation intra-class (association between two or more classes.)
        - Association
        - Composition
        - Aggregation
        - Heritage

These are the entities provided by modeling language, but their semantics may be enriched by information related to usage domain. In other words, different types of associations may be constructed to represents different types of association between elements. Therefore, if these model elements without any other associated information related to domain are compared, then similarities between models, which are independent from the domain of these models, are identified. This type of similarity is called syntactic. Syntactic similarity is about to find the similarities syntactically between two entities. Criteria related to a syntactical similarity are name, type and format of entity. Detailed syntactical criteria which are different for each type of entity are presented below:

Attribute:

In model comparison, each attribute of a model should be compared with the attributes of the other model in order to find its more similar correspondent. The problem arises when a class has an attribute as a correspondent in the other model, which was introduced earlier as granularity-related problem. In our model comparison, the comparison is done between all model elements. Therefore, by default, the proposed similarity framework checks the similarity between classes and attributes.

It should be noted that the attributes constructing a class, possess the semantics of its associated class as well. It means that in model comparison, attributes are not compared independently from their classes. For example the attribute "ID Number" of the class "Tool" has a semantic of ID number of tool. Therefore, during comparison of attributes, their containing class is compared. In order to fulfill this condition, the name of class is concatenated to the attribute name, like: "Tool.ID-Number".

For an attribute, its name is amongst the criteria of similarity. Two attributes from these two models that have a same name may be correspondents. It depends also on the other criteria. But the similarity of the names is not restricted to a perfect accordance of their names. According to the domain and enterprise requirements and terminology, a glossary of each

concept must be constructed. In this glossary, all the synonyms of used terms are collected, for example, for "name", other terms such as "denomination", "label", "appellation", etc. There is no difference between the case where two elements have an identical name and the case in which both of them have the synonyms. If the names of attributes accompanied by their class name are identical, their score is the highest, 1. If their names of attributes are synonym but their class names are different, their score is 0.5 and if the name of attributes is not similar at all, their score is 0.

The second criteria, is the format of attributes. Attributes with the same formats, such as Number, String, Float, etc. are more likely similar. As the scoring procedure, if both of them have a same format, the score is 1 and in the other case, it will be 0. But other criterion, which may influence in comparison of attributes, is their form. Form here means the way that an attribute is constructed. For example in some companies, for their ID-Number, they use a special form. In comparing some values of attribute, it should pay attention to these types of possible dissimilarity, which comes from the enterprise particularities.

Class:

Classes are considered as a set of attributes. Then a class has the semantics of their attributes and moreover it has the semantics related to this set, like its name. So the criteria related to classes are the name of the class and the number of attributes along with their similarities. It means that two classes, which have the same attributes and similarities, are correspondent.

The most important difficulty of class-related similarity is a class with the attributes spread in different classes in the other model. For example, class A in source model has two attributes, a. and b. With the help of model comparison, the correspondent of a and b are found, they are c and d. But c and d are not in the same class, c is in class C and d is in class D. the problem now is the correspondent of class A in the target meta-model. In this case, the correspondent of this class in the other model is the group of these classes. It will be discussed in more detail in the following sections.

Association and Association End:

Associations are the most complicated elements of the models to compare. Each association has two association-ends. So for each association in a model, three different entities must be evaluated. In order to find the correspondent of an association in the other model all of these three entities should be compared. In the next section, during the discussion of different possible types of similarity for each entity, the probable situations of association similarity will be clarified, but it should be noted that an accurate description of the association semantics will be crucial in association comparison. Here the basic semantics of association entities are introduced and then they will be enriched.

According to UML semantics, each association is identified by its name. But the properties of association-ends are more complicated. For each association-end, role names, ownership indicators, multiplicity, direction, and the type of association should be identified. The type of association is presented in one of its association ends. An association may be a simple association, a composition or aggregation, or a specialization/generalization. The

associated semantics related to these types demonstrated in association-ends, because their effects are on each end of association and the classes related to them. All of these types have a single direction, except a simple type of association, which has a double direction. These directions are shown in association ends. In the case of an association class, it is considered as a single class with two associations.

These are semantics related to UML language, however, there are some other particularities related to a specific domain of modeling, here product configuration model. These semantics are described in the next section.

All the descriptions presented here are designed with UML modeling language in accordance to UML meta-model, "MOF". MOF, Meta-Object Facility, is a standard designed and updated by OMG. It is a meta-model of all UML diagrams (Gardner & Griffin 2003)(OMG 2005). It provides a detailed description of all elements present in all types of diagrams and their interaction/relationships. A simple representation of MOF, which more focuses on class diagram, is illustrated in third chapter, figure 3.3.

As mentioned before and based on the MDE principles, in modeling framework for product configuration, product models must conform to a product meta-model. This product meta-model, which defines how to build product models (in terms of concepts and relationships between them), is constructed using UML. It means that it conforms to MOF. In other words, the concepts of product meta-model are a set of enriched concepts of MOF, dedicated to the considered domain (here product configuration within PLM).

The concepts listed here are the most important elements of UML models in product configuration domain. This list does not cover all product configuration concepts but it contains the most important and usual ones according to PLM systems features. This list may be extended or restricted based on the particularity of the industrial case. The aim of this list is to set up an appropriate framework, which will provide a base to compare more accurately meta-model concepts. This list is presented in the following:

- Name: a text expression, meaning the name of the concept (item / product / document, etc)
- Reference / ID: a unique combination of text and/or numbers, which accurately identifies a concept in a unique manner
- Dimension: a set of numbers showing the volume of a part. In Cartesian coordination system, it includes three numbers, with a specific scale.
- Weight: a number representing the weight with a scale.
- Material: name of material used to make a part, recorded in the system.
- Elementary part: this means that the considered part cannot be splitted into other parts.
- Composed part: this means that the part is composed of at least two other parts (elementary or composed).
- Specific part: it is a type of parts that is related to a generic part and at least has one valued parameter in addition to its generic associated part.
- Generic part: it is a type of parts that has at least one non-valued parameter.
- Exemplary part: it is a type of parts, that all of its parameters are valued. This part is manufactured in enterprise.

- Abstract part: All parameters of this type of parts are not valued. This part may be regarded as a model or template of all other types of parts.
- External document: this means that the considered document is created outside the PLM system (such as a CAD file).
- Internal document: this means that the considered document is created within the PLM system.
- State: a text, which can only have certain values (depending on enterprise's typology to distinguish the different states of a concept). A state expresses the maturity of information related to the considered concept
- Version: a number or a combination of text and number, which must change accordingly to each modification, so to distinguish the various modifications
- Tools: text, which defines the tool (as a resource) that are needed to construct the associated part.
- Supplier: text, designating the name of a company who has furnished the considered concept.
- Date of effectiveness: It means the date of validity of the information associated to the concept
- Production situation: a text can be one of (Manufactured / purchased / subcontracted (co-designed)) which defines how the company purchases a product.
- Author: text, creator's name of part or document.
- -User: text, name of a system's user, who has the access to technical data of a product, a part or documents. It may have different access rights.

In PLM systems, these concepts are used in order to product models describing products designed and fabricated in enterprise.

4.3. Method for comparing two models

Product meta-model from each system can be compared. In this process, each meta-model concept is compared with concepts of the other meta-model in order to find the most similar concept. The general process is executed as follow:

One concept, such as an attribute is considered, and then compared to all attributes of the other meta-model according to different criteria. For each criterion, a number that specifies the degree of similarity is allocated. Then the sum of these degrees multiplied by the weight of the associated criterion provides the total similarity between these two attributes. Following formula is used to calculate this similarity:

For each concept i from MM1 (source meta-model) and concept j from MM2 (target meta-model), according to each criterion n, a degree of similarity is calculated. If two concepts according to the criterion are identical, then degree of similarity for this criterion, Sn(I,j) is 1. If there is not any similarity, Sn(i,j) is equal to zero. Sn(i,j) may be a number between 0 and 1, if they are relatively similar. This value is allocated regarding to the expertise of the case under discussing. It will be discussed in the following paragraphs.

For each criterion, weight or Wn shows the importance of this criterion relatively to other criteria. This is a normalized number between 1 and 10. Assuming that there is k criteria, S(i,j) which introduce the overall similarity between concept i and j are calculated as :

$$S(i,j)= \sum_k (Wn*Sn(i,j))$$

Criteria are different according to compared concepts. If concepts are attributes, classes or associations, the criteria are the same ones as those presented in the previous section. For example, if they are attributes, the criteria are name, format and form. It should be noted that based on discussion in previous section about existing concepts in product configuration meta-models, this comparison here between attributes (and other types of concepts) are more oriented and adopted to this application.

The procedure of allocating the degree and weight for criteria may be semi-automatics. It means that this procedure is not done only by the system in an automatic manner, but the user of this system is allowed to modify and customize these values.

If overall degree of similarity, S(i,j), for i and j lies in interval [N,C], then they are the correspondent concepts. N and C are identified and allocated according to the case of study but as a proposition, they may be calculated as C = 10 and N= 5. If the similarity for i and j is equal to 1 for all of the criteria, which means that i and j are correspondent based on all of them, then S(i,j) is equal to 10. (Weights are between 1 and 10). Therefore the best and maximum number for S(i,j) for the correspondence is 10. It's obvious that the values of N and C can be modified due to user's intention.

For example, the attribute "Part.Name" from MM1 is compared to all attributes of MM2 (j), and S(Part.Name, j) is calculated for each j in MM2. As an example here, let's imagine that in MM2, there are three attributes: "Component.Identification", "Project.Name" and "Component.code". Part.Name is a text, which describes the name of a part. Component.Name determines the name of component, Component.Code shows its codification and Project.Name defines a project's name. A project may contain lots of components. The process of calculation compares different properties of these entities with the properties of Part.Name. For an attribute these properties or criteria are the name, form and format. These are the results of comparison:

- Name (Part.Name, Component.Identification) = 1 => part and component as well as Name and Identification may be considered as the equivalents.
- Form (Part.Name, Component.Identification) = 1 => both of them are text
- Format (Part.Name, Component.Identification) = 1 => both of them doesn't follow a special rule of naming.
- Name (Part.Name, Project.Name) = 0.5 => both them explains the name but one of them is about the part and the other about project.
- Form (Part.Name, Project.Name) = 1 => both of them are the text.
- Format (Part.Name, Project.Name) = 1 => both of them doesn't follow a special rule of naming.
- Name (Part.Name, Component.Code) = 0.5 => both of them contains the equivalents.

- Form (Part.Name, Component.Code) = 0 => it's clear
- Format (Part.Name, Component.Code) = 0 => codification follows some specific rules.

Therefore the S are calculated as:

- S (Part.Name, Component.Identification)= 1*3,33+1*3,33+1*3,33= 10
- S (Part.Name, Project.Name)= 0,5*3,33+1*3,33+1*3,33= 8,32
- S (Part.Name, Component.Code) = 0,5*3,33+0*3,33+0*3,33= 1,66

It shows that "Part.Name" and "Component.Identification" are similar. Although this was a very simple example, it shows the steps of proposed procedure.

The problems, which may arise in the comparison process, are:

- For one concept, its correspondent concept is of different nature. This happens more between classes or attributes. For example, one attribute correspondent in MM2 may be a class. Finding this type of correspondence is not so obvious, as the criteria are different for comparison of a class and an attribute. In this case, the result of comparison returns nothing for the concerned class or attributes. It should be done by user's interaction. The user may define this correspondence himself.
- For one concept, more than one correspondent concept is found. For example in MM1, only the "user" for a part is identified but in MM2, the user has been divided in two concepts of "designer" and "client who command it". In this case, both of them are the correspondents of "user" in MM1. Therefore both of them are user of this part.

Besides exceptions described above, four types of result may be obtained as a similarity calculation output:

1- For a concept i, a similar concept j is found

2- For a concept i, a similar concept j is not found, but the information related to concept i is distributed between several concepts in the other meta-model.

3- For a concept i, no similar concept is found, but the information necessary to set up similarity link may be extracted from the system.

4- For a concept i, there is no correspondent concept and there is no way to extract information from anywhere to set up similarity link.

The first possible situation is quite simple. Two concepts are correspondent. Executing a query may also treat the second situation. This query, which connects the different concepts in MM1 to different concepts in MM2, specifies which part of semantic in one MM1 concept may be connected to which part of semantic in MM2 concepts. It means that this query, via the assignment of correspondent concept pairs, such as (i,j), specifies the similar semantics in two meta-models.

The third situation is more complicated. In this case, necessary information to build up a concept in one meta-model must be extracted from information distributed in several concepts of the other meta-model. In order to solve this type of similarity, a standard process cannot be applied, but according to particularities of different cases of study, an adapted solution may be

proposed. Distribution of necessary information is not only limited to a simple break down of information in several concepts but also it may even concern a situation where, for identifying a similar concept between meta-models, industrial data in models level are used. It means that information existing in models helps to determine necessary information to integrate in the meta-model level of target system. This similarity is not entirely treated in computer science literature. However, (Kurtev & K. G. van den Berg 2004) has counted this type of similarity. This is illustrated in figure 4.8. Next chapter discusses more precisely one example of this type of similarity.

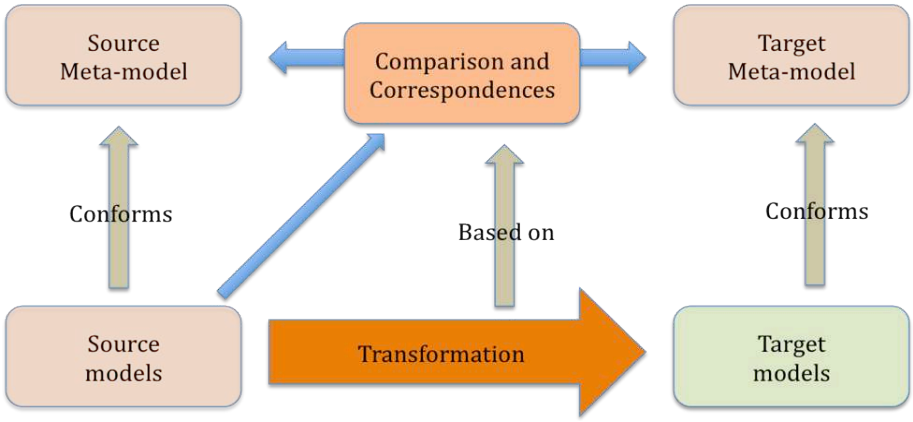


Figure 4-8, Inter level Transformation

Although these propositions do not follow a specific pattern or algorithm, for some general cases, guidelines can be used. Next chapter illustrates a guideline for a specific industrial case study, which focuses on generic product modeling.

In the fourth situation, some pieces of information are not extractable from the other meta-model. No means exist to find a correspondent for some of the concepts. Generally there is no general solution for this type of situation. If a concept in the source meta-model has no correspondent in the target meta-model, it means that some pieces of information during the evolution will probably be lost. These pieces of information are not transmissible in the new system models. If the target system can be changed (the case of installing new PLM tool or creating new system) the new concepts may be added to the target meta-model in order to store these pieces of information. But if the target system is imposed (for example in case of an information system which is going to be integrated into another enterprise system) these pieces of information in product models will be lost. On the other hand, if a concept of the target meta-model has no correspondent in the source meta-model, then it means that the new system will have some non-valued/non-instantiated concepts or properties.

4.4. Constraints and rules

The other components of the models, which should be transformed into the new system, are the constraints and rules. On one hand, constraints define additional limits to modeling; they exist in both meta-model and model level. Rules and constraints in the meta-model level determine the perimeters, which should be respected during the model construction and execution. They specify the relationships between the entities or concepts of the meta-models.

But rules in models govern the concepts in models and their relationships, and generally come from the modeling language and the domain expertise. They are important parts of models and should be transformed during model transformation. Before describing a general list of possible constraints and rules in product configuration context, it should be noted that the boundary between a rule and a constraint is pretty ambiguous. In other words, rules may be reformulated and represented as constraints. A constraint can also be described by a set of rules that impose it. For example, the rule that specifies the relationship between two types of entities in the meta-model may be described as a constraint. In the simple meta-model for a product like car, they may exist a mobile part and fix part. Mobile parts in a car like wheel in model are the instance of the entity "mobile part" in meta-model of product. A rule in model of car may be like that: "all of the mobile parts should satisfy tests of rotary." It means that in any model of product, an instance of test of rotary may be connected to each instance of mobile part. During the model transformation, this rule should be transformed and applied to any instance of the correspondent of mobile part in target meta-model as shown in figure 4.9

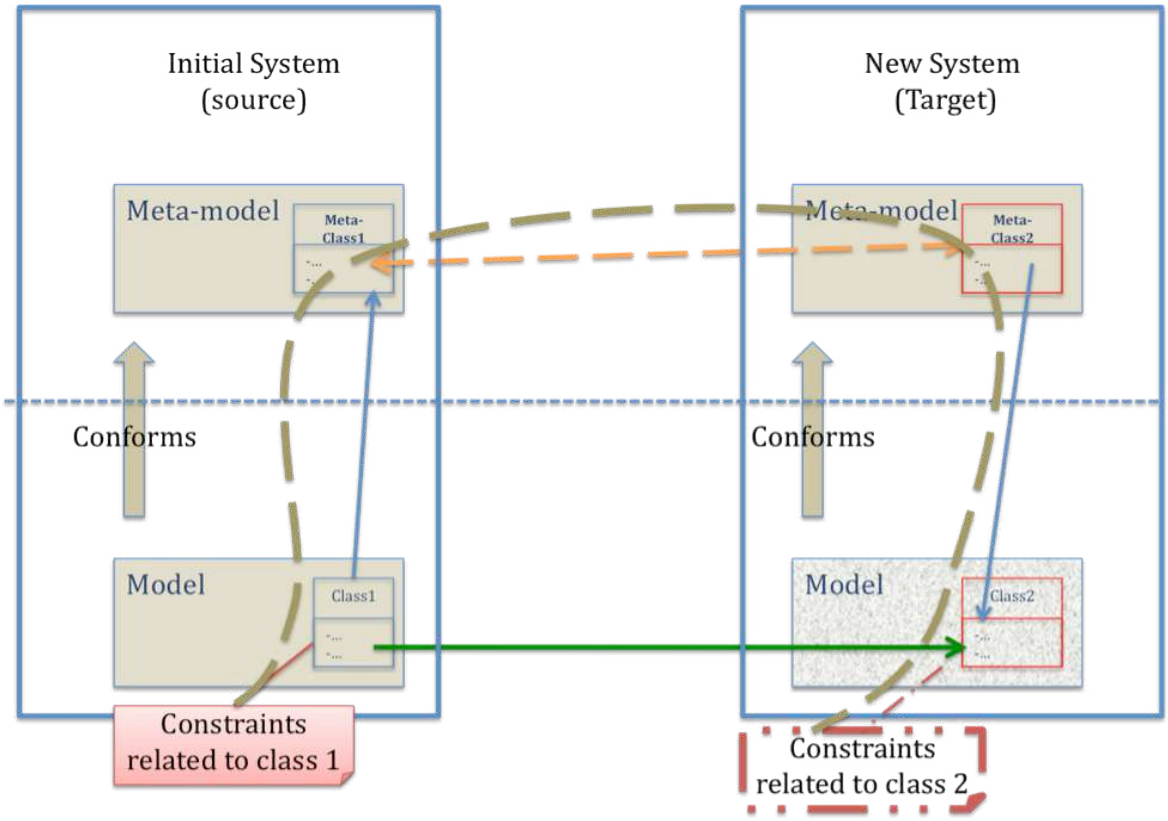


Figure 4-9, Transformation of constraints

Constraints may apply on product model within only one phase of its lifecycle, or they govern the product model during some phases of it. Static constraints are due to a specific state of product lifecycle and define the limits of product model in this phase. Dynamic constraints determine the restriction of product model evolution during the different phases of product lifecycle. In other words, dynamic constraints applied mainly during the transition of the product model from one phase to another. As an example, the rules that specify the migration of

a model from its design phase to manufacturing are dynamic rules and can be represented by dynamic constraints. The following list proposes some kinds of constraints in product modeling:

- Static Constraints
  - Product feasibility: deals with the constraints that are necessitated by developer for a specific product line, which defines the restriction of product models components relationship and their relationship. Product feasibility constraints consist of instantiation rules as well as product constraints. Instantiation rules define how an exemplary product (real product) is created from its design model, by taking into account its feasibility. For example, even if in the design model, a car may have different types of tyre, in a real car, it can have four tyres of only one type.
  - Special constraints based on models: are related to a specific context. In different cases, the modeling framework may impose the different types of constraints. These constraints are defined and applied by the expert of modeling domain. For example, in the generic modeling framework, each specific part must be associated with a generic part, or each generic part must have some non-valued attributes. The domain expert defines these constraints during the process of modeling framework construction for each specific domain. This framework is called DSML, Domain Specific Modeling Language. This subject is discussed at the end of this chapter.
  - Conformation between model and its meta-model: this type of constraints guarantees coherence of the models with their associated meta-models. In general, the modeling framework automatically controls this type of constraints by validation the conformation model and meta-model. Moreover, the construction of models in accordance to meta-models, by its nature, ensures the conformation.
- Dynamic Constraints
  - Dynamic constraints applied all along the lifecycle of the models. The evolution of product models during different development phases is controlled by dynamic constraints. In other words, they manage the coherence between product model and data during different phases (design, manufacturing, etc.)
  - Dynamic constraints or rules also define how a product model may be modified. The design of the information system predicts the model evolutions in one phase of product lifecycle and specifies the rules and constraints to manage these evolutions in the way that the system can handle them easily.

4.4.1. Constraint/rule Transformation

Constraints and rules are one of the parts of product models. They are written generally in OCL forms and specify the relationships between concepts; therefore a constraint may be considered as a phrase contains one (or more) concept(s) of model or meta-model with specific information about this concept. For example, the constraint “the attribute A of class B should have an even value” specifies a property for a specific concept. Transformation of this constraint

should be followed by a transformation of the concerning concept. When the correspondent of concept B(A) is found, the constraint should be transformed. Therefore in order to transform a constraint, the correspondent of existing concepts in it should be "tracked".

Managing the transformation of constraints is treated after the transformation of concepts. The transformation of constraints is based on using model transformation results and the tractability of concepts during transformation. The constraints can be written in OCL (Object Constraint Language). After the execution of transformation process, the correspondent concepts are found. The new constraints in the new system are rewritten, as only a textual query by using the correspondent concepts. Transformation of constraints, as a satellite objects of meta-models are studied in (B. Amar et al. 2008).

This mechanism of constraint transformation can treat all types of constraints. The constraints which are related to a modeling languages as well as ones that are concerned the business and domain of modeling are writable with the OCL languages and place in this form of concept/property. For example, a constraint may define that the composed part should be composed of at least two parts. This specific rule links the concepts (part and composed part) with the property (at least composed of two). For example, the transformation of concepts results as (part corresponds to external design project) and (composed part corresponds to internal design project). In the new organization, the enterprise purchases elementary parts and only assembles them. Therefore, the transformation of constraint may be rewritten of the phrase with the new concepts: "an internal design project must include at least two external design projects". Other types of constraints are treated as this example and transformed by using tractability of transformed concepts of the models.

4.5. Conclusion

Similarity, the framework is a key component of model transformation. As mentioned before, it depends on the context of modeling. This proposition has a particularity vis-à-vis the other transformation methods, which are developed in software engineering. It has been shown that in research on software engineering, the meta-models are not so different, or in some cases, transformation is between the languages and therefore the concepts in meta-models represent only languages concepts with a standard ontology. Consequently, finding similarity between them is not a very complicated.

But in PLM product modeling, the situation is different. Product modeling does not follow a general and widely accepted framework. Therefore product modelers use their own-defined concepts, rules and semantics to design a model. This complicates the model comparison because there is no common semantics or standards with which the entities of mode may be compared. In other words, there are different DSML which are not compatible. The only standards defined in product models are only related to data exchange between systems, such as STEP. But very structured standard framework for product modeling, which is applied in PLM systems, does not exist, and the different enterprises use their own framework, adapted to their requirements and particularities.

On the other hand, PLM systems are complex systems. The product models within them consist of several entities with their relationship and complicated rules and constraints. These

entities cover a large number of concepts in models. Therefore model comparison in PLM product models is a very difficult task, and its results must be revised and validated regularly by users and domain experts in order to guarantee its conformity with a context.

5. Phase four: Transformation and implementation

When the model comparison is done and its results are validated by the domain expert, the transformation becomes similar to the process described in third chapter. It starts when the product meta-model similarity results are known. Based on similarity identification, the concepts in the target model can be created from its meta-model. The process is described as follows:

- A concept in Model 1 is chosen.
- Its meta-concept in Meta-Model 1 is identified.
- The correspondent of this meta-concept in Meta-model 2 is found.
- This meta-concept in Meta-model 2 then must be instanced in order to create the corresponding concept in Model 2.

This process should be iterated for all existing concepts in the source model. As shown in the previous chapter, these steps are the general steps of model transformation. Transformation of constraints and rules are done after the transformation of other concepts in the model.

But the difficulties arrived for constraints that exist in meta-models level. They contain the knowledge about the modeling domain and language. During the transformation, these constraints also should be transformed to the target meta-model. But the constraints should be added to the target meta-model, in order to capture and preserve the knowledge within the source meta-model. This transformation can be performed by using the correspondence relationship. As described for tractability, the OCL-represented constraints of source models are rewritten with the new concepts of target models, which are the correspondents. For constraints in meta-model, the process is the same. The constraints will be rewritten by using the correspondent concepts in the target meta-model.

To our knowledge, no commercial or academic tool exists that can execute the model transformation fully-automatically, due to the complexity of model comparison. The number and the performance of these software increase during these years. Some of them like UML-QTV are developed to analyze models and translate them between languages or generate executable code. Nevertheless, some tools have been developed in order to execute the transformation. Developed by INRIA of Nantes, ATL, for Atlas Transformation Language, (Frédéric Jouault et al. 2008) is used widely and became a standard tool for model transformation. It is integrated within Eclipse environment and is known as the best platform of model transformation. ATL with its own grammar provides facilities to create models, meta-models and to create an ATL file, which stores the correspondence relationships between concepts in each meta-model. This tool and its application will be discussed in chapter 6.

6. Conclusion

This chapter has proposed an original procedure for executing product model evolution within PLM systems. This procedure starts with problematic translation that converts industrial problem to an informatics equivalent. This informatics problematic should be mapped to one of the four scenarios. According to these scenarios, the solution may vary but all of them take benefits from the similarity framework. The similarity framework helps users to find the differences and similarities between meta-models and set up the correspondence relationship. Different possibilities for these correspondences may occur, that are the input data for transformation execution.

This proposition is a complete procedure for model evolution in product configuration management. It covers all phases of model evolution. The previous research on model evolution, presented in chapters 3 and 4 is more focalized on each phase separately and more commonly on the transformation process itself. Moreover, the studied models belong to informatics and software engineering based on languages of modeling and implementation, platforms etc. On the other hand, this proposition studies an industrial problematic by using computer science approaches, namely product configuration management. Researches presented in the second chapter that are related to this problematic do not have an entire and comprehensive sight on the whole evolution phases. In addition, this proposition aims to benefit from new approaches in modeling and informatics system design. It introduces these concepts and methods in industrial product configuration management in order to solve evolution-related problematic. This procedure will be tested in the next chapter for an industrial problematic.

Chapter 5 An Industrial Need of Evolution, Generic Configuration

Chapter four is devoted to elaborating the general description of the proposed framework for product model evolution within PLM systems. But as mentioned earlier, this framework is hugely dependent on the particularities of the studied industrial problematic. In other words, its main phases should be customized for each case of study. In this chapter, we illustrate this framework for a particular industrial problematic in PLM systems. It is the "generic product configuration for an industrial manufacturer of culinary products. Indeed, this chapter defines and explains a special methodology for construction of "generic" configuration of product and integration with PLM systems of an enterprise by using of procedure described in previous chapter.

The studied problematic deals with the migration form a PLM system that can't manage the genericity of product lines to a PLM system that is capable to deal with this genericity. In other words, the existing system contains only models and meta-models to manage "specific" products. However, the target system is aimed to manage "generic" products. These terms will be discussed in detail in this chapter. In the studied problematic, the new target system is supposed to not be pre-defined and the process of information extraction for constructing needed concepts of target meta-model will be introduced.

This chapter starts by defining the terminology of "generic" /"specific" product configuration and the importance of this subject in the industries' information system. The particularities related to "generic" configuration and its advantages then will be presented. Our proposed framework for the migration will be customized and presented in details, illustrated by a particular example. It should be noted that this particular problematic corresponds to the second scenario of evolution, as described in chapter 4. The target meta-model should be constructed for this case. As mentioned in the previous chapter, if during meta-modeling, the considerations related to transformation and existing modeling framework (source meta-

models and models) are predicted, the process of transformation may be executed more properly without lost of source models' information during the migration. This studied example is a particular case in model transformation. In this case, the information in model level is used to determine correspondence relationship. This particularity will be discussed later in this chapter.

1. "Generic" Modeling

1.1. Product Generic Representation, an industrial need

Implementation of a PLM system within an enterprise is an opportunity to improve its information management and rationalize its document structure, since it preliminary requires structuring plenty of product information. Product information to be structured in a PLM is essentially the documents and the configuration (Rangan et al. 2005)(Brière-Côté et al. 2010).

Analyzing some industrial cases of PLM deployment shows that the management of product configuration limits, in the most of cases, to construct the configuration of each commercialized product, independently from configuration of other similar products in the enterprise. Therefore the amount of information to be managed is relatively huge. This manner of configuration management can generate some difficulties concerning to the creation and possible future modification of configurations. Examples of these difficulties are:

- Managing only the "specific" configuration increases the information and data in the system. Similar products contain some similar data, which can be factorized and centralized in "generic" configuration. As a result, lots of information should enter, stock and managed within the system.
- These huge amounts of technical data are sometimes linked, related or inter-dependent. Any modification on some of them can influence on the others and as the consequence of each changing in the configuration of a single "specific" product, the configuration of some of other products may needs the revision. Therefore, managing the evolution of product models in these systems is difficult and time consuming.
- Enterprise doesn't have a global image of its products. It means that all products, have their own configurations, and any general representation of them in information system doesn't exist.

Some commercialized products manufactured by the enterprise belong to a family of product, or a product line. Therefore, Probably, they have some similar and common properties. This commonality is not usually taken into account, with and organized and structured procedure, in industrial practices. As a result, each product configuration is created independently from the configuration of other similar product. This procedure, as mentioned before, makes the system so huge and difficult to manage, mostly during its modification.

So, in order to avoid this problem, a suitable solution consists of a "generic" representation of products belonging to a product line, which factorizes the common

information of them, and create one “generic” product configuration, instead of creating separately a “specific” configuration for each product (Männistö et al. 2001).

In industry, there are several business viewpoints, like design, manufacturing, supply chain, etc, and each of them represents a product structure and configuration via its requirements. Besides, the “specific” product configuration is created based on two more common viewpoints, design or fabrication(Jiao & Tseng 1999). Therefore, at least two types of configuration may be managed in PLM systems, “as-designed” and “as-built”. These two types of “specific” configuration can correspond to two types of “generic” configuration. The “generic” configuration “as-designed” is more known because the “generic” configuration usually created by designers and used for design purposes.

With the help of “generic” configuration, the amount of information to be stocked and treated in the system decreases since all common knowledge are transferred to the “generic” level and saved only one time. Moreover the evolution of this system is much easier, because a modification in the “generic” product configuration will be automatically propagated in its entire specialized “specific” product configuration.

1.2. Levels of product configuration

To effectively manage the information about the product, the product configuration can be managed at three levels of abstraction(L. Gzara 2000):

- Exemplary Level: it is the most concrete level of the configuration, which is the decomposition of a given physical product, manufactured and presented to a client. The evolution of this configuration is used to manage the different phases of product lifecycle "as-designed," "as-built", "as-maintained".

- Specific level: it is a more abstract level than the previous level, which represents the decomposition of a set of products of the same type called "virtual product", describing the structure common to all copies made by the same model (same configuration)

- Generic level: the most abstract level that represents the decomposition of an entire line or range of “specific” products, including options and alternatives.

In each of this level, the evolution of product model signifies a different concept and requires different procedures. This is shown in figure 5.1.

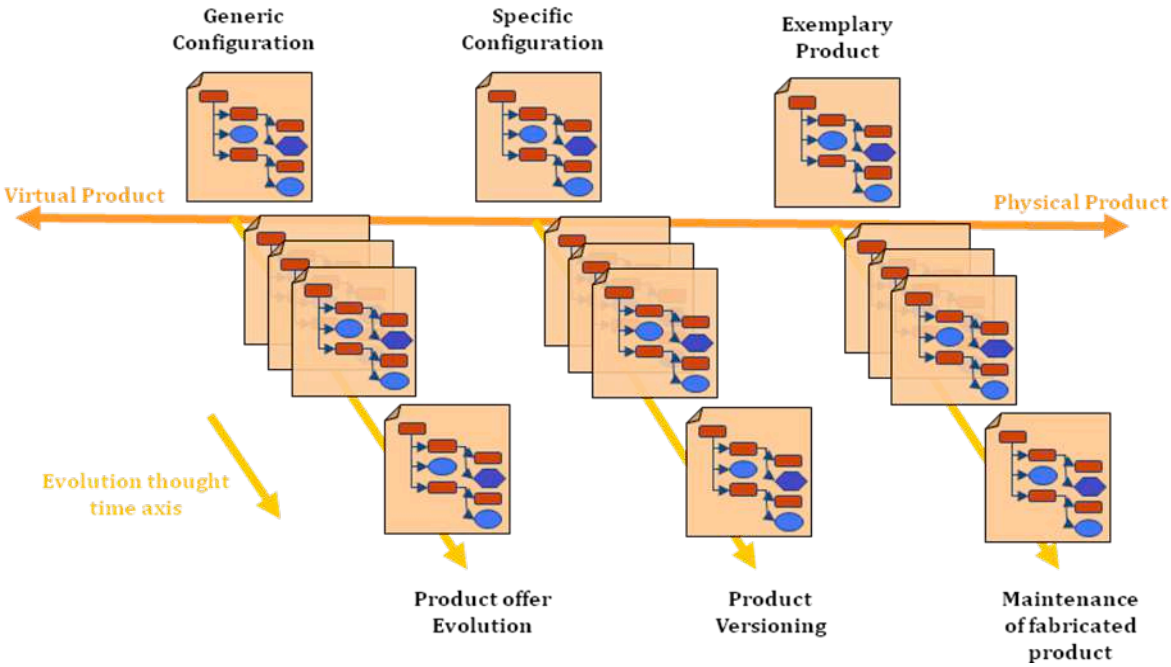


Figure 5-1, Evolution and different levels of abstraction

1.3. Properties of "generic" modeling frameworks

Generic configuration has been proposed with the aim to regroup and factorize all of common properties of the products belonging to product line. So it contains all options and variants of these products(Gzara et al. 2003). Here some important particularities of "generic" configuration are presented:

- Abstraction:

  All the objects (document, components, etc.) in the "generic" product configuration are abstracted. It means that they can't be represented physically with real objects. It's a model for a set of real products or components. The "generic" configuration should cover all the variants of a product family. So it should have several parameters without values. Going from "generic" configuration to "specific" one is done by allocating values to these parameters.

- Genericity:

  Generic configuration is a representation of a set of products (a range of products). It means that it contains several parts or components with possible variants (i.e. color) and optional items (GPS on bicycle). The attributes attached to those parts will be valued during the transition to "specific" or exemplary configuration. As stated previously, the genericity of this configuration is intended to factor out the common information between the various components of exemplary products. This factorization differentiates data structure from a simple configuration management product. It should be noted that this concept that should be kept in product model is often ignored during design and implementation of product model.

- Heritage:

Within the product configuration, the specialized objects (variant objects that are assigned to a "generic" object) inherit the properties of their "generic" object. It means that if a decomposition schema is defined for a "generic" composed object, all of the objects, which are the specialization of this "generic" object, obey this decomposition schema. This is a benefit of using object-oriented theory's concepts in the product configuration context. The heritage link stands between "generic" entity and its specialized objects. In a product configuration, both of the specialization and composition are represented in a hierarchical tree. The composition embodies the relationship between a composed part and its components. In this hierarchy, a composite can be derived from its components (there is no circular structure). Specialization highlights the relationship between an item and its variants. Each variant inherits not only the properties and attributes of the "generic" part, but also all relationships of the "generic" part with other entities (including therefore its composition).

This is illustrated in figure 5.2.

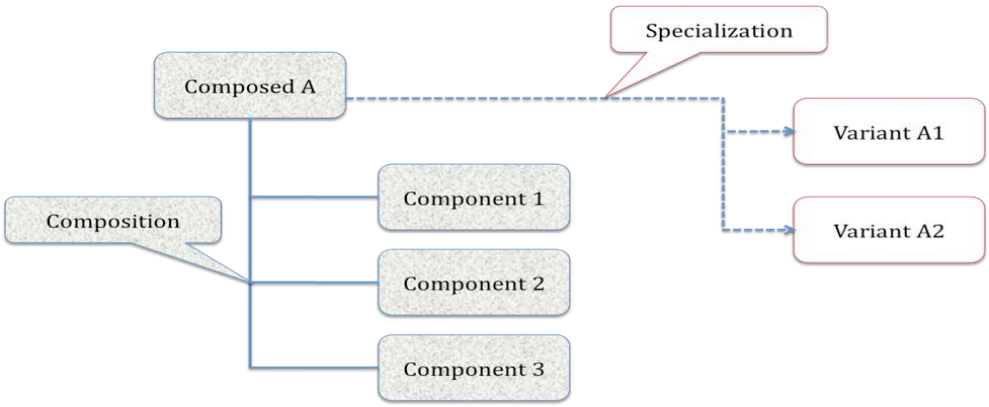


Figure 5-2, Specialization and composition in product configuration

For example, all the "specific" motors, which are related to the "generic" model of a type of rotary motor, must have all of its attributes, components and structure. If this "generic" motor has some specifications, all of its related "specific" motors possess these specifications. And if a property of "generic" rotary model changed, this property may be modified automatically in all of its "specific" motors. It means that the "specific" entities are not separated from their "generic" entity after their creation. In other words, the "specific" models are always dependent on their "generic" representation. However, in some cases heritage is applied only in the beginning of "specific" configuration structuring, which means the saved "specific" configuration is not more dependent on the "generic" product configuration. These practices don't profit from the entire advantages of heritage in "generic" modelling.

- Component types:

The "generic" configuration may contain three types of components. A component may be obligatory, which means that this component exists in all derived "specific" products. In the other hand, a component can be optional; it belongs to a "specific" or

exemplary product depending the customer's choice. It can also be variant or constant. Variant components proposed a variety of several values for their properties and during the specialisation, one value should be taken, in contrary with constant components that don't propose this choice.

It is also possible to describe levels of compatibility between options and variants of components. They are defined as the constraints of specialization and explain which variant or option may be chosen for a product.

2. A New Approach to Built Generic Product Model

As discussed in previous section, a "generic" modelling framework is an appropriate solution for an enterprise to manage its product configuration. It may be more important for enterprises that fabricate high-variety products. In this case, a "generic" product may factorize a huge amount of knowledge in information system. In this section, we demonstrate an appropriate solution for migration form a specific-based PLM to a "generic" based one, according to the methodology described in previous chapter.

In our viewpoint, models are constructed from meta-models. The relationship between "generic" and "specific" models and meta-models are illustrated in figure 5.3. A "generic" model conforms to a "generic" meta-model. It can then convert to one or more "specific" models by specialisation. These "specific" models conform to a "specific" meta-model. Specific model may have different layers of specialisation, regarding to number of allocating variants or options. An exemplary object, which is the instance of models, can be created from all of these models. These instances are independent from each other.

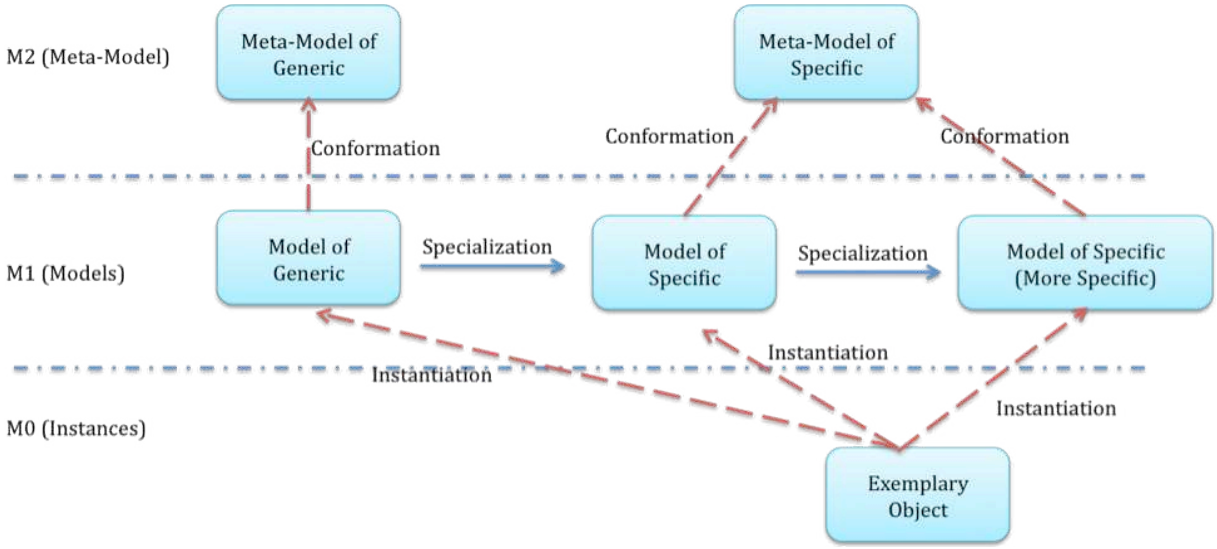


Figure 5-3, levels of abstraction via MDE

2.1. Industrial context and problematic

Our industrial example is about a company with information system that handles only exemplary and "specific" levels of product configuration. According to the advantages of

genericity management, During PLM system implementation, the company's administration decided to set up a "generic" configuration framework in order to reduce the cost of data insertion to system as well as increase the design and manufacturing processes by having a "generic" view of product. Therefore the problematic consists of two parts, 1) needs to have a mechanism for building the "generic" models and 2) but which procedure, the new modelling system should be set up.

The current situation of the enterprise, and its expected future system is summarized in Table 5.1. In the current system, there is a central database, which contains only the codification of all parts and products of enterprise, without any information about the structure of products or the configuration. The configuration management is done by several excel files. In each file, the components of several products are listed. Therefore, after some years, the number of these files increases hugely. In the other hand, the communication between enterprise and its partners become difficult, because these files are not so easy to read or use. The enterprise has developed a system for communication between its internal sections. But the product structure is not diffused via this system. Therefore the enterprise has decided to replace its own-made semi-PLM and set up a commercial PLM.

In its new system, which can communicate easily with the information system of their partners, the enterprise would like to create the "generic" configuration of its products. Besides it prefers to migrate its existing products configuration to the new system and reorganise and stock them with the new structure conforming to "generic" configuration.

Existing system used in the enterprise

General state of the current information system

- Centralized system of codification for all components and products fabricated in the enterprise (CACAO)
- A simple separated information system for communication
- Excel files, which contain all of the technical description of products including the configuration of all products.

Current product configuration system:

- A huge number of "specific" product configurations within lots of Excel files.
- Each product configuration defines a few number of products fabricated within the enterprise.

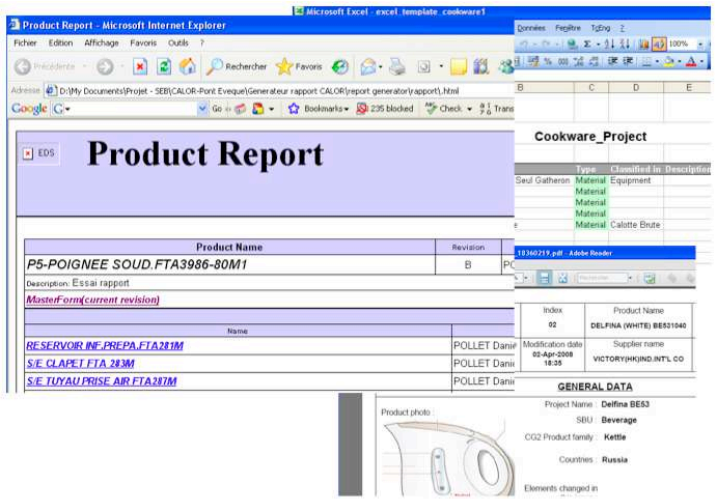
Target system that is aimed to be installed

General state of the future information system

- A commercial PLM system, which permits to
  - Manage the knowledge related to products
  - Facilitate the communication
  - Collaborate with enterprise's worldwide partners
  - Make the relationship between different products and components

Future product configuration system:

- Generic product configurations which contain the genericity of several products
- Possibility to follow the state of evolution of each component within different products.



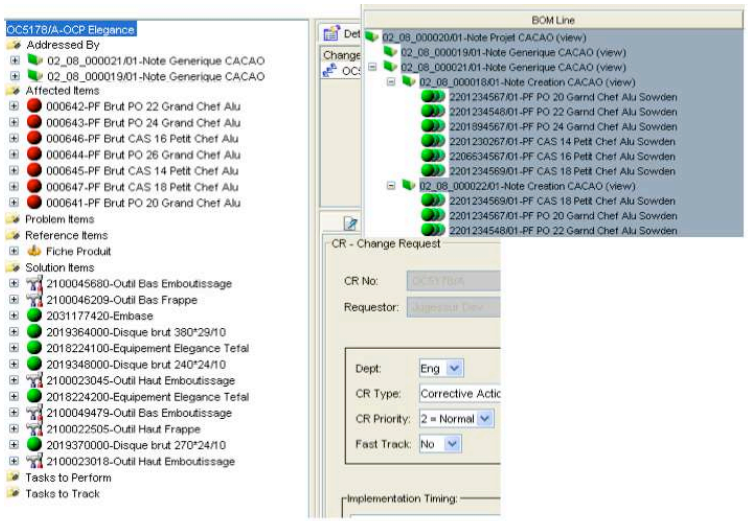


Tableau 2, Current and future systems

2.2. The Solution

Our solution is based on the MDE principals and the evolution procedure described in previous chapter. With MDE viewpoint, this industrial problematic may be reformulated to a process of migration of PLM system with a meta-model, which supports only the "specific" products to a PLM system with a meta-model that can handle genericity. In other words, in the initial system only "specific" models conform to product meta-model but our objective is to have a system in which, associated "generic" models may conform to product meta-model.

This industrial case study corresponds to the second scenario of evolution (cf. chapter 4). The target meta-model is not identified previously, so our proposition includes an approach for identification and formulation of the key concepts of "generic" product meta-model. Similar to problematic, it contains two parts. The first part is to construct the target meta-model, and the second part is to execute the transformation. In order to construct and feed the "generic" product meta-model, we used the data that exists in current "specific" system.

To present every step of the proposed approach, we rely on this industrial example on structuring the "generic" configuration of cookware. According to our overall procedure described in previous chapter, we have translated the industrial problematic to an informatics equivalent that is based on MDE principals and concepts. So, the first phase is completed. We have determined that our problematic corresponds to which scenario, the second. Figure 5.4 illustrates the rest of the procedure, which is customised for this special part of problematic.

There are four types of cookware that is fabricated in this company: frying pan, saucepan, wok and stockpot. Each type has different models and categories. The information system of the company manages separately each "specific" model and the objective of this procedure is to construct a system, which manages them in a "generic" way. In the following sections, the different steps of figure 5.4 will be discussed and explained.

The objective of different steps belonging to the second phase is to construct the meta-model of "generic" configuration. The proposed method to construct this meta-model is based on the enterprises requirements and expectations for the future system. After several meetings with the experts of enterprise, the concepts that they need to be present in the future system are identified. Then these concepts are used to construct the first draft of meta-model. In this step, entities of the meta-model are not completely defined. This draft should be completed during the different following steps, via a recursive process. This succession will be demonstrated by a concrete example.

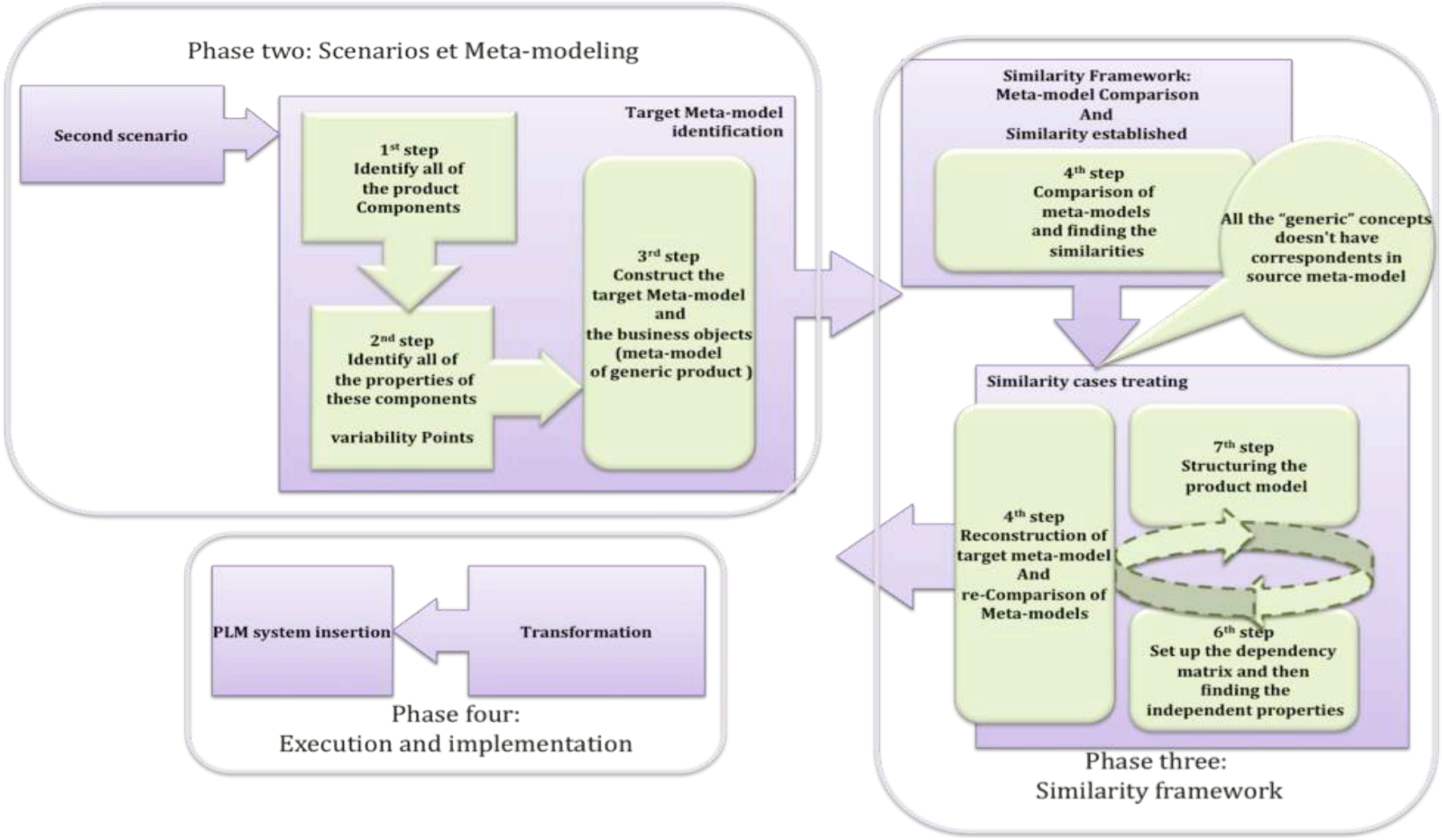


Figure 5-4, Steps of our proposition

Third phase is based on similarity between meta-models. In this phase the correspondents for meta-models' entities are identified. As mentioned before, this example is special case of second scenario. In this case, the target meta-model will be constructed, and the user can modify its entities or its structure according to his needs and the problematic situations. We used this particularity and modify the draft of target meta-model even during meta-model comparison. This innovative part, which will be presented in future sections relays on data and information from model level of "specific" configuration. In this type of meta-model comparison and model transformation, establishing the correspondence relationship between entities of meta-model requires the information from lower layer of modeling. This kind of model transformation is called inter-level model transformation.

During the forth phase, the transformation and implementation is the execution and insertion the results into the PLM system. During this research, two internships were done in order to fulfill a feasibility study for this part of implementation and create a tool. These works are presented in the sixth chapter.

Following this brief representation of different parts of the proposed method, our industrial example will be treated in order to clarify these steps and illustrate their details. The subject of our example, as it was presented before, is a cookware. In each step, an example of cookware, a frying pane is presented and the details are explained for it. But it's evident that these details are acknowledged for other types of cookware.

2.2.1. Phase 2:

In the second phase, the target meta-model should be found. In order to identify its entities, following steps should be taken.

The first step starts by identifying all elements in the composition of various products in a product line. The list of these components must cover all of "specific" products' components. This list defines the data of current system of enterprise. All of the cookware fabricated in the enterprise has the parts illustrated in the figure 5.5. They are composed of a deep dish, a pin, a basement, a handle and a screw to fix the handle to basement. These components are stocked in the current system with different properties. The current system contains all of this data for each "specific" component and "specific" product.

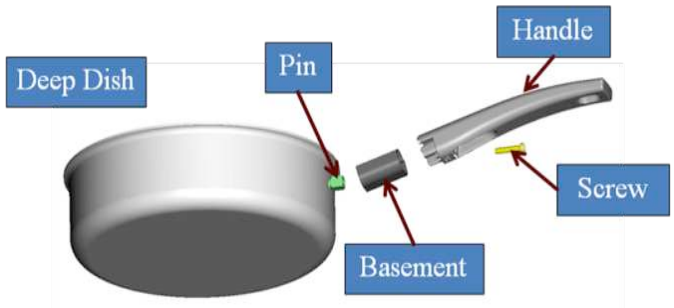


Figure 5-5, Composition of a pan

The second step is to identify all properties that are the source of the diversity of products within a "specific" level. It leads to the preparation of a list of variability points .The main concept of variability point comes from the fact that in a line of products, certain

properties of configuration elements differentiate a "specific" product from the others. These properties are the sources of diversity in a range of products(Zeng & Jin 2007). To build the "generic" configuration for a line of products, we should identify and manage these sources of diversity.

The properties of elements at the origin of the variety of pans are listed in the table 5.1.

Objects	**Properties**
Frying pan (the whole product)	Dimension of product
	Family of product
Deep-Dish	Dimension of Deep-Dish
	Family of Deep-Dish
Tool of Deep-Dish	Codification of Tool of Deep-Dish
Pressed Disk	Dimension of Pressed Disk
Tool of Pressed Disk	Codification of Tool of Pressed Disk
Equipments (handle, Pin, Basement and Screw)	Family of equipments
	Parameters related to dimensions of equipments as well as dimension of product (especially for Basement)
lid	Dimension du lid
	Family of lid
Tool of lid	Codification of Tool of lid

Tableau 3, Component properties

These properties distinguish a "specific" product from another. In the current system, in which there is no trace of genericity, all of "specific" products are stocked with their "specific" components, which possess different properties. All of these properties of "specific" components have the value in the system.

In the third step, the nature of concepts of configuration is defined and formalized. It means that in this step the physical items, documents, plans, etc. should be identified and represented as a meta-model. The definition of these business objects can define the elements of the meta-model of product model. This meta-model manage the construction process of the product model by defining business objects and their relationships. As mentioned before, product meta-model describes the concepts and their relationships. For our industrial example, the meta-model of the new system is defined in Figure 5.6.

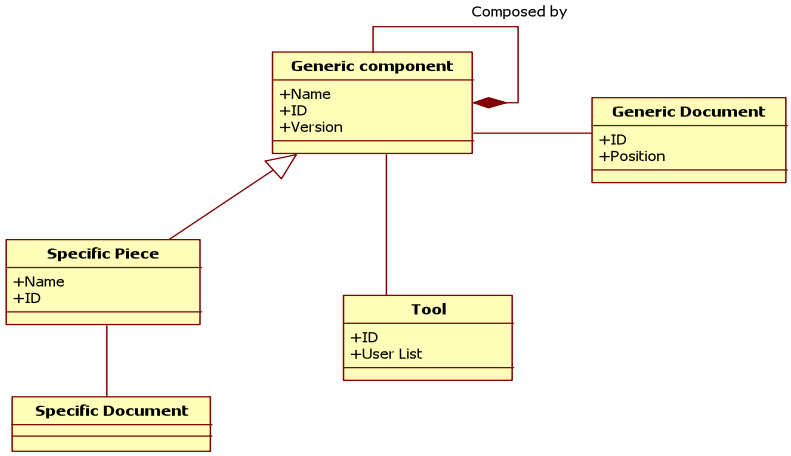


Figure 5-6, the draft of "generic" product meta-model

This meta-model is a draft of final meta-model. It should be modified in order to determine its attributes and connections. A meta-model for a product describes a language for representing the real concepts; model' elements. Therefore the properties of meta-model entities should have a real signification. In other words, during meta-modeling, we should always take into account that the models, which will be constructed from this meta-model, should contain enterprise information, i.e. not just the representations of "generic" product.

2.2.2. Phase 3:

This meta-model then should be compared to the meta-model of the initial system, "specific" meta-model, in order to find the correspondences between their entities. The fourth step is to compare the initial and target meta-model to identify these relationships between concepts by taking into account the similarities between them. Then the type of similarity is identified based on various cases of similarity, described in previous chapter. Figure 5.7 illustrates the comparison of meta-models, for our industrial example.

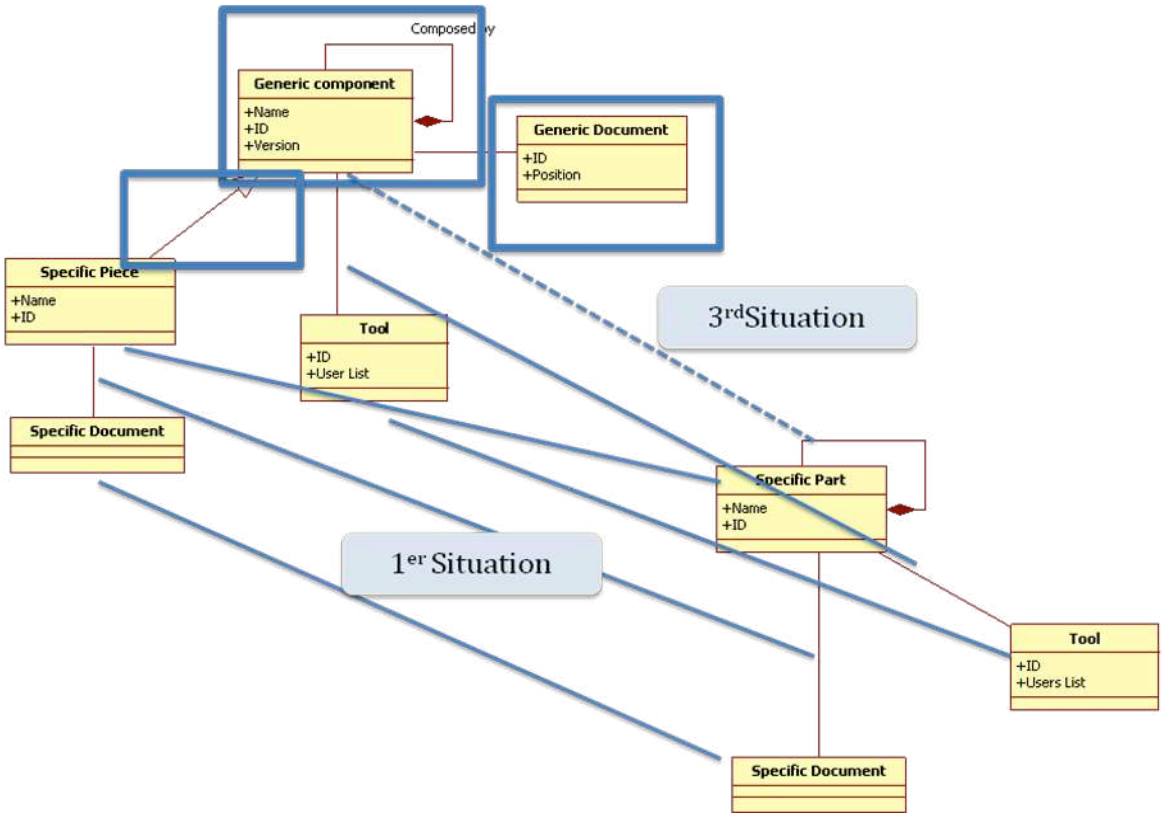


Figure 5-7, Meta-model comparison

In this figure, dark links represent exact similarities between concepts, i.e. instances of these concepts are identical in target and initial meta-models. They correspond to the first situation of similarity described in the previous chapter.

Concepts in the squares are the concepts of the target meta-model that are not reflected in the initial meta-model. They don't have any correspondent concept. We can find no similar concept for them in the initial meta-model. They have no equivalent in the current system then they are not precise and therefore may remain empty in the new system. To avoid this, we must invent a procedure to find the necessary knowledge from initial models or meta-model, which helps us to identify these concepts in the target system.

- The document is a "generic" concept related to the "generic" parts. Then it will be determined after construction of the concept of "generic" part. That means that it remains unmatched.
- In the other hand, the heritage relationship between "generic" and "specific" parts is represented by the factorization of "specific" part's information to its associated "generic" part. It will be done by eliminating the specific-related information in the "specific" part and bring them to its upper level, "generic" part, after its construction.
- The following steps are proposed to identify the information on the construction of the concept of "generic" part. This is possible based on existing information in the initial models. To establish connections at meta-model, we use the

information that is present in the lower level models. This is a very special case in model transformation.

To illustrate this approach, we present the approach to build the concept of ""generic" part" from instances of "specific" parts.

The following step is devoted to define the properties of "generic" part from the dependencies between product concepts and components' properties. Fifth and sixth steps are about to find these dependencies and construct the model of "generic" product, which conform to "generic" configuration draft meta-model. Proposed procedure is to construct some new concepts, which regroup the component of "specific" product. These new concepts have some properties that depend on the properties of the containing components. For example, a motor, which regroups several components have some attributes depending on these components, such as the capacity. Capacity of a motor is a new concept for a motor that has a value depending on its components' or parts' properties. Therefore "capacity" is not an independent attribute of a motor. As well, some of the components of motor have some dependent properties on each other. For example, the diameter of pistons depends on the size of cylinders. The objective of these steps is to find these dependencies between properties. In order to do it, we design the product model. This model defines the structure of the product configuration, explaining its organization, attributes, and entities. At this step, components may regroup or new abstract elements may be constructed. The purpose is to identify dependencies between the variability points, which are listed at the end of the second step. To identify these dependencies, we use the dependency matrix (Dependency Structure Matrix)(Clarkson et al. 2004)(A. A. Yassine & D. Braha 2003). In this dependency matrix, we compare all the variability points of all components. If the property in raw depends on the property in column, the according cell is checked.

We distinguish two types of dependence between variability points: the strong dependency and weak dependency. The strong dependency means that if a variability point has a fixed value, necessarily the other variability point also has a value. In case of weak dependency, the variability point can have multiple values. So in this case there is not a match but a restriction (HARMEL 2007).

After construction of the matrix, we must invert rows and columns to have some independent "submatrices". Figure 5.8 illustrates the principle of this technique. In this hypothetical example, there are 4 properties to compare; property A depends on D and C depends on the property B.

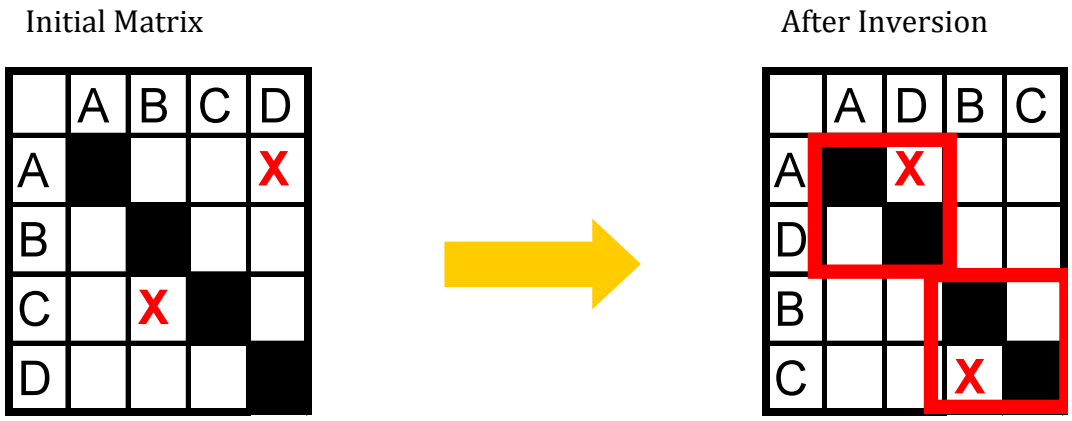


Figure 5-8, Dependency matrices

These components belong to the owners' parts of 1, 2, 3, and 4. The result of this seventh step is to complement the work already done in the sixth step. Each independent sub matrix reveals a possible combination of components (the components associated with the properties included in the sub-matrix). Figure 5.9 shows this principle with two groups of components.

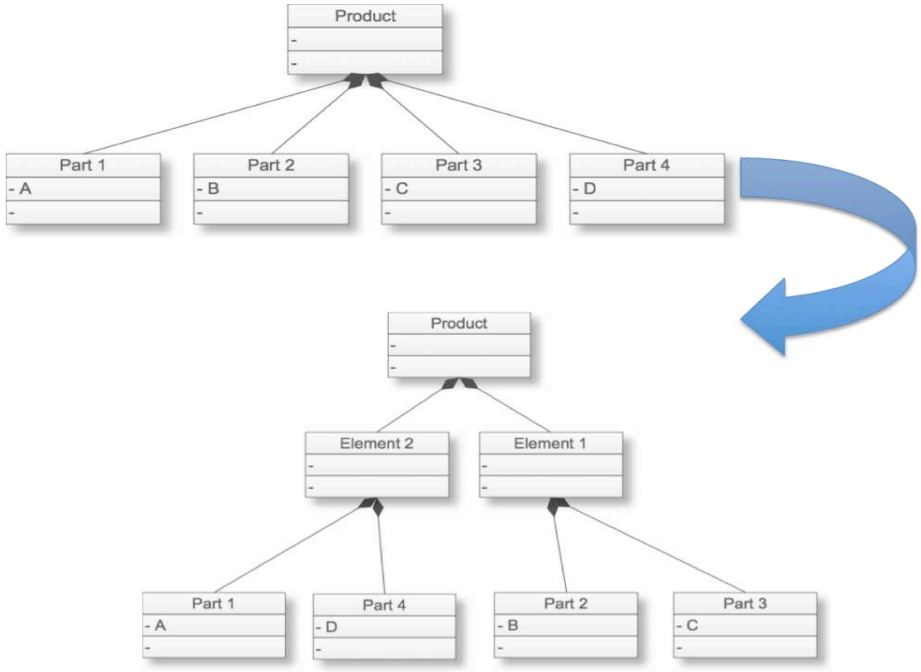


Figure 5-9, Components' regrouping

This grouping introduces two new elements in the intermediate classification. They are abstract because no physical component corresponds to these elements. Dependencies between the properties may also help us to factorize properties and attributes to build these abstract components by removing the redundant attributes of each component. This is illustrated in Figure 5.10.

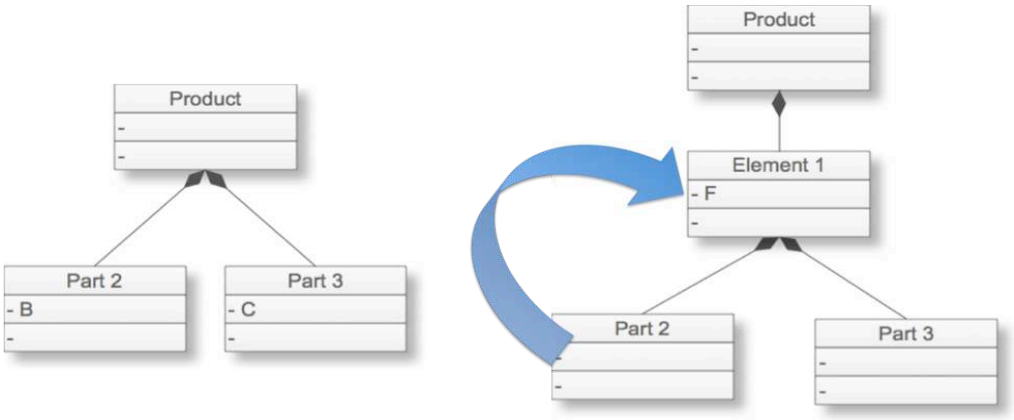


Figure 5-10, Factorization

In Figure 5.10, properties B and C are similar so we can remove them and bring them to its upper abstract level, and therefore factorize them. But this is not always possible for all cases. In some cases the new attribute in abstract level, F may be introduced and the value of B and C is determined while the value of F is known.

At this step there are 3 possible models:

1. There is a common property (high dependency) between the components, and then we construct the intermediate elements, as Figure 5.10. C and B are identical, or by identifying F, both will be recognized.

2. There is no common property but it is possible to introduce a new abstract property to the intermediate elements, which factorizes some properties of the components of the group. Thanks to this, the number of independent properties decreases.

3. There is no possibility of eliminating the component properties in the group. In this case, the intermediate element of this group has not attribute but at least we can group the related components of product and offer a more logical structure.

However, with this grouping, the constraints associated with the properties of components come together, facilitating their definition and modification.

If we add new properties, we must rebuild or reconfigure the matrix and repeat the phase of product model construction. The result of these steps is a model structure, which factorize through the information corresponding to component properties of the product in a more optimal manner.

The “generic” configuration is built according to the properties that remain independent. For each set of allowed values of these properties, there is a corresponding “generic” configuration. Specific configuration, and therefore the exemplary configuration are constructed by determining properties and choosing optional variants.

By applying steps 5 and 6 to our industrial case, the dependency matrix, which is shown in figure 5.11 is obtained.

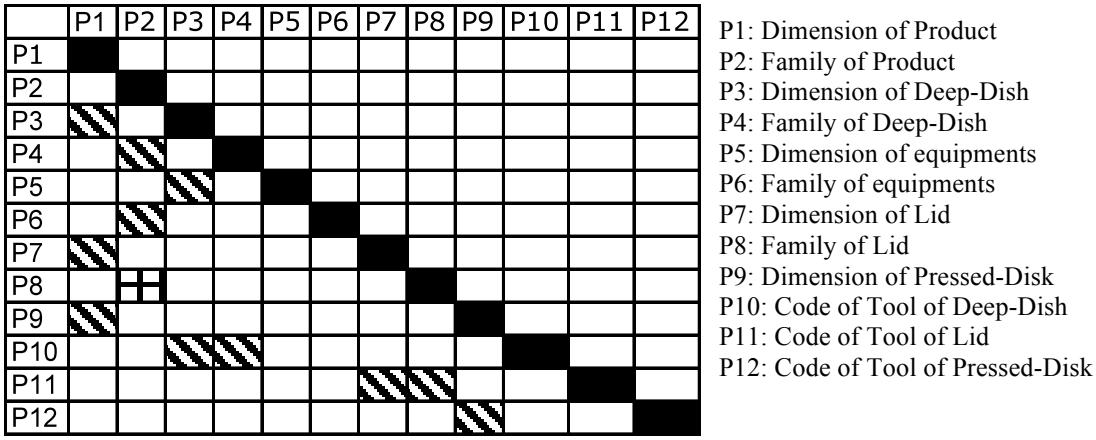


Figure 5-11, Dependency matrix of pan

Strong dependencies between properties reduce amount of dependent attributes. These are summarized in Table 5.3

Property A	**Property B**
Dimension of Deep-Dish	Dimension of Product
Family of Deep-Dish	Family of Product
Dimension of Pressed Disk	Dimension of Product
Prometers related to the dimension of Equipements	Dimension of Product (for Basement, dimension of Deep-Dish)
Family of Equipements	Family of Product
Dimension of Lid	Dimension of Deep-Dish
Family of Lid	Family of Product
Tool of Lid	Dimension and Family of Lid
Tool of Deep-Dish	Dimension and Family of Deep-Dish
Tool of Pressed Disk	Dimension of Pressed Disk

Tableau 4, Dependencies for a pan

The dependency matrix indicates that the size and family of product are independent properties of a pan, with which it is possible to distinguish the characteristics that define the "generic" configuration. Therefore the "generic" configuration becomes a function of the family and size of product. For example, a "generic" frying pan may be "26 cuisine" (26: dimension, cuisine: family)

The structure of the "generic" nomenclature of this frying pan is illustrated in Figure 5.12.

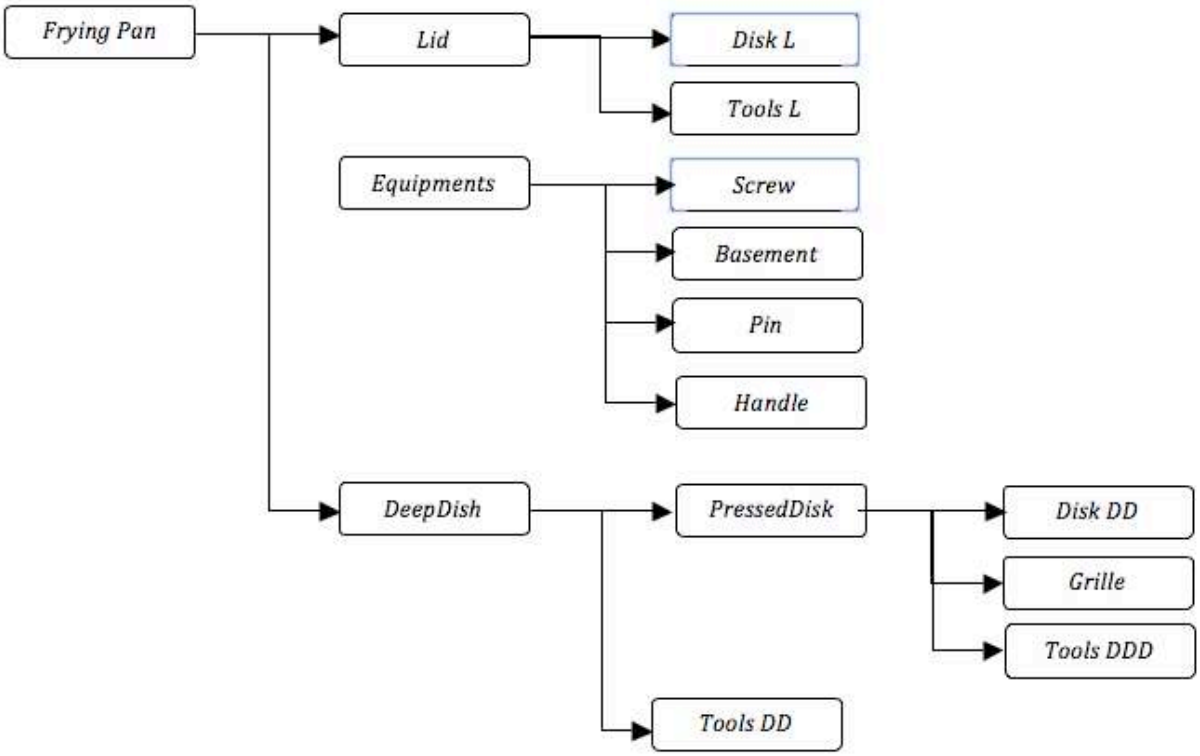


Figure 5-12, structure of product

This structure is used to describe the "generic" configuration of product. This "generic" cookware represents the "generic" product of the enterprise, its meta-model "generic"

Then, by using this approach, we can find the information needed to construct the "generic" concept and add instances of this concept in the associated "generic" models. This information helps us to identify the attributes present in the "generic" parts. Therefore, our draft meta-model will be completed by concrete definitions for different entities and concepts existing in it.

2.2.3. Phase 4:

Model transformation is done by using the results of phase 3. In this phase, based on the "specific" models, some "generic" models will be constructed and then the "specific" models, which associated with them, will be reconstructed. In this reconstruction, all factorized information will be replaced in "specific" model by the heritage concept. Then "specific" model depends en their "generic" model during its evolution. Each evolution in "generic" model will influence automatically on associated "specific" models.

The process of transformation and implementation is done as follow: for each source model concept:

- We identify the meta-concept of initial meta-model.

- The corresponding meta-concept in target meta-model is identified.

- The target model contains the instance of this target meta-concept.

- The information may be needed in order to enrich the concepts of target models will be added.

This process is done automatically by a tool, which may be fed by XML files. Technical aspects of model transformation as well as implementation and communication will be discussed in the next chapter.

3. Conclusion

Generic product configuration is a special case of product model evolution within the enterprise information system. However, its importance and frequent usage in industry tempted us to take it as our example of product model evolution in PLM systems. The objective of our proposition is to manage the entire process of evolution from a "specific"-based system to a "generic"-based one. It consists of establishing a methodology for meta-modeling of "generic"-based models, but we prefer to notice that if the meta-modeling is done properly, finding the similarity will became more regular. In other words, a good constructed target meta-model leads to help the model comparison. It means that even if the biggest part of this methodology relays on meta-modeling, but it must never keep out of mind that model comparison was facilitated by a good meta-modeling. This methodology is feasible and validated by the company's experts. According to their remarks, it can produce the new models conforming to enterprises requirements and situation.

Until now, it doesn't exist an organized and structured framework that contains the criteria to evaluate meta-models and meta-modeling processes, especially for product meta-model. Our evaluation is based on experts' observation and validation. These experts are the future users of system. Needs to set up a concrete and customized based framework for evaluating product meta-models are seen. Future researches may improve and progress the process of meta-model evaluation and validation.

Implementation of this method depends also on the commercial tool that is used in the enterprise. Some tools propose to users an open structure to build up their own plug-ins and execute their processes within the system, some others don't allow this. This will be discussed in the next chapter to show the particularities of implementation. We will talk about the process of design and test of our tool, and the other functionalities that we would like to add to our proposed tool.

Chapter 6 Tool

Chapter four was devoted to propose a methodology in order to solve the evolution problematic of product models in PLM systems. The conceptual procedure was illustrated in figure 4.1. It starts with problematic translation. Then according to an identified scenario, the solution follows its path throw meta-modeling if it's necessary. Subsequently, in similarity framework, the source and target meta-models are compared in order to establish the correspondence relationships between their concepts.The different types of these relationships are described in chapter four. Results of this stage is then introduced in transformation engine, to execute the evolution and new models conforming to new meta-models, as well as new data corresponding to new models to be generated.

Besides to the proposed methodology, we present in this chapter the informatics tool we develop to support this methodology. The aim is to help and direct PLM users in defining their meta-models and models, establishing the similarity relations and executing the transformation. This tool is semi-automatic; besides an automatic part a manual part guides tool's user toward the phases that should be followed to accomplish product model evolution. This chapter is devoted to describe different functionalities and particularities of proposed tool, as well as a brief description of its design and implementation project.

Before starting this chapter, it should be noted that the user of this tool is a person who governs the evolution of the PLM in an enterprise. He may be an integrator or a PLM administrator of the enterprise. In this chapter, "user" refers to this person.

1. Tool's needed functionalities

The most important functionalities of this tool are managing similarity framework and executing model transformation. Managing similarity framework includes the following functionalities:

- Comparing two meta-models.
- Identifying possible correspondences between concepts of two meta-models.
- Visualizing of the correspondence relationships.
- Determining the type of similarity and proposing a possible solution in case of abnormality.
- Allowing the user to create, delete and modify this proposition.
- Visualizing and validation of final proposition.

These functionalities constitute the main core of the proposed tool. In order to fulfill these functionalities, the following parameters must be accurately identified:

- List of possible concepts
- Criteria of comparison
- Scoring scale for each criterion
- Weight for each criterion.

These parameters are issued from the DSM, which describes the concepts and their properties, as well as their relationship. Therefore, this tool must allow declaring and updating information related to DSM, which may be used by similarity framework in a semi-automatic way. Therefore this tool must provide an interface for visualizing and inserting this information.

It should also be able to read input files coming from PLM system and providing output files which are readable by the PLM system. This tool is feed with information that comes from a PLM system. PLM systems are supposed to create XMI files, which contain their product's data model. Therefore these XMI files should be readable. Also meta-models should be inserted to the tool.

A use case diagram summarizing the needed functionalities, is presented in figure 6.1:

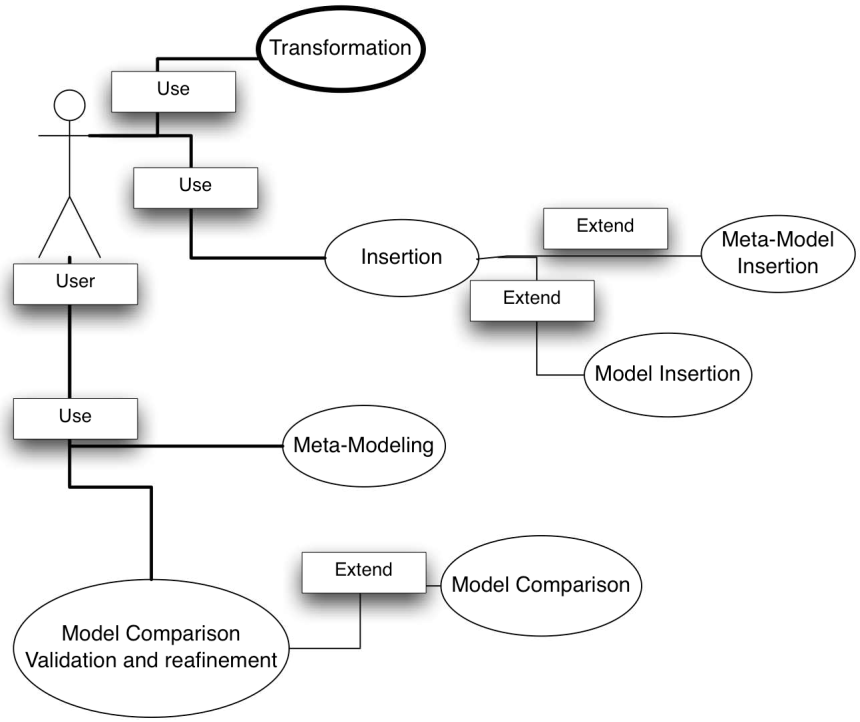


Figure 6-1, Use Case Diagram of proposed tool

2. Tool's operation

The activity diagram presented in figure 6.2 illustrates the succession of different stages within this tool in order to ensure the functionalities described in § 1:

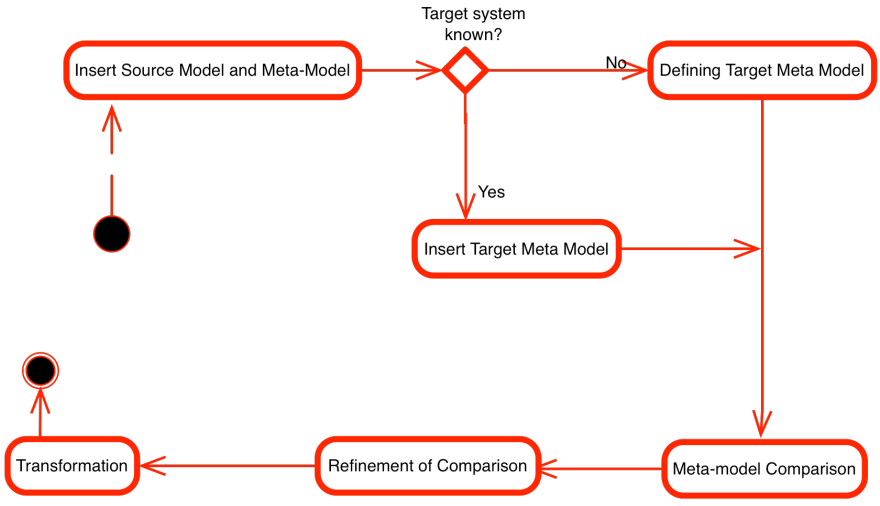


Figure 6-2, Activity diagram

Following sub-sections are detailing the various stages introduced in this diagram.

2.1. First section, Input

Firstly, meta-model and models of initial PLM system should be inserted to the tool. It may be done manually by uploading XMI files, or as a plug-in, this tool directly is fed by the PLM system. This depends on the PLM systems particularities. Meta-models are not yet so organized and structured in PLM systems. They are designed by PLM integrators but it seemed that there is no output file, which contains them. Therefore, this schema must be designed by the user. Our tool may provide an interface in order to construct meta-models.

Then, this tool must validate the conformity of model and meta-model. There are some associated tools, which fulfill this task. In our chosen framwork, Eclipse, which be described later, this task is done automatically. Subsequently, meta-model of target system should be identified. If it's known, the modeling interface may help the developer to design its schema, and if not, he can construct it.

2.2. Second part, DSM and Meta-model Comparison

The next stage is to insert criteria, scores and weight in order to run the model comparison, as described in forth chapter. The tool can have some pre-inserted contexts containing the possible concepts, which may exist on meta-models. In the case of formerly-known modeling context for the tool, the list of possible existing concepts, which may probably display in meta-models, are shown with their associated criteria, scores and weights. User can modify this list. If these concepts and associated criteria is not known, he can create this list by using information exists in meta-models. Indeed, as mentioned in chapter 4, criteria are based on existing concepts in two meta-models.

The meta-model comparison is then processed, calculation based on the formula presented in chapter 4 is done, and the list of possible correspondences is presented to the developer. Next, the user found solutions for different difficult situations. As described in the forth chapter, these difficult situations correspond to the cases where the correspondences lines are not straightforward. The user, based on the context, must find the necessary information in order to establish these correspondences, like the case described in fifth chapter. A graphic interface, which illustrates the relationship between concepts of two meta-models, the correspondence line, may help the developer to follow properly the procedure.

2.3. Third part, Transformation

Based on the correspondences, the transformation takes place. For our case, we have chosen the "Eclipse, ATL platform" to execute this transformation. This platform is presented in detail in the following section. ATL, proposed by INRIA (Institut National de Recherche en

Informatique et en Automatique) de Nantes, is an open platform, which is devoted to transform models. Transformation process will be described in section 3.

2.4. Forth part, Inserting the data in to new system

In this phase, the new model should be inserted to the new PLM system. It depends on the particularities of this specific PLM system. The programming language to use for inserting data files in PLM may hugely influence on this part of tool's activity.

This is a global overview of the proposed. Section 3 describes in detail how does it work.

3. Tool development advancement

Development of this tool was a subject of an internship within G-SCOP laboratory(C. Duque 2010). This internship aimed to implement the transformation of product models in PLM systems by using Eclipse platform.

The general objective is to build a software tool for model transformation, divided into two steps. The first step is about receiving the source and target meta-models and the particular source models, and then building the target models by using transformation process. Then the aim of second step is to use these source and target models and by using codes of earlier transformation, find new instances of the target models, from the source model instances. Therefore, in first step, finding the target model is the first challenge of the tool. Figure 6.3 illustrates the first layer processing, transformation of first step.

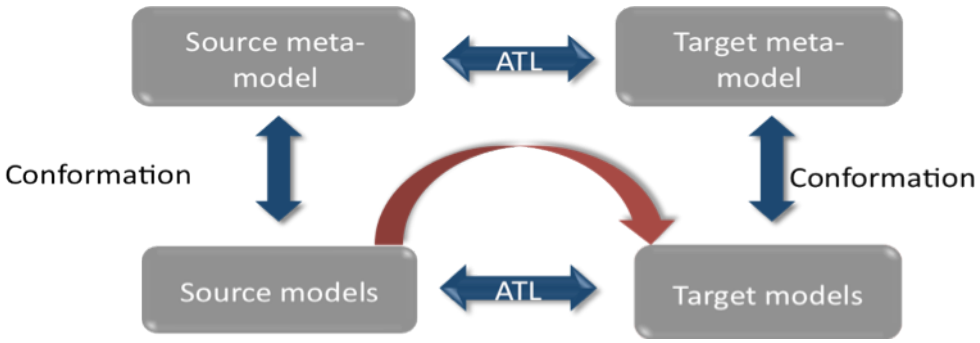


Figure 6-3, first layer of transformation

Then, in order to use Eclipse as platform environment, inevitably, the two sources and target meta-models should be translated in an Eclipse appropriate language (Anon 2010). For meta-models, Eclipse ATL accepts them in Ecore language(García-Magariño et al. 2009).

After finding the target model, which is generated in XMI format, the second layer of processing allows the migration of data from a PLM tool to another. This means that the proposed tool have to retrieve instances of the source model (data) and then to build a new ATL transformation(Frédéric Jouault et al. 2008), which, after execution, will generate a set of instances conform to the target model (conform: fitting to the target model format) that can then be inserted in the new PLM system. Figure 6.4 describes this second layer of transformation.

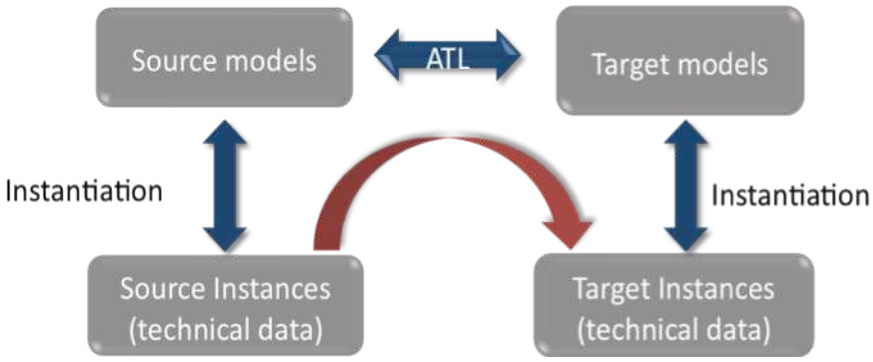


Figure 6-4, Second layer of transformation

In order to implement this tool, the first step was to choose the technical environment to use. For model transformation, there are several possible tools; XSLT, KMTL and ATL.

After some meetings with MDE experts and taking into account the visibility and selecting Eclipse as our implementation framework, we have chosen Eclipse especially that the EMF Framework was quite sufficient to meet the required conditions.

Eclipse is an open source community that amid to construct a development platform for free. The performance of this software can be increased because of the different frameworks, tools and execution platforms. The EMF (Eclipse Modeling Framework)(García-Magariño et al. 2009) project is a framework for code generation in order to facilitate the construction of applications based on the model structure. From a model defined in XMI (XML Metadata Interchange), EMF provides tools and execution engine to produce a set of Java classes for the model, all accompanied by a set of other classes which facilitates the process of viewing and commanding to construct and edit the model as well as a basic editors. Within this framework, ATL(Atlas Transformation Language)(Anon 2005) is a language for transforming models, designed by the research group ATLAS at INRIA, as part of the IDM approach. ATL provides developers with an automated procedure in order to find the target models from a set of source models.

The ATL language is a hybrid of declarative and imperative programming. However the preferred style is declarative because it easily structures the process of set up correspondences between source and target meta-models. In the same way, ATL provides imperative instructions to structure the correspondences that are not easily produced by declarative way.

An ATL transformation program establishes a set of rules which, in one hand, define the correspondence between the source meta-model and target meta-model, and in the other hand, determine the path from source model's elements to create target model's elements.

The ATL language, developed as a component of Eclipse framework, provides a set of standard tools, which make it easier to write the transformation. Besides, ATL-Eclipse offers several additional functions for manipulating models and meta-models. These functions or features include a textual notation for simple specification of models, such as standard tools that make a communication bridge between the general language of modeling and representation language of models in Eclipse.

These functionalities offered by ATL-Eclipse, confirm our choice to implement this project within Eclipse, since the user doesn't need to write the Java codes. It is sufficient for him to enter the UML diagram and then Eclipse is responsible of generating the Java code that exists behind. These facilities, which are associated with working environment, ATL-Eclipse, make it easier the process of reading and visualizing projects.

4. Experimentation scenario

In the considered scenario to experiment the proposed tool concerns a bicycle manufacturer. The initial PLM system used by the manufacturer to manage bicycles data is based on the following meta-model:

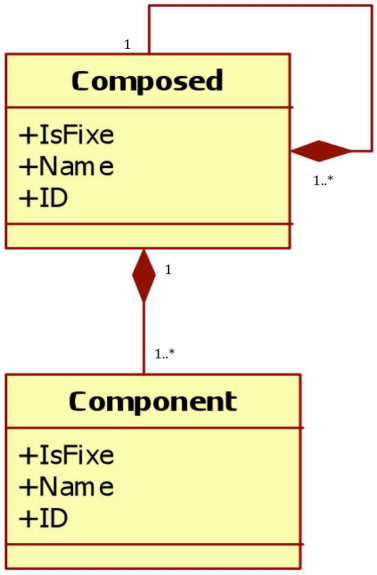


Figure 6-5, Source meta-model

Then, due to market changes, the manufacturer feels the need to transform its PLM tool so to be able to take into account its new production strategy. Therefore, it must change the meta-model of its PLM to another one in which its product (bicycle) divided into a fixed structure and a mobile structure. This meta-model is illustrated in figure 6.6.

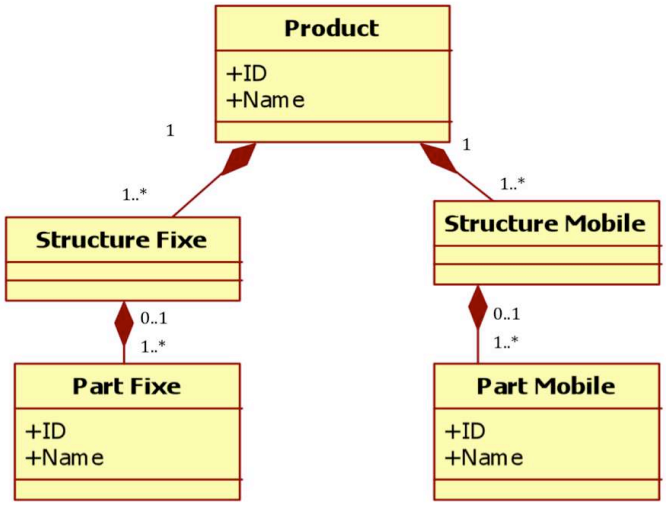


Figure 6-6, Target meta-model.

To use the new structure, the manufacturer must transform its current product models (source models), which are currently conform to source meta-model defined in Figure 6.5, into models which are conform to the target meta-model given in Figure 6.6. In eclipse environment,

input models (here source model) are written in XMI formats. The source model, which is inserted to transformator, is illustrated in figure 6.7.

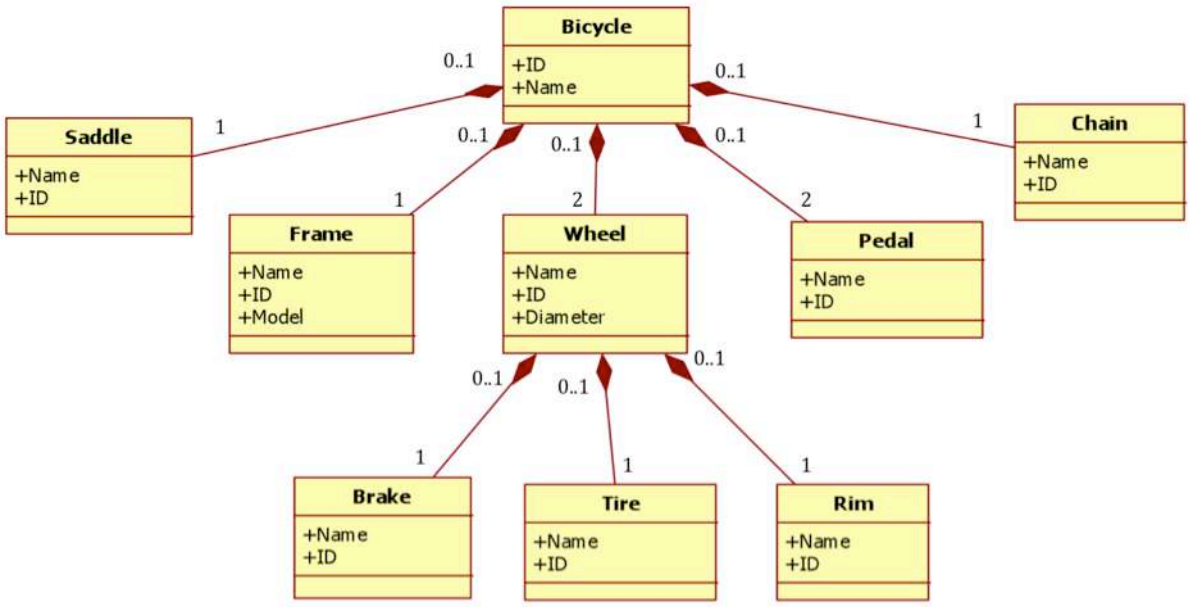


Figure 6-7, Source Model.

The code transformation script should be able to make the correspondence between the concepts of source and target meta-models. Then, as mentioned before, a detailed comparison between these two meta-models must be made. According to this comparison, and based on the methods proposed in chapter 4, some rules which guarantee the accuracy and correctness of correspondences are identified and implemented in ATL code of transformation. Without entering to detail, as an example, some of these rules are: 1) product is always a root component; 2) The mobile structure includes all components that are mobiles. These rules are rewritten in the form of transformation rules: 1) a composed part becomes the product if and only if it is a root component; 2) a component is a fixed part, if its attribute “Is Fixe” is true.

As described before, second layer of transformation is about data (model instances). At this stage of experimentation, the tool has constructed the new model to be inserted into the target PLM system, but data stored in initial PLM system is not yet compatible with the models in the new system.

Returning to the example, the bicycle manufacturer company follows a model of a bicycle, which is made of composed parts and components. It means that all data related to bicycles fabricated in this company are stored in PLM in the format of the source model. These bicycles are all of the instances to give as input to the second layer of transformation.

Second layer transformation objective is the transformation of data for each bicycle, which are conform to the model of bicycle, obtained at the first layer of processing (target model). Therefore in this layer of transformation the superior level is models and the inferior level contains data. The result of transformation consists of all technical data stored in initial PLM system, but with the new format, which conforms to new PLM system's meta-model and models.

This transformation must be able to define the correspondence between "bicycle component/composed part system" and "bicycle structure fixed/mobile system". In the same way as in the first layer, it begins with model comparison. The principles of this comparison are same as the precedent. By execution of the two layers of transformation, this tool provides, as a result, the target model and the new instances. Then, the enterprise is ready to use its new PLM system, which contains the new meta-model, new models as well as associated data.

As mentioned in forth chapter, loss of information during model transformation depends on the meta-models' correspondence. Therefore, in some cases the loss of information is inevitable. But, one of important advantages of using this process of data transformation is to ensure that there is no loss of information or data compatibility problems, if the correspondents are found during meta-model comparison. In addition, new data conforms to new model and, if it will be necessary, the probable future transformation may be feasible.

.An important problem encountered during this practice is the format of output and input files in ATL Eclipse platform. The superior level is in Ecore format, but inferior level's format is an XMI. The output of first transformation is a XMI file that represents target model, which is the input of second transformation as the superior level. Therefore, inevitably, XMI files coming out from first transformation should be "transformed" to Ecore files in order to enter to second transformation. Without entering to technical details of solution, it should be noted that this type of translation should be done by another transformation, prepared and implemented during tool development. Its objective is to translate the XMI files to Ecore files. This is shown in figure 6.8.

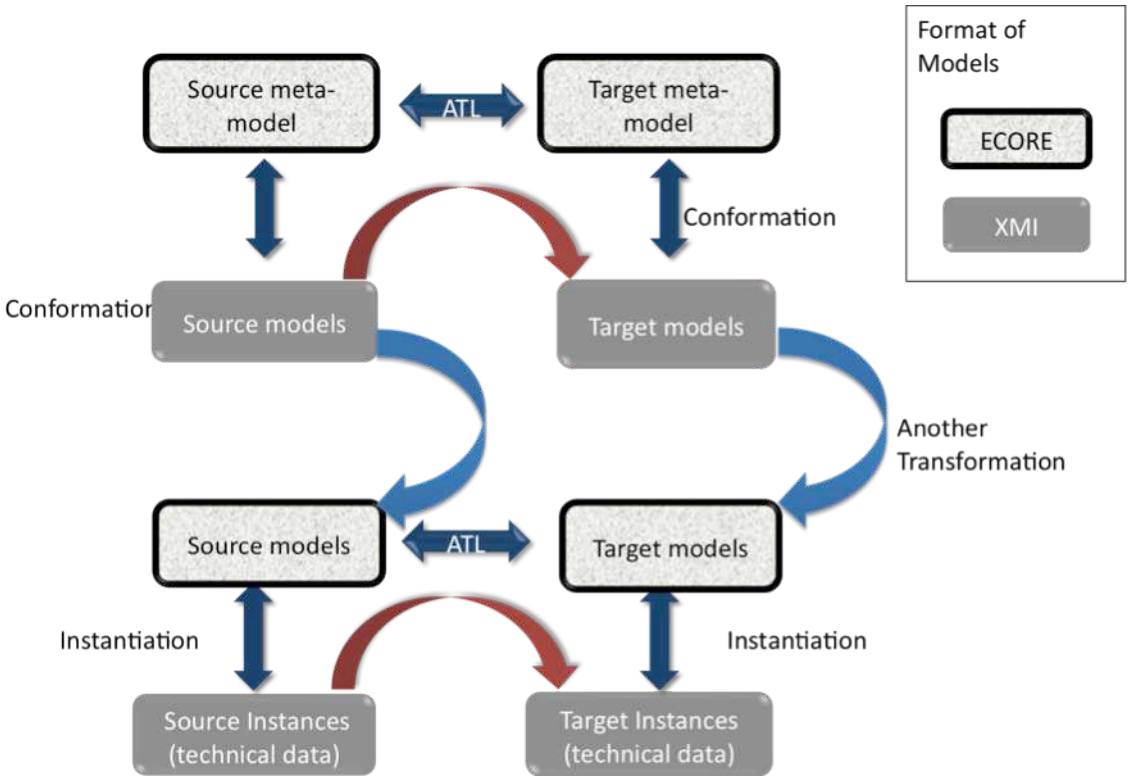


Figure 6-8, The transformation between formats (translation)

5. Tools Demonstration

As discussed earlier, tool development was designed and implemented in the eclipse ATL framework. In this platform, the meta-models are inserted in Ecore language, which are shown in figures 6.9 and 6.10:

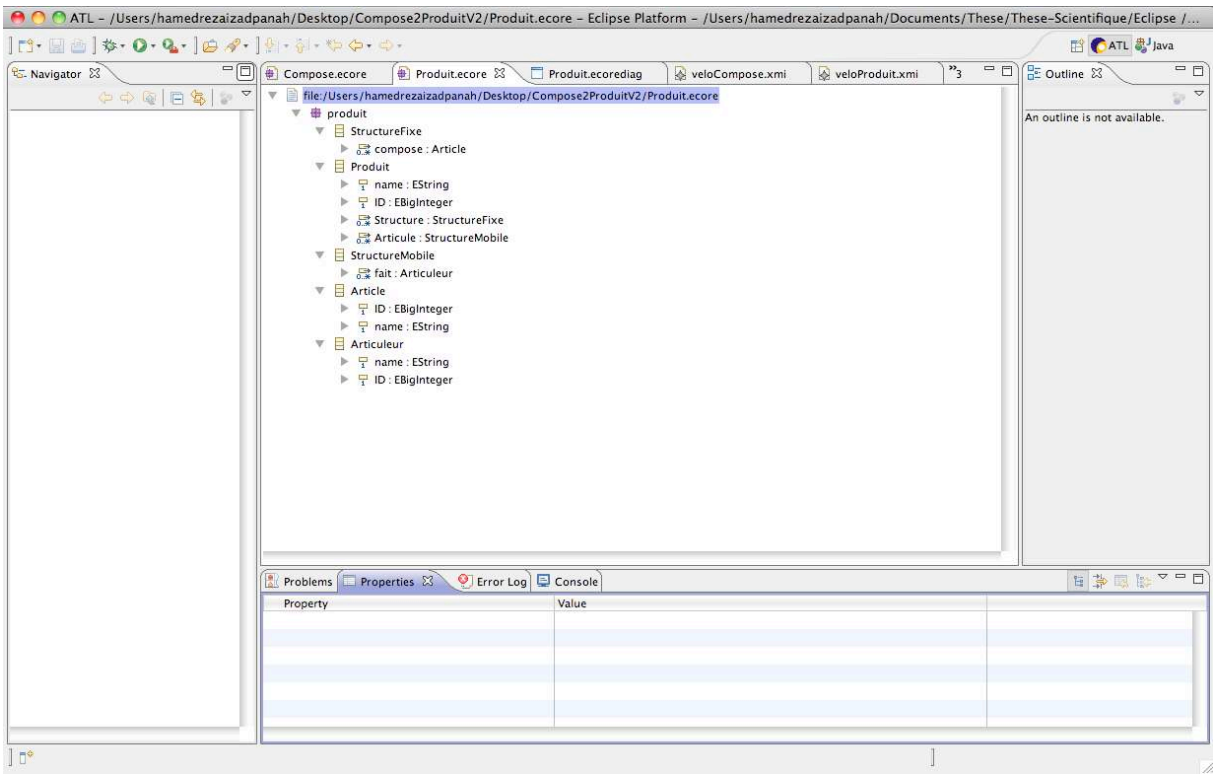


Figure 6-9, Initial meta-model insertion

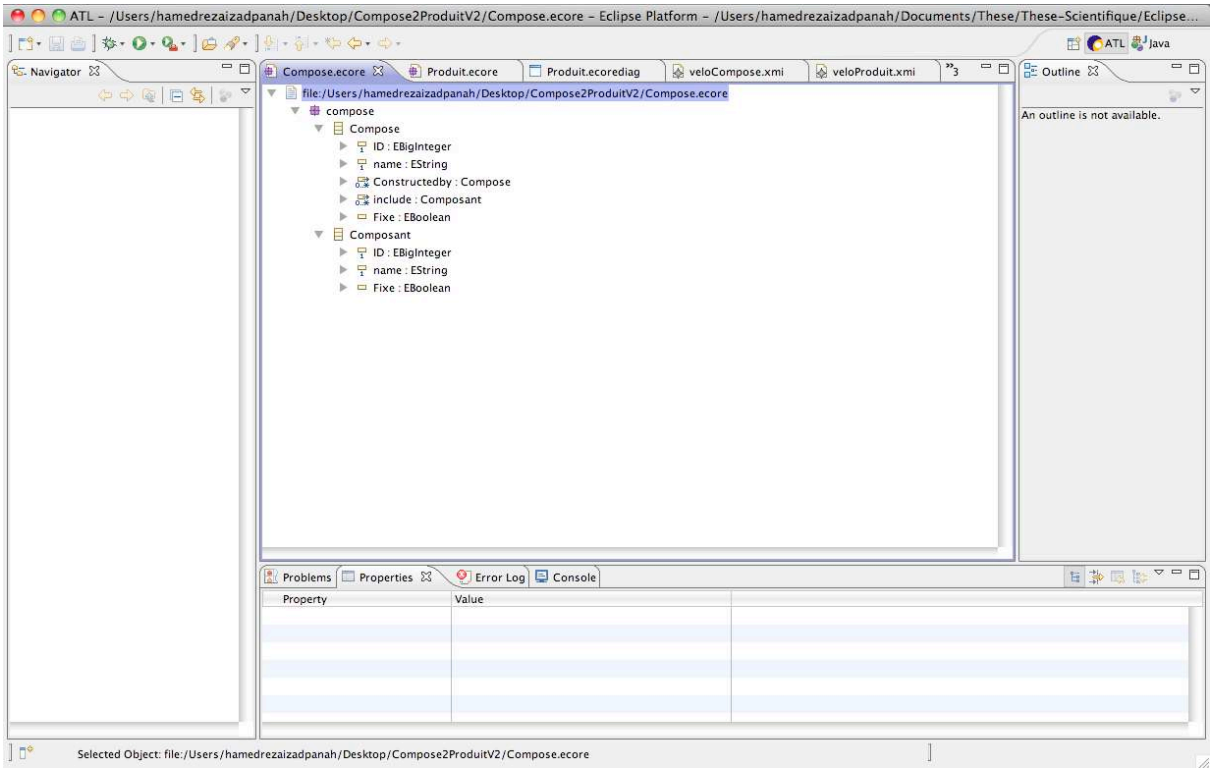


Figure 6-10, Target meta-model insertion

The comparison is done in an ATL environment by connecting the correspondent concepts, as shown in figure 6.11:

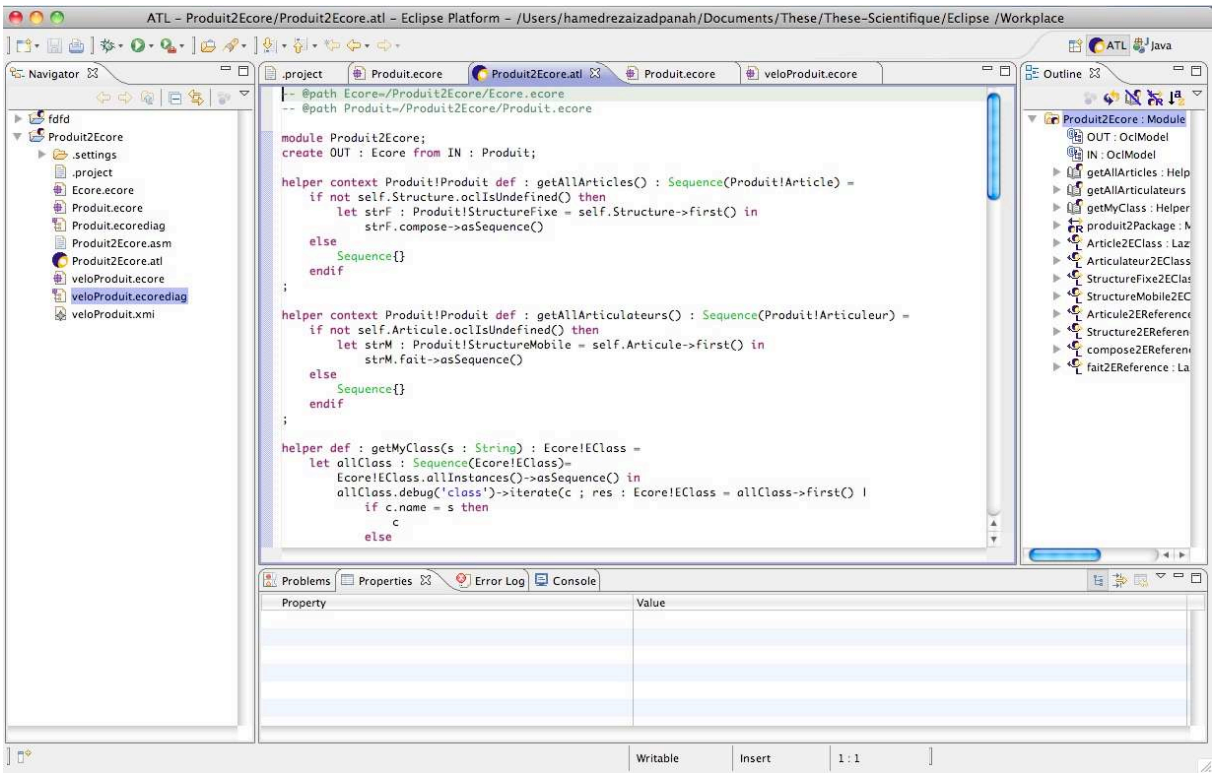


Figure 6-11, ATL comparison framework

The initial model should be fed to system in XMI format, and then the result, target model, is constructed by ATL framework with this format.

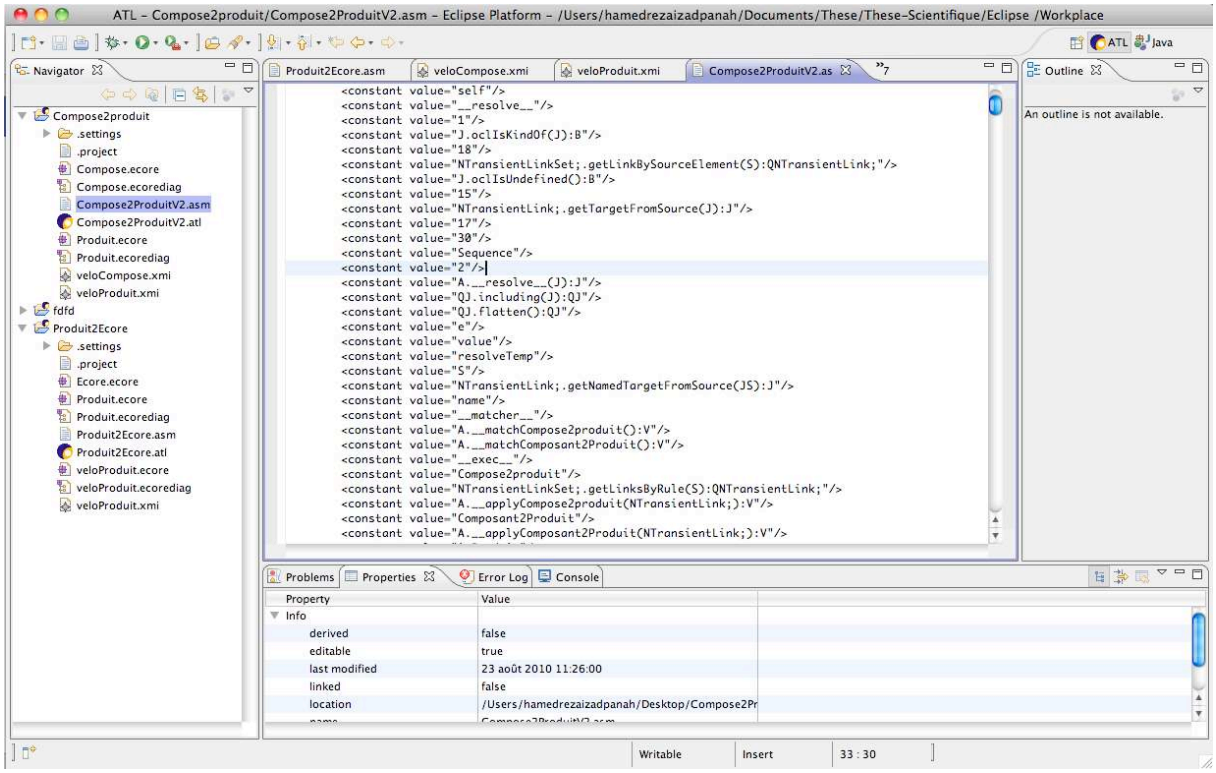


Figure 6-12, XMI format of model level

6. Conclusion

This prototype is a part of the whole project of tool development, which is envisaged and initiated for this research. The complete tool should be capable of communicating with PLM systems in order to take into account and give back the information. Besides, it can help users during the similarity analysis and establishing correspondence relationship between meta-models' concepts. Another desired functionality of this tool is to describe possible terminology of a specific domain and represent the terminology of the PLM system within an enterprise. The user can modify and update its ontology by an appropriate interface, which guides him through different levels.

Moreover, during internship of Carolina Duque (C. Duque 2010), an interactive user guide has been proposed in order to understand and facilitate probable future modification of prototype codes and procedures.

General Conclusion and perspectives

New advancements in information technologies provide a variety of facilities for enterprises to implement and deploy new information systems. These new innovations of information technology persuade the enterprises to profit from new abilities of these services and set up a more reliable and general covering information systems, which manage their functionalities, facilitate their communications, and treat their data. Product lifecycle management systems have an important role in this context and prepare a worldwide collaborative framework in order to organize and manage product-related technical data and information within the industry and treat workflows and processes related to it. In other words, theses systems aid the enterprises to stock, organize and share data between their partners and collaborators. As a result, nowadays, PLM systems have a crucial role between other information systems in enterprises. They propose a reliable communication between different systems and follow any changes not only in product structure but also in enterprise's organization. PLM systems aren't restricted to only the used informatics application but also cover all of information management strategies in enterprise.

Evidently they may be influenced by several probable evolutions that take place within enterprise. Nowadays, industry faces lots of challenges, which make modification, change and evolution as ineluctable strategy, aiming to better adapt to its environments or proper respond to its clients. This wave of change may also touch its information system, which manage and flow its information and shape its organization. One of these affected systems is PLM. Many industrial problematic can cause the evolution in PLM systems. Second chapter of this dissertation has counted several possible sources of PLM evolution. Any change in product, informatics application or even business processes of enterprise may entail a PLM evolution problematic. As a summary, it should be noted that one of the most important parts of proposed solution is finding which part of a PLM system may be affected during the execution of this evolution. In

other words, the solution depends totally on the influenced part of PLM architecture. As a scientific literature conclusion, we would like to point out that the general discussion of PLM evolution is a vast subject, which has not been yet explored regarding to industrial events and requirements, despite of huge needs.

Moreover, product model is a central structure in PLM systems. It defines with which organization the information should be stocked and managed. This makes the product model more sensitive to any changes of enterprise. Many causes of PLM evolution cause product model evolution. In the other hand, changes of product model may cause some probable modification in structure of PLM systems. Therefore there is reciprocal relationship between PLM and product configuration in the question of evolution. Product model evolution is an old problematic in industrial information system management. Product is one of the most evolutive object in enterprise and therefore its model in information system may undergoes lots of modifications. We have studied the different types of product model evolution and the corresponding effects and the various related industrial problems. Modifications on product model have different effects depending on the nature and reason of change. Some demand of modification can be only treated by simple changes on model itself, but in some others, the higher level of treatment may be needed to solve the modification problematic. In other words, modeling concepts or elements should be redefined or added, which even changes the modeling framework. So, the problematic can change the strategy of product model evolution.

Model Driven Engineering (MDE) has been chosen as our approach to treat the product model evolution and propose a method to handle and solve the problematic. MDE is a new paradigm for model management in informatics community, which shows a good performance in other areas, such as industrial model management. Then, our analyze and proposition are under the norms and concepts of MDE. MDE separate a modeling framework on several levels. In each level, the necessary concepts for constructing the lower level concepts are defined. In third chapter the principles of MDE and their implication in PLM modeling have been presented and discussed. Model transformation is a core process of MDE. Several methods and tools are developed to execute model transformation. Our proposed method uses its concept to perform product model evolution.

We have presented and described the detail of our proposition in forth chapter. In this method, we have tried to classify the problematic in four different scenarios and then propose a roadmap for each of them. As discussed in forth chapter, except first scenario, the others arrive to a similar transformation problematic. But, in second scenario, when design of the new system will be just considered as a phase of evolution, the process of transformation becomes more structured with less complication. In other words, a good meta-modeling is considered as a central part of model transformation. It may be one of the particularities of using MDE-related methods for industrial problematic. DSML was presented in the forth chapter as a standard method of meta-modeling; which is used in several business and industrial cases. The overall proposition for solving product model evolution needs to be more customized for each specific problematic in order to adapt to its situations and requirements.

For a special case, this proposition was customized and then tested. This case, which was studied during this research, is a real problematic of transformation from a system that manages

the “specific” product configuration to a system, which is capable to handle the genericity of product. Related concepts in “generic” product models are discussed in fifth chapter, which was continued to the adaption of overall proposition to this case and finished with explanation of each step’s details.

To conclude, we prefer to justify our interest to study product model evolution, implication of MDE and the validity of the proposition.

1. Perspectives

1.1. DSML

Domain of PLM studies needs a very structured and comprehensive modeling language, which covers the principle concepts of different parts of this type of information system. This DSML may be also the synergy of concepts that exist in several parts of PLM architecture. Some efforts have been done to construct an appropriate ontology for PLM and its associated product model, but it’s not really sufficient. A good DSML may facilitate all process of implementation and evolution of PLM systems.

1.2. MDE

MDE is a powerful method in model evolution. It has some standards and related tools. But in industrial cases, especially in information systems, it’s not already used effectively. Some academic researches have been started, which show its usability and performance in solving information system problematic. On the other hand, MDE has been presented in order to simplify complicated modeling framework (by separating model and meta-model) and facilitate the conception and management of models and make the modification more fast and reliable. In PLM systems management is not already used as well. Some researches are seemed to be needed in order to adapt MDE methods to the domain of PLM design and maintenance.

1.3. Comparison methods

MDE model transformation relays hugely on comparison of meta-model. Therefore several researches are devoted to develop different methods and approaches to model comparison. These vary from simple text comparison to some more complicated model differentiation and graph comparison. More adapted to software engineering models, they should be adapted to information system models in order to fulfill particularities of this domain. It should be noted that the DSML and model comparison might be dependent. The concepts that are defined in DMSL should be compared in model comparison. Therefore these activities are more interrelated.

1.3.1. Other parts’ evolution in PLM systems

This research was dedicated only to the important problem of product model evolution, but other parts of PLM architecture may be affected during evolution. For example, workflows,

which define the processes to be followed in the enterprise, are sensitive to evolution and particularly product model evolution. Developing a framework to handle and execute theses parts' evolution should be studied in future researches.

References

A. A. Yassine & D. Braha, 2003. Complex Concurrent Engineering and the Design Structure Matrix Method. *Concurrent Engineering*, 11(3), p.165-176.

A. A. Yassine et al., 2004. Investigating the role of IT - in customized product design. *Production Planning & Control: The Management of Operations*, 15(4), p.422.

A. Cicchetti, 2008. *Difference Representation and Conflict Management in Model-Driven Engineering*. Universita di L'Aquila.

A. Cicchetti, D. Di Ruscio & A. Pierantonio, 2007. A Metamodel Independent Approach to Difference Representation. *Journal of Object Technology*, 6(9), p.165-185.

A. Cicchetti et al., 2008. Meta-model Differences for Supporting Model Co-evolution. *2nd Int. Workshop on Model- Driven Software Evolution (MoDSE 2008)*.

Anon, 2005. ATL User Tutorial.

Anon, 2010. Eclipse Hero User Guide.

B. Amar et al., 2008. Un Framework de traçabilité pour des transformations à caractère impératif. Dans Langage Modèles et Objets LMO'08. Montreal, Canada.

Brambilla, M., Fraternali, P. & Tisi, M., 2009. A Transformation Framework to Bridge Domain Specific Languages to MDA. Dans *Models in Software Engineering*. p. 167-180.

Brière-Côté, A., Rivest, L. & Desrochers, A., 2010. Adaptive generic product structure modelling for design reuse in engineer-to-order products. *Computers in Industry*, 61(1), p.53-65.

Bézivin, J., 2005. On the Unification Power of Models. *Software and System Modeling*, 4(2), p.171-188.

C. Duque, 2010. *Automatisation pour la transformation de modèles orientée aux outils PLMOUTILS PLM*, IUT Grenoble.

C. Sung & S. Park, 2007. A component-based product data management system. *The International Journal of Advanced Manufacturing Technology*, 33(5), p.614-626.

Charles Eastman & Jeng, T.S., 1999. A database supporting evolutionary product model development for design. *Automation in Construction*, 8(3), p.305-323.

Chen, C.-H., Ling, S.F. & Chen, W., 2003. Project scheduling for collaborative product development using DSM. *International Journal of Project Management*, 21(4), p.291-299.

Clarkson, P.J., Simons, C. & Eckert, C., 2004. Predicting Change Propagation in Complex Design. *Journal of Mechanical Design*, 126(5), p.788-797.

Demoly F., 2010. *Conception intégrée et gestion d'informations techniques: application à l'ingénierie du produit et de sa séquence d'assemblage*. Université de Technologie de Belfort-Montbéliard - UTBM.

Demoly F. & Gomes, 2009. Assembly-oriented product structure based on preliminary assembly process engineering. Dans *Proceedings of the International Conference on Engineering Design, ICED'09*. ICED'09. Stanford, San Francisco, CA, USA.

Deridder, D. et al., 2009. Model Co-evolution and Consistency Management (MCCM'08). Dans *Models in Software Engineering*. p. 120-123.

Dhungana, D. et al., Structuring the modeling space and supporting evolution in software product line engineering. *Journal of Systems and Software*, In Press, Corrected Proof.

Eastman, C. M. & Bond A., 1991. A Data Model for Engineering Design Databases. Dans 1st International Conf. on A.I. in Design. Edinburgh, Scotland.

Eynard, B. et al., 2004. UML based specifications of PDM product structure and workflow. *Computers in Industry*, 55(3), p.301-316.

Eynard, B. et al., 2006. PDM system implementation based on UML. *Mathematics and Computers in Simulation*, 70(5-6), p.330-342.

García-Magariño, I., Fuentes-Fernández, R. & Gómez-Sanz, J.J., 2009. Guideline for the definition of EMF metamodels using an Entity-Relationship approach. *Information and Software Technology*, 51(8), p.1217-1230.

Gardner, T. & Griffin, C., 2003. Review of OMG MOF 2.0 Query/Views/Transformations Submissions and Recommendations towards final Standard. *OMG*.

D. Garwood, 1988. *Bill of Material, Structured for excellence*,

Goknil, A. & Topaloglu, Y., 2005. Ontological perspective in metamodeling for model transformations. Dans Esbjerg, Denmark: ACM, p. 7.

Golovatchev, J.D. & Budde, O., 2007. Next Generation PLM - an integrated approach for the Product Lifecycle Management. Dans *Proceedings of ICCPR2007: International Conference on Comprehensive Product Realization 2007*. Proceedings of ICCPR2007: International Conference on Comprehensive Product Realization 2007. Beijing, China.

L. Gzara, 2000. *Les patterns pour l'ingenierie des systemes d'information produit*. Institut national polytechnique de Grenoble.

Gzara, L., Rieu, D. & Tollenaere, M., 2003. Product information systems engineering: an approach for building product models by reuse of patterns. *Robotics and Computer-Integrated Manufacturing*, 19(3), p.239-261.

HARMEL, G., 2007. *Vers une conception conjointe des architectures du produit et de l'organisation du projet dans le cadre de l'Ingénierie Système*. Université de Franche-Comté.

Hick, J.-M. & Hainaut, J.-L., 2006. Database application evolution: A transformational approach. *Data & Knowledge Engineering*, 59(3), p.534-558.

Hong-Bae Jun et al., 2006. System architecture for closed-loop PLM. Dans *Information Control Problems in Manufacturing 2006*. Oxford: Elsevier Science Ltd, p. 805-810.

J. M. Favre, 2004a. Foundations of Meta-Pyramids: Languages vs. Metamodels - Episode II: Story of Thotus the Baboon. *Language engineering for model-driven software development, number*.

J. M. Favre, 2004b. Foundations of Model (Driven) (Reverse) Engineering: Models -- Episode I: Stories of the Fidus Papyrus and of the Solarus. *Post-proceedings of dagsthul seminar on model driven reverse engineering*.

J. M. Favre, 2004c. Towards a Basic Theory to Model, Model Driven Engineering. *Workshop on software model engineering, wisme2004, joint event with UML2004*.

J. M. Favre & T. NGuyen, 2005. Towards a Megamodel to Model Software Evolution Through Transformations. *Electronic Notes in Theoretical Computer Science*, 127(3), p.59-74.

Jackson, E. & Sztipanovits, J., 2009. Formalizing the structural semantics of domain-specific modeling languages. *Software and Systems Modeling*, 8(4), p.451-478.

Jiao, J. & Tseng, M.M., 1999. An Information Modeling Framework for Product Families to Support Mass Customization Manufacturing. *CIRP Annals - Manufacturing Technology*, 48(1), p.93-98.

Jiao, J. et al., 2000. Generic Bill-of-Materials-and-Operations for High-Variety Production Management. *Concurrent Engineering*, 8(4), p.297-321.

Jouault, Frédéric & Kurtev, I., 2007. On the interoperability of model-to-model transformation languages. *Science of Computer Programming*, 68(3), p.114-137.

Jouault, Frédéric et al., 2008. ATL: A model transformation tool. *Science of Computer Programming*, 72(1-2), p.31-39.

Kolovos, D.S. et al., 2009. Different models for model matching: An analysis of approaches to support model differencing. Dans IEEE Computer Society, p. 1-6.

Krause, F.-L. & Kaufmann, U., 2007. Meta-Modelling for Interoperability in Product Design. *CIRP Annals - Manufacturing Technology*, 56(1), p.159-162.

Kurtev, I. & van den Berg, K.G., 2005. MISTRAL: A language for model transformations in the MOF meta-modeling architecture. *Lecture notes in computer science*, 3599(s).

Kurtev, I. & van den Berg, K.G., 2004. Unifying approach for model transformations in the MOF metamodeling architecture. Dans Model-Driven Architecture with Emphasis on Industrial Applications. Enschede, the Netherlands.

Kurtev, I., van den Berg, K. & Jouault, Frédéric, 2007. Rule-based modularization in model transformation languages illustrated with ATL. *Science of Computer Programming*, 68(3), p.138-154.

Lin, Y., Gray, J. & Jouault, Fré, 2007. DSMDiff: a differentiation tool for domain-specific models. *European Journal of Information Systems*, 16, p.349-361.

M. D. Del Fabro, 2007. *Gestion de métadonnées utilisant tissage et transformation de modèles*. Université de Nantes.

M. D. Del Fabro & P. Valduriez, 2009. Towards the efficient development of model transformations using model weaving and matching transformations. *Software and Systems Modeling*, 8(3), p.305-324.

Martin Gwyther, 2008. AVEVA NET – AVEVA's Open PLM Platform forShipbuilding. *International Conference on Product Lifecycle Management*.

Muccini, H., 2007. Using Model Differencing for Architecture-level Regression Testing. Dans *Proceedings of the 33rd EUROMICRO Conference on Software Engineering and Advanced Applications*. IEEE Computer Society, p. 59-66.

Männistö, T. et al., 1998. Modelling generic product structures in STEP. *Computer-Aided Design*, 30(14), p.1111-1118.

Männistö, T. et al., 2001. Multiple abstraction levels in modelling product structures. *Data & Knowledge Engineering*, 36(1), p.55-78.

Nieva T., 2001. *Remote data acquisition of embedded systems using internet technologies: A role based generic system specification*. EPFL.

Noel, F. & Roucoules, L., 2008. The PPO design model with respectto digital enterprise technologies among product life cycle. *International Journal of Computer Integrated Manufacturing.*, 21(2), p.139-145.

OMG, 2001. MDA Guide V1.0.1.

OMG, 2005. MOF 2.0/XMI Mapping Specification, v2.1.

Oh, Y., Han, S.-hung & Suh, H., 2001. Mapping product structures between CAD and PDM systems using UML. *Computer-Aided Design*, 33(7), p.521-529.

Panetto, *Meta-modèles et modèles pour l'intégration et l'interopérabilité des applications d'entreprises de production*. HDR. Université de Henri Poincaré, Nancy 1.

Pierantonio, A. et al., 2007. Special issue on model transformation. *Science of Computer Programming*, 68(3), p.111-113.

Rangan, R.M. et al., 2005. Streamlining Product Lifecycle Processes: A Survey of Product Lifecycle Management Implementations, Directions, and Challenges. *Journal of Computing and Information Science in Engineering*, 5(3), p.227-237.

Rivera, J.E. & Vallecillo, A., 2008. Representing and Operating with Model Differences. Dans *Objects, Components, Models and Patterns*. p. 141-160.

S. Rachuri et al., 2008. Information sharing and exchange in the context of product lifecycle management: Role of standards. *Computer-Aided Design*, 40(7), p.789-800.

Sadeghi, M., Noel, F. & Hadj-Hamou, K., 2009. Development of control mechanisms to support coherency of product model during cooperative design process. *Journal of Intelligent Manufacturing*.

Santos, A.L., Koskimies, K. & Lopes, A., 2010. Automating the construction of domain-specific modeling languages for object-oriented frameworks. *Journal of Systems and Software*, 83(7), p.1078-1093.

Schuh, G. et al., 2008. Process oriented framework to support PLM implementation. *Computers in Industry*, 59(2-3), p.210-218.

van Sinderen, M.J. & Ferreira Pires, L., 2004. Model-Driven Architecture with Emphasis on Industrial Applications. Dans 1st European Workshop, MDA-IA 2004. Enschede, the Netherlands.

Sudarsan, R. et al., 2005. A product information modeling framework for product lifecycle management. *Computer-Aided Design*, 37(13), p.1399-1411.

Svensson, D. & Malmqvist, J., 2002. Strategies for Product Structure Management at Manufacturing Firms. *Journal of Computing and Information Science in Engineering*, 2(1), p.50-58.

T. Asikainen & T. Männistö, 2009. Nivel: a metamodelling language with a formal semantics. *Software and Systems Modeling*, 8(4), p.521-549.

Terzi S., 2007. A new point of view on Product Lifecycle Management.

Varró, D. & Balogh, A., 2007. The model transformation language of the VIATRA2 framework. *Science of Computer Programming*, 68(3), p.214-234.

Willink, E.D., 2003. E.D.Willink UMLX: A graphical transformation language for MDA UMLX: A graphical transformation language for MDA.

Winkelmann, J. et al., 2008. Translation of Restricted OCL Constraints into Graph Constraints for Generating Meta Model Instances by Graph Grammars. *Electronic Notes in Theoretical Computer Science*, 211, p.159-170.

Y. Lin, 2007. *A model transformation approach to automated model evolution*. University of Alabama at Birmingham,.

Yang, D., Dong, M. & Miao, R., 2008. Development of a product configuration system with an ontology-based approach. *Computer-Aided Design*, 40(8), p.863-878.

Zeng, F. & Jin, Y., 2007. Study on product configuration based on product model. *The International Journal of Advanced Manufacturing Technology*, 33(7), p.766-771.

Zina, S. et al., 2006. Generic modeling and configuration management in Product Lifecycle Management. Dans *Computational Engineering in Systems Applications, IMACS Multiconference on*. Computational Engineering in Systems Applications, IMACS Multiconference on. p. 1252-1258.

Annexes

MANUEL D'UTILISATEUR : ECLIPSE EMF ET ATL

INTRODUCTION

Dans le cadre de l'ingénierie dirigée par modèles et plus spécifiquement dans la transformation de modèles, il recommandé d'être muni d'un environnement de travail assez puissant pour pouvoir mettre en place tous les avantages offertes pour cette approche.

Faire une transformation de modèles, demande un logiciel capable de manipuler modèles, ce qui veut dire qu'il doit être capable de les générer, les modifier, les instancier, les transformer, etc.

Le group Eclipse.org a développé tout un module, appelé « Eclipse Modeling Tools» qui est capable de gérer toutes les situations antérieurement décrites. Ce document est un manuel d'utilisateur envisagé à donner toute la documentation nécessaire pour réaliser une transformation de modèles dans la plateforme Eclipse avec la framework EMF.

Dans les chapitres suivants le lecteur pourra trouver la documentation depuis l'installation d'Eclipse jusqu'à la construction et exécution de toute une transformation ATL. Avec l'intérêt de faire plus agréable la lecture de ce manuel, il est accompagné d'un exemple qui rend plus claire toute l'information ci–jointe.

Définition de l'exemple

On suppose être au sein d'une entreprise de vélos dont son processus de production suit le méta-modèle suivant :

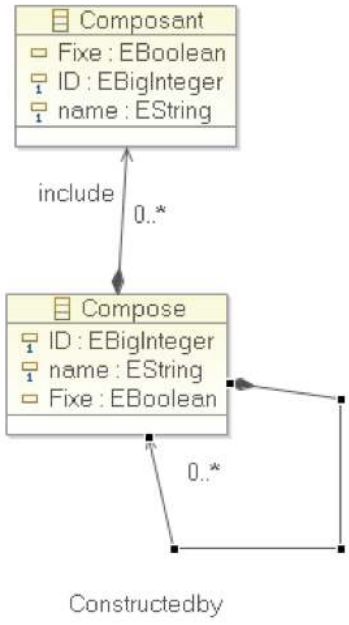


Figure 1 : Meta-modèle source

Cependant, à cause des conditions du marché, elle a besoin de transformer son méta-modèle pour un autre qui le permettra de savoir quels de ses produits appartient à la structure fixe du vélo et quels à la structure mobile. La figure 2 montre le nouveau méta-modèle.

L'entreprise doit alors transformer son modèle de production, qui actuellement est en accord avec le méta-modèle source, défini dans la figure 1, pour un modèle, conforme au méta-modèle cible de la figure 2.

Dans les chapitres suivants on expliquera au fur et à mesure tous les étapes nécessaires pour transformer ainsi bien, le modèle de production comme toutes les données des produits de l'entreprise.

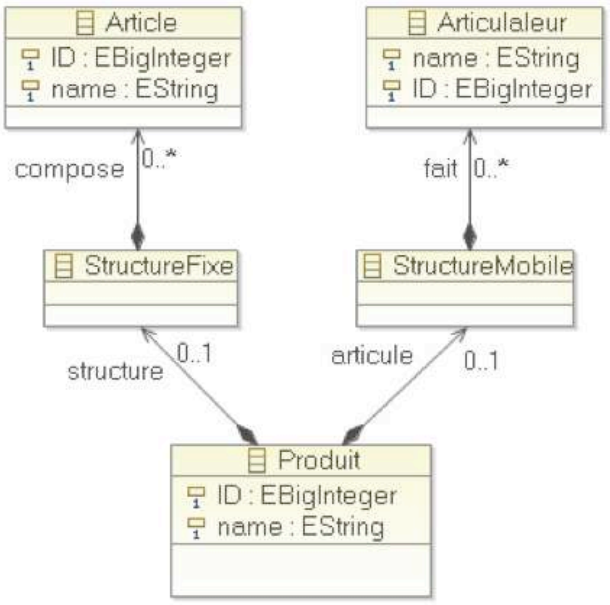


Figure 2 : Méta-modèle cible

I. INSTALLATION ECLIPSE EMF

I.1 Vous avez déjà une version d'Eclipse dans votre ordinateur

Vérification version Eclipse

Tous les versions d'Eclipse ne comptent pas avec ATL ni sont équipées pour travailler avec de modèles, il faut donc avoir la bonne version. Pour connaître la version sur laquelle vous travaille allez dans le menu « Help » à « About Eclipse » puis assurez vous d'avoir les spécifications suivantes :

Figure 1 : About Eclipse. Eclipse MyLyn Eclipse.org –EPP

Eclipse Modeling Project Eclipse Packing Project Eclipse.org

I.2 Vous n'avez pas une version d'Eclipse dans votre ordinateur

Pour télécharger Eclipse, allez-vous sur le site www.eclipse.org, choisissez le module « Modeling » et téléchargez le paquet « Eclipse Modeling Tools (includes Incubating components) » ensuite choisissez la bonne version selon votre ordinateur et téléchargez- le.

Décompressez le fichier dans le répertoire de votre choix.

I.3 Installation langage ATL

Pour pouvoir coder en langage ATL il faut l'installer à partir du bouton « Install

Modeling Components » (placé dans le menu principal) ; Ce bouton ouvre un menu qui vous permettra de choisir ATL. Validez la caisse à cocher comme le montre la figure 2, puis appuyez sur « finish » pour commencer l'installation.

Suite au téléchargement, vérifiez que la caisse à cocher ATL-SDK soit coché et puis appuyez sur « Next », lors de l'installation, révisez les détailles et pour les valider appuyez sur « Next », ensuite acceptez les conditions et appuyez sur « Finish »

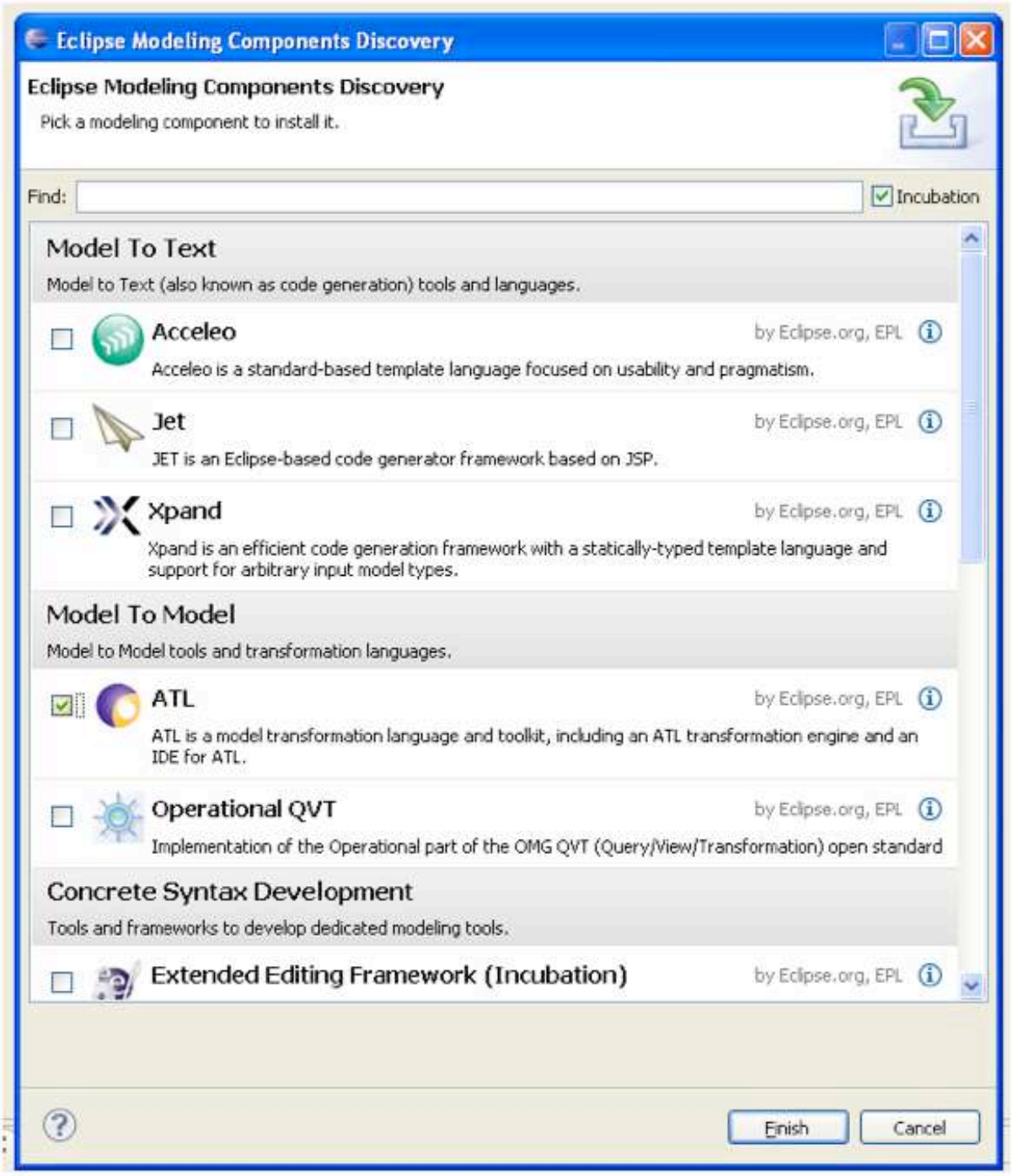


Figure 2 : Composants d'Eclipse Helios.

I.4 Démarrer Eclipse

Exécutez le fichier eclipse.exe du répertoire « eclipse », ensuite choisissez le workspace, c'est-à-dire le répertoire où seront stockés tous les projets à réaliser, puis appuyez sur « OK ».

La première fois qu'Eclipse est ouvert, choisissez d'aller sur la « workbench » pour aller directement à l'environnement de codage.

II. CREER UN PROJET

Le module d'Eclipse Helios permet plusieurs développements et n'est pas toujours utilisé que pour la transformation de modèles, il faut donc, lui mettre en évidence qu'il se dispose à faire de transformation. Allez dans le menu WindowsàOpen Perspectiveàother... à ATL.

II.1 Créer un projet ATL

Allez sur le menu principal FileàNewàOther... et choisissez « ATL project » comme l'indique la figure 3

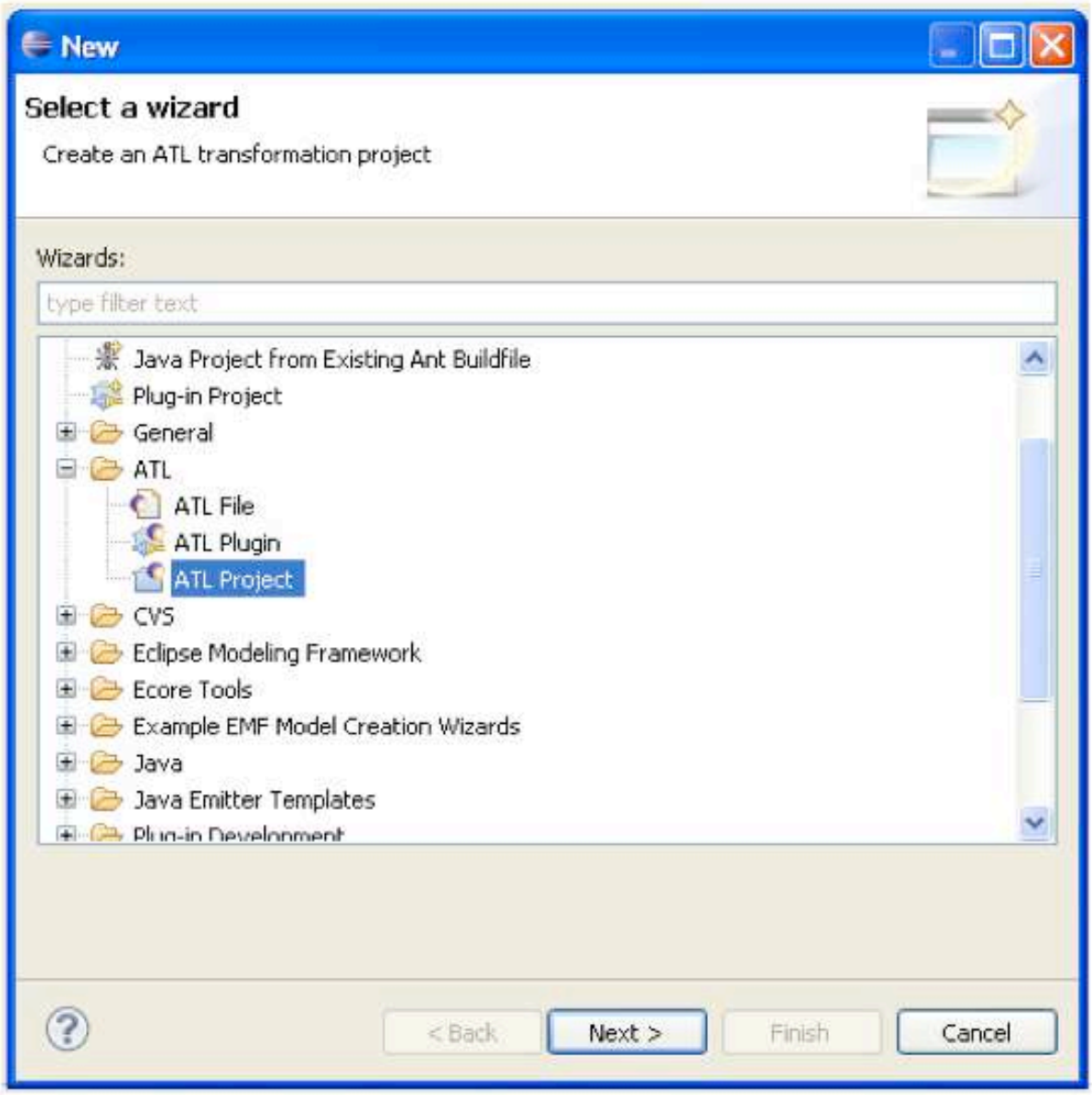


Figure 3 : création d'un projet ATL Ensuite, appuyez sur « Next » et assignez un nom au projet puis appuyez sur « Finish ».

Dans le menu de gauche, vous avez un nouveau répertoire qui porte le nom donnée au projet et qui contiendra tous les fichiers nécessaires pour la transformation.

III. COMMENT FAIRE UNE TRANSFORMATION

III.1 Insérer un méta-modèle

Placez vous dans le répertoire d'un projet, ensuite, avec le click droit de la souris, choisissez newàOthers...àEcore ToolsàEcore Diagram comme l'indique la figure 4.

Donnez un mon au diagramme et appuyez sur « Finish ».

Eclipse vous placera, automatiquement, sur un éditeur qui vous permet de designer un diagramme de class UML. Avec la palette de design, vous pouvez designer de classe, des associations, des conteneurs, héritage, etc. Par exemple, pour créer une classe, il faut appuyer, dans la palette, l'option « EClass » et ensuite, la designer sur l'espace de travail.

Dans le menu en bas, sur l'onglet « Properties » vous pouvez modifier le nom et les propriétés de la classe.

Pour ajouter des attributs, il faut choisir dans la palette, l'option « EAttribute », en suite faire click sur la classe à la quelle il appartient, puis dans l'onglet « Properties » vous pouvez customiser les attributs.

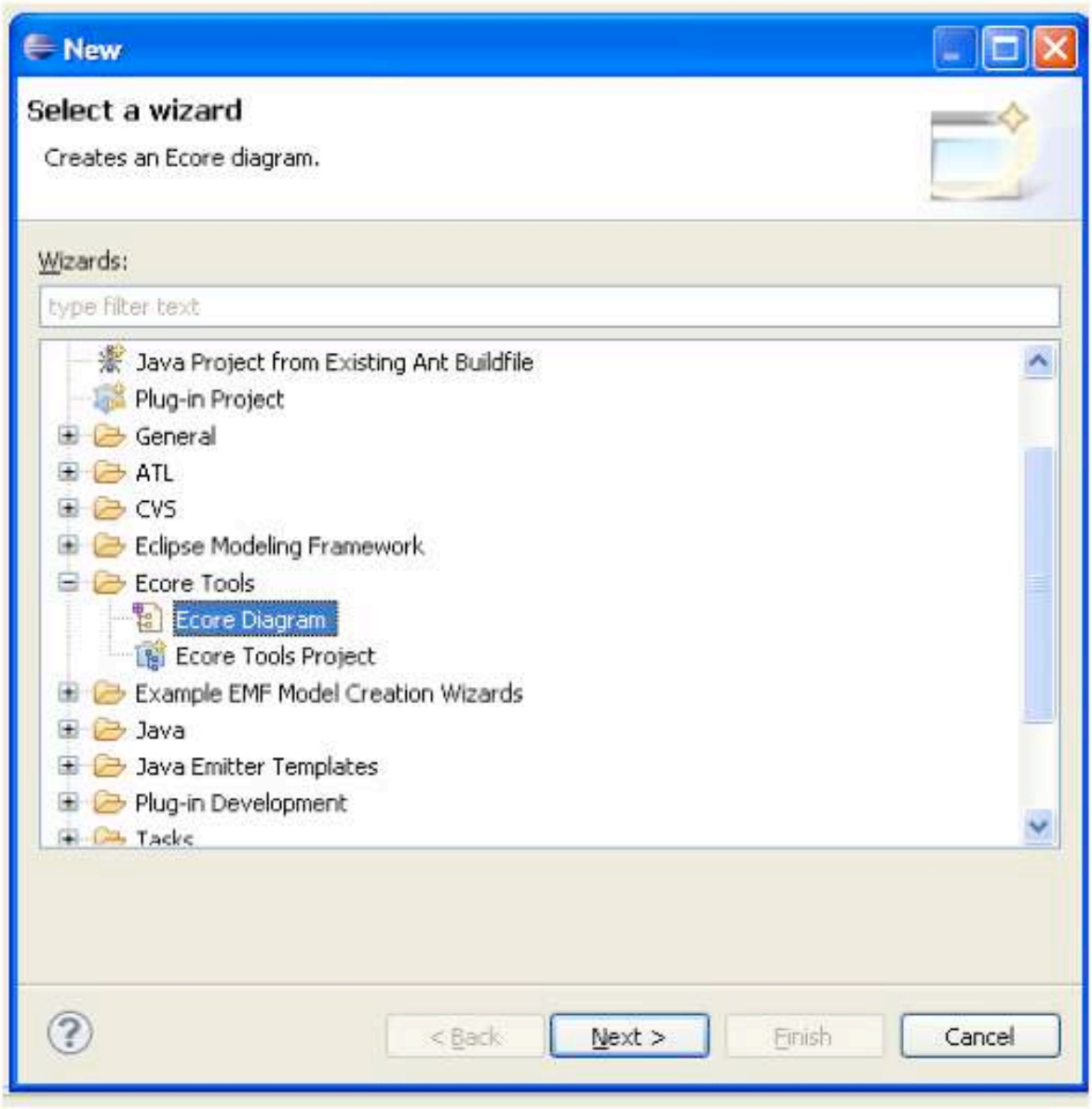


Figure 4 : création méta-modèle.

Voici un exemple d'un diagramme de classe.

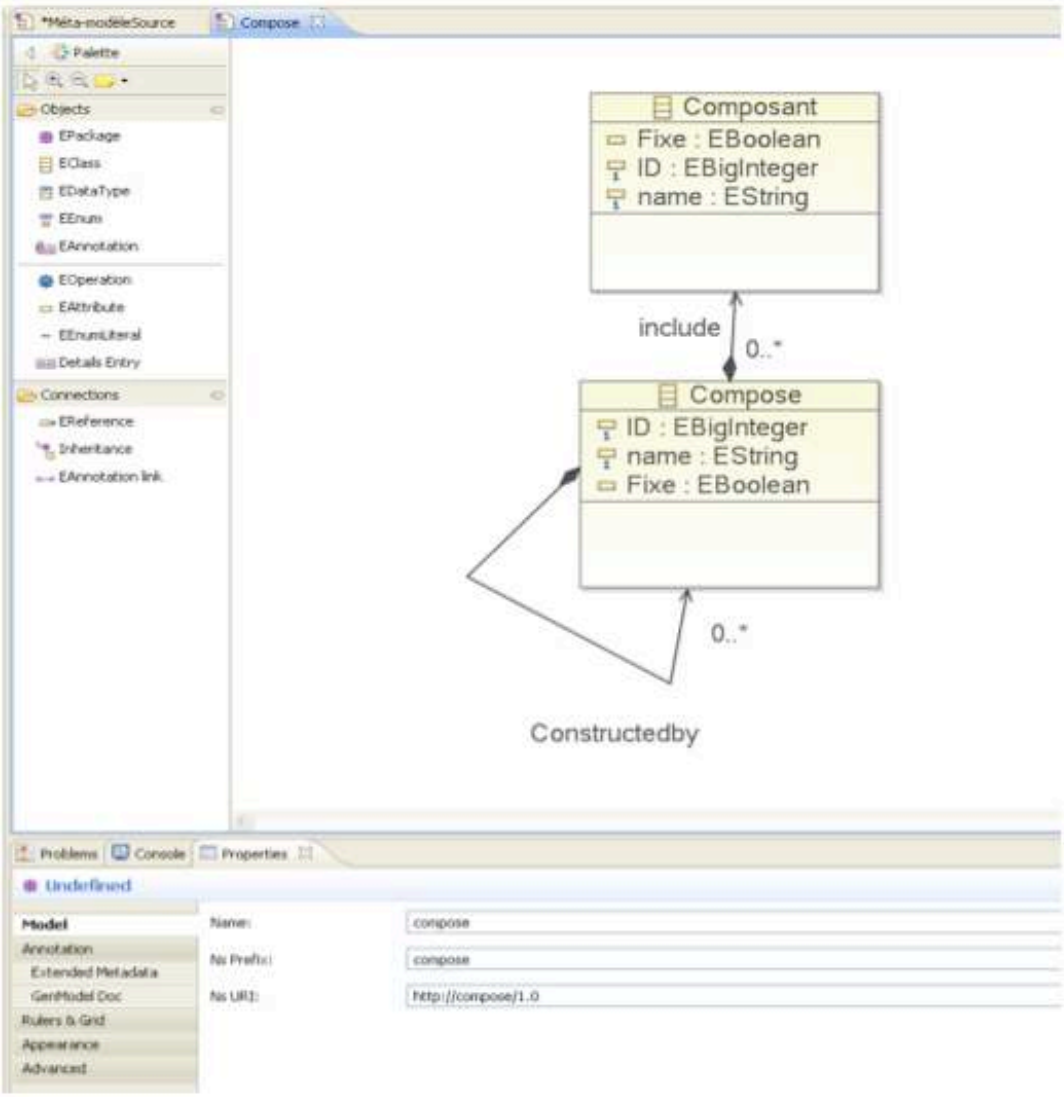


Figure 5 : Exemple Diagramme de classe.

Lors de la création d'un fichier « Ecore Diagram » Eclipse génère automatiquement un fichier .ecore qui décrit le méta-modèle dans un autre langage qui suit le Méta-Méta- Modèle d'Eclipse, ce qui vous permettrez de lui instancier, et de lui travailler en tant que modèle et ne pas comme un fichier normal.

III.2 Création d'une instance

Placez-vous sur le fichier .ecore antérieurement créé et ouvrez-le avec « Sample Ecore Model Editor ». Par exemple si vous ouvrez le fichier décrit pour le diagramme de figure 5, vous obtiendrez le résultat suivant :

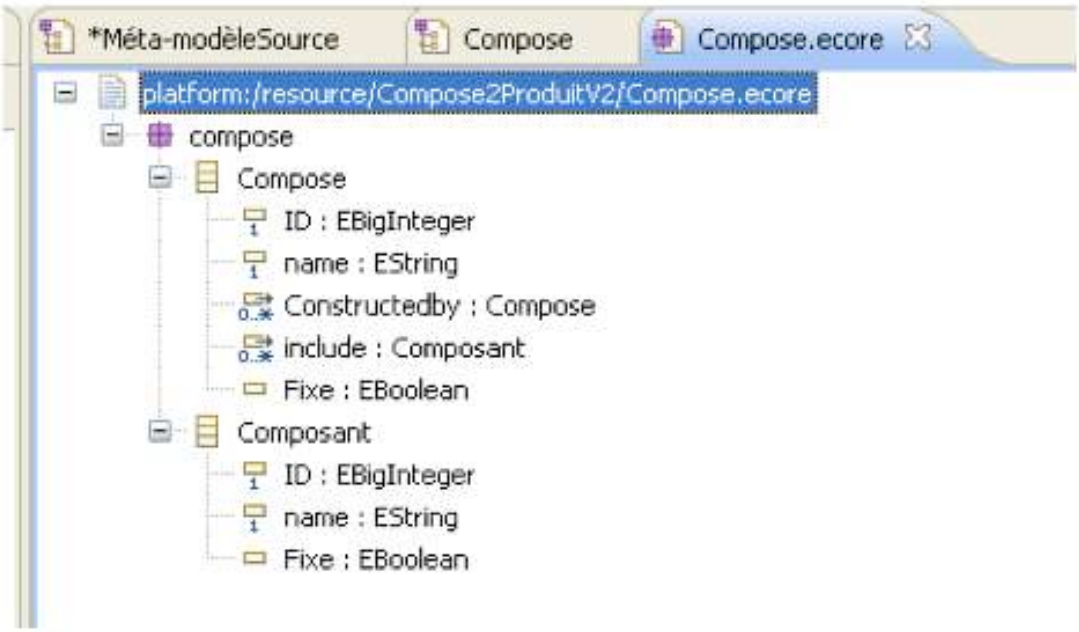


Figure 6 : modèle en format .ecore

Pour créer une instance, placez-vous sur la classe que vous souhaitez instancier, puis avec le click droit de votre souris choisissez « Create Dynamic Instance... » Comme l'indique la figure 7.

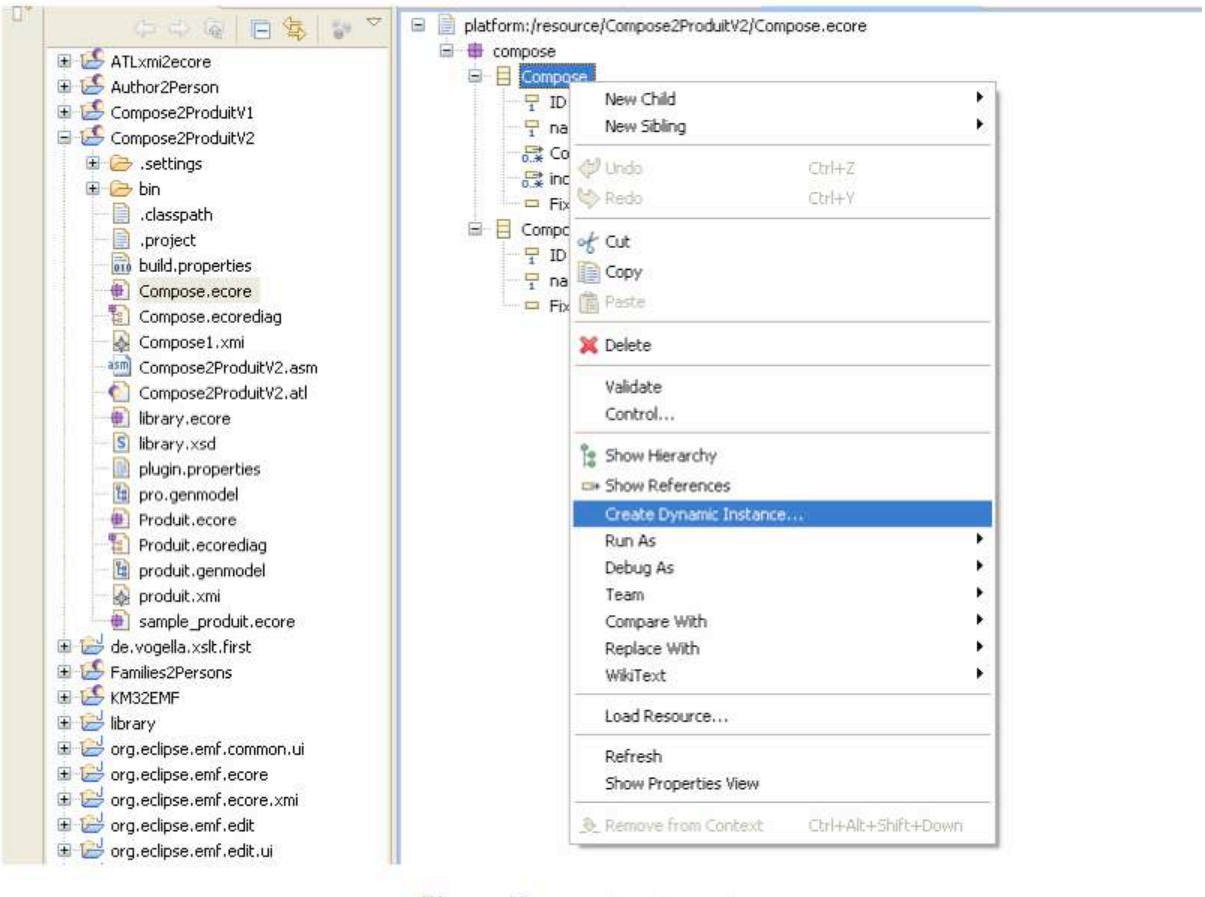


Figure 7 : création d'une instance.

Vous assignerez un nom et un répertoire conteneur pour fichier, par default une instance est toujours un fichier .xmi, c'est recommandé de garder ce format sauf si vous avez besoin d'obtenir un fichier avec un format spécial, ensuite, appuyez sur « Finish ».

Vous serez placé dans l'éditeur d'instances d'Eclipse (Sample Reflective Ecore Model Editor) qui vous permettra de donner une valeur à chaque attribut et d'instancier chaque association et classe qui sont en contact avec la première instance. Voici un exemple :

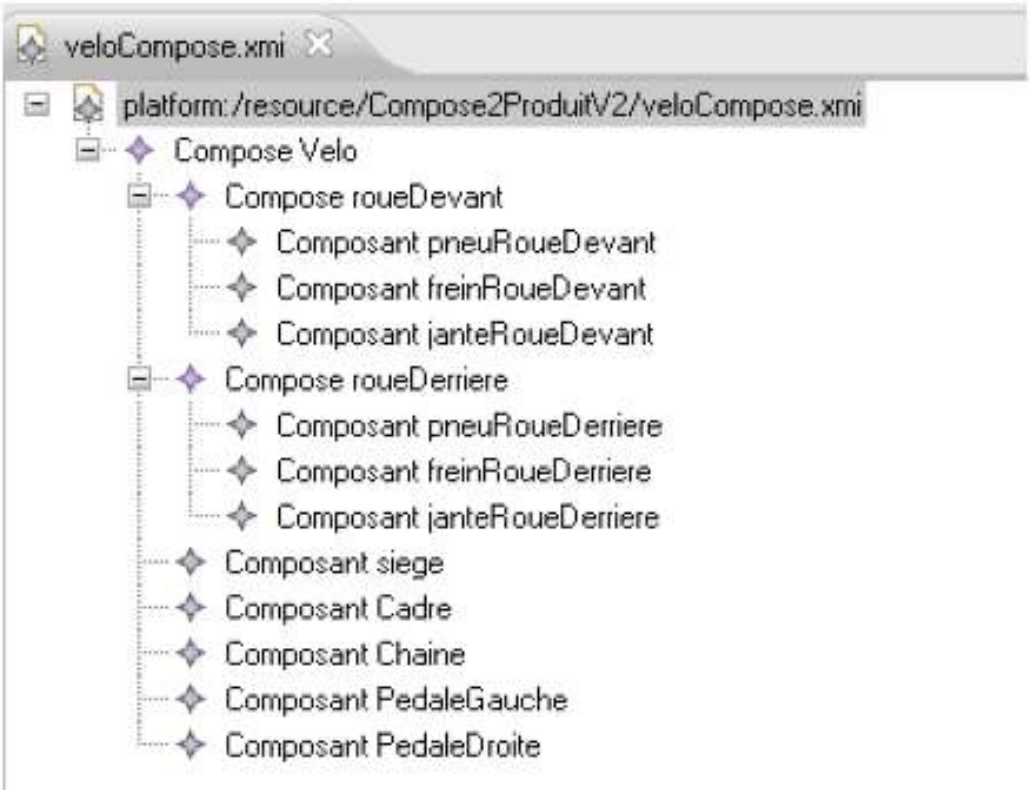


Figure 8 : Instance du modèle de la figure 5.

La façon pour utiliser cet éditeur est de faire click sur la première instance, celle qui est crée automatiquement, ensuite customiser les attributs dans l'onglet « properties », La figure 8 corresponde à une instance d'un vélo selon le méta-modèle de la figure 1.

Ensuite, faire click droit sur Vélo et choisir, l'option « New Child », ce que vous permettra d'instancier les associations liées au Vélo, il est montré une liste avec tous les

liens possibles, en fonction de la classe qui se trouve de l'autre coté de lien, une nouvelle instance sera créée alors.

Par exemple, VeloàNew ChildàConstructedBy vous permettra de créer une deuxième instance de Composé et de la lier au Vélo. Au fur à mesure, vous pourrez instancier tout le modèle.

III.3 Créer une transformation

La transformation de modèles dans ce document, est toujours en langage ATL, il faut donc, ajouter un fichier .ATL au répertoire racine du projet.

Placez-vous sur le répertoire du projet. Ensuite, faites click droitàNewàOther...àATLà ATL File comme l'indique la figure 9. Choisissez le nom de la

transformation et le répertoire où elle doit être stockée. Puis appuyez sur « Next ». Rentrez le méta-modèle source et cible ainsi que les libraires si vous en avez besoin. Ce formulaire vous aide à construire l'entête de la transformation facilement. Si vous le souhaitez, vous pouvez clicker sur finish dès que vous avez donné un nom pour la transformation et le répertoire où elle doit être stockée, et puis taper à la main l'entête de la transformation dans l'éditeur.

Ensuite vous serez placé sur l'éditeur d'ATL d'Eclipse, vous pouvez donc, taper le code de la transformation que vous souhaitez faire. Cet éditeur vous aidera dans la mesure où il souligne les mots clés et fait la fermeture automatique des accolades.

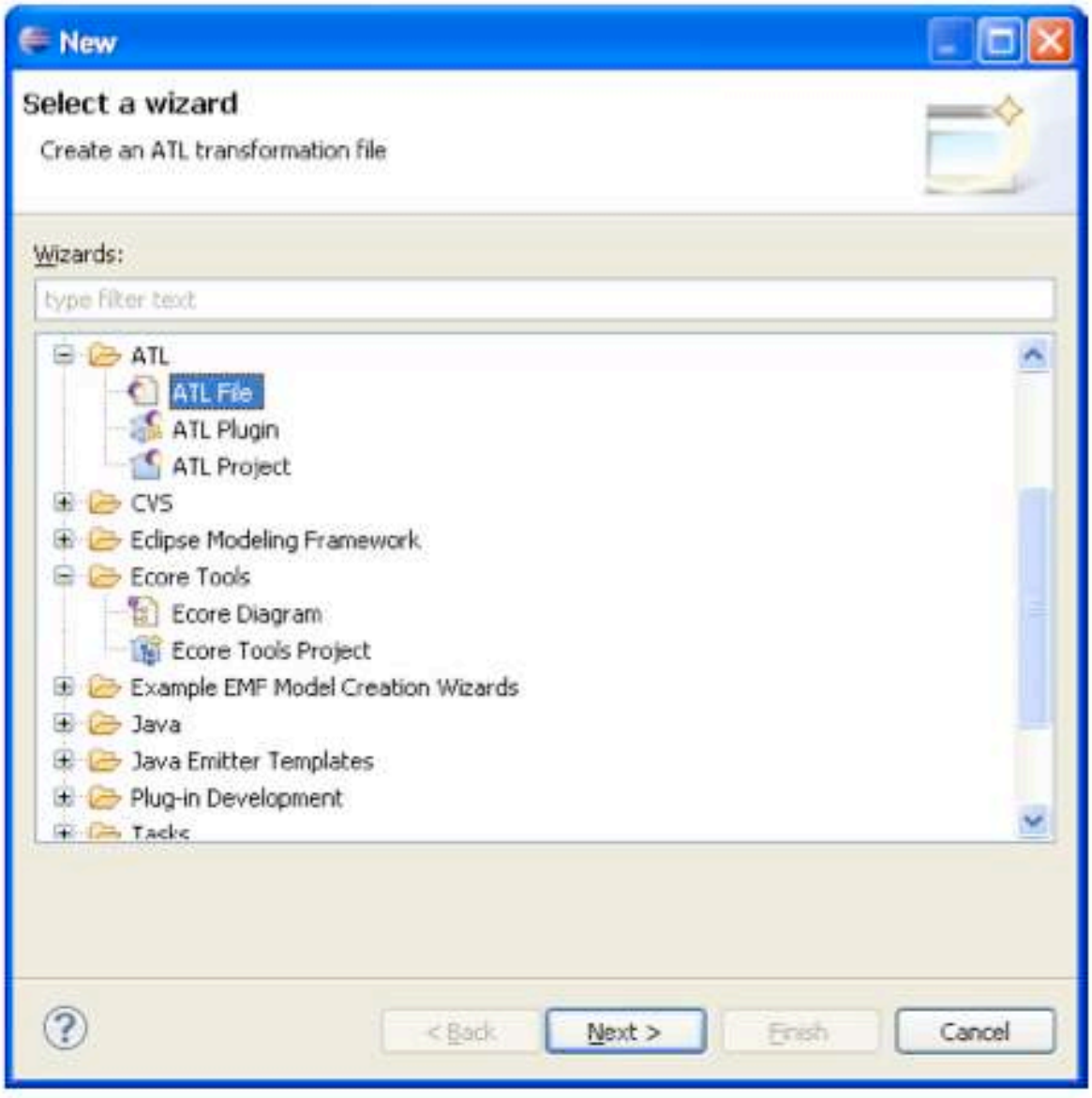


Figure 9 : Nouvel Transformation Pour vérifier si votre code est juste, c'est-à-dire pour le compiler, allez dans le menu

principal sur ProjectàBuild Project, les erreurs seront affichées dans l'onglet console.

III.4 Exécution du code ATL

Une fois le code ATL écrit et compilé, allez dans le menu principal sur RunàRun Configurations... Clickez sur « New launch configuration » pour créer une configuration d'exécution comme l'indique la figure 10.

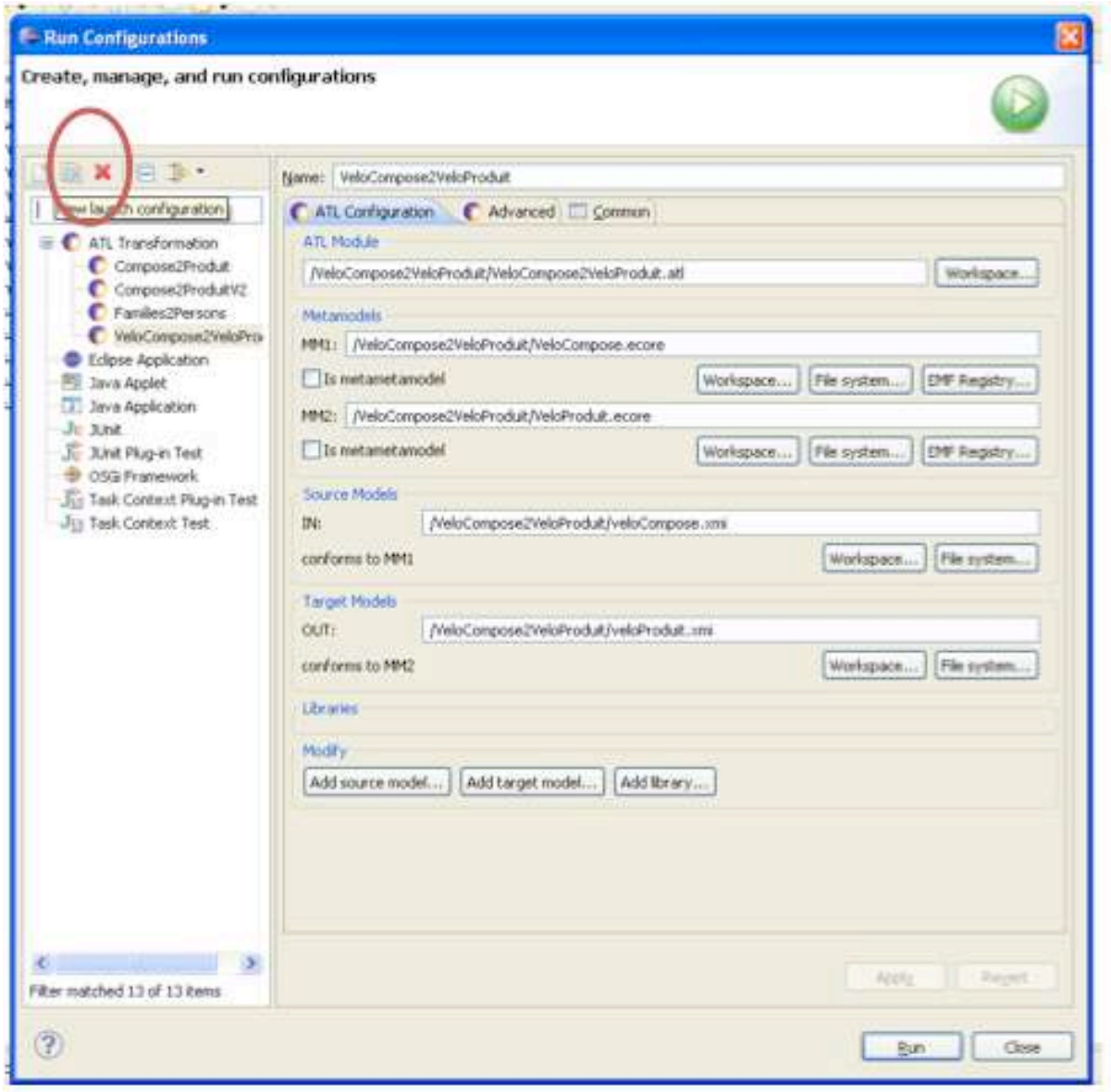


Figure 10 : créer une nouvelle configuration d'exécution

Ensuite, donnez un nom à votre configuration et choisissez le fichier .atl à exécuter avec le bouton « workspace » du champ « ATL module ». Ceci vous permettra de choisir les Méta-modèles source et cible de la transformation. Vous devez toujours choisir le fichier .ecore et non pas le .ecorediag.

Ensuite dans le champ « Source Model » choisissez le fichier d'instances (normalement le fichier .xmi), vous pouvez le choisir depuis le « workspace » ou autre.

Puis dans le champ « Target Model » choisissez le répertoire où sera stocké le résultat de la transformation et donnez-lui un nom avec l'extension .xmi (sauf si vous souhaitez trouver un autre type de fichier comme résultat).

Ensuite, enregistrez votre configuration avec le bouton « Apply » et puis « Run » pour l'exécuter. Le résultat se trouvera dans le répertoire antérieurement choisi.

Les codes d'Eclipse

```
-- @path MM2=/VeloCompose2VeloProduit/VeloProduit.Ecore -- @path
MM1=/VeloCompose2VeloProduit/VeloCompose.Ecore
module VeloCompose2VeloProduit; create OUT : MM2 from IN : MM1;
-- Ce helper renvoie l'ensemble de roues d'un vélo
helper context MM1!Velo def: getRoues() : OrderedSet(MM1!Roue) = self.mesRoues->iterate(r;
roues : OrderedSet(MM1!Roue) =
;
)
OrderedSet{}| if r.oclIsKindOf(MM1!Roue) then
else endif
roues.append(r) roues
-- Ce helper renvoie les pneus d'un ensemble roues


;
-- Ce helper renvoie la
helper context MM1!Velo OrderedSet(MM1!Jante) =
jante d'un ensemble roues
def : getJante(roues : OrderedSet(MM1!Roue)) :
if not r.maJante.oclIsUndefined() then jantes.append(r.maJante)
)
;
-- Ce helper renvoie le siège d'un vélo
helper context MM1!Velo def : getSiege() : MM1!Siege = if not self.monSiege.oclIsUndefined()
then
;
else
endif
self.monCadre false
;
else
endif
self.monSiege false
roues->iterate(r; pneus : OrderedSet(MM1!Pneu) = OrderedSet{} |
-- Ce helper renvoie le cadre d'un vélo
helper context MM1!Velo def : getCadre() : MM1!Cadre = if not self.monCadre.oclIsUndefined()
then
-- Ce helper renvoie la chaine d'un vélo
helper context MM1!Velo def : getChaine() : MM1!Chaine = if not self.maChaine.oclIsUndefined()
then
else
endif
self.maChaine false
;
-- Ce helper renvoi l'ensemble de pédales d'un vélo
else endif
pneus
roues->iterate(r; freins : OrderedSet(MM1!frein) = OrderedSet{} |
else endif
freins
roues->iterate(r; jantes : OrderedSet(MM1!Jante) = OrderedSet{} |
else endif
jantes
MM1!Velo def : getPedales() : OrderedSet(MM1!Pedale) = self.mesPedales->iterate(p; peds :
OrderedSet(MM1!Pedale) =
OrderedSet{} | if p.oclIsKindOf(MM1!Pedale) then
helper
context
else endif
)
peds.append(p) peds
;
--Ce règle transforme une instance de vélo qui suit le modèle de Velocomposé à une instance de
vélo qui suit le modèle de VeloProduit rule velo {
from v : MM1!Velo
to
e : MM2!Velo ( name <- v.name,
ID <- v.ID, structure <- thisModule.strFixe(v), articule <- thisModule.strMobile(v)
)
--Ce règle crée automatiquement la structure fixe d'un velo
lazy rule strFixe { from
}
```

```
}
}
to
v : MM1!Velo
s : MM2!Structure_Fixe ( monSiege <- thisModule.siege(v.getSiege()), monCadre <-
thisModule.Cadre(v.getCadre()) )
--Ce règle crée automatiquement la structure mobile d'un velo
lazy rule strMobile { from
to
v : MM1!Velo
s : MM2!Structure_Mobile ( mesPneus <-(v.getPneu(v.getRoues()))-
>collect(f|thisModule.Pneu(f)), mesFreins <-(v.getFrein(v.getRoues()))-
>collect(f|thisModule.Frein(f)), mesJantes <-(v.getJante(v.getRoues()))-
>collect(f|thisModule.Jante(f)), maChaine <- thisModule.Chaine(v.getChaine()),
)
mesPedales <- (v.getPedales())->collect(f|thisModule.Pedale(f)), mesRoues <- (v.getRoues())-
>collect(f|thisModule.Roue(f))
--Ce rule transform une instance de siège qui suit le modèle de Velocomposé à une instance de
siège qui suit le modèle de VeloProduit lazy rule siege {
from to
s : MM1!Siege
i : MM2!Siege ( ID <- s.ID,
name <-s.name
)
--Ce règle transform une instance de cadre qui suit le modèle de Velocomposé à une instance de
cadre qui suit le modèle de VeloProduit lazy rule Cadre {
}
from to
s : MM1!Cadre
i : MM2!Cadre ( ID <- s.ID,
name <-s.name
)
--Ce règle transform une instance de pneu qui suit le modèle de Velocomposé à une instance de
pneu qui suit le modèle de VeloProduit lazy rule Pneu {
}
from to
s : MM1!Pneu
i : MM2!Pneu ( ID <- s.ID,
name <-s.name
)
--Ce règle transform une instance de frein qui suit le modèle de Velocomposé à une instance de
frein qui suit le modèle de VeloProduit lazy rule Frein {
}
from to
s : MM1!Frein
i : MM2!Frein ( ID <- s.ID,
name <-s.name
)
}
--Ce règle transform une instance de jante qui suit le modèle de Velocomposé à une instance de
jante qui suit le modèle de VeloProduit lazy rule Jante {
}
from to
s : MM1!Jante
i : MM2!Jante ( ID <- s.ID,
name <-s.name
)
--Ce règle transform une instance de chaine qui suit le modèle de Velocomposé à une instance
de chaine qui suit le modèle de VeloProduit lazy rule Chaine {
from to
s : MM1!Chaine
i : MM2!Chaine ( ID <- s.ID,
name <-s.name
)
--Ce règle transform une instance de pédale qui suit le modèle de Velocomposé à une instance
de pédale qui suit le modèle de VeloProduit lazy rule Pedale {
}
}
}
```

Annexe I :

```
from to
s : MM1!Pedale
i : MM2!Pedale ( ID <- s.ID,
name <-s.name
)
--Ce règle transform une instance de roue qui suit le modèle de Velocomposé à une instance de
```

```
roue qui suit le modèle de VeloProduit lazy rule Roue {
from to
s : MM1!Roue
i : MM2!Roue ( ID <- s.ID,
)
-- @atlcompiler atl2006
module test; create OUT : Ecore from IN: Compose ;
helper context Compose!Compose def : isRacine() : Ecore!EPackage = if
      self.refImmediateComposite().oclIsUndefined() then
not
else
false true
endif; helper context Compose!Compose def : isClassNormal() : Boolean =
if
else
endif;
self.isRacine() then if self.refImmediateComposite().Constructedby->includes(self) then
not
else endif false
true false
helper context Compose!Compose def : getAllCompose() : Sequence(Compose!Compose)=
self.Constructedby->iterate(comp; elements : Sequence(Compose!Compose)=
Sequence{} | if  comp.Constructedby.oclIsUndefined() then -- si l'asociation
not
construtedby definie then...
elements.append(comp)->union(comp.getAllCompose())
else endif
elements.append(comp)
; helper context Compose!Compose def : getAllComposant() : Sequence(Compose!Composant) =
Compose!Composant.allInstances()->asSequence()
; helper def : getMyClass(s : String) : Ecore!EClass =
let allClass : Sequence(Ecore!EClass)= Ecore!EClass.allInstances()->asSequence() in
allClass.debug('class')->iterate(c ; res : Ecore!EClass = allClass-
>first() |
)
if c.name = s then c
)
; rule composepackage2Epackage {
from p1 : Compose!Compose to p2 : Ecore!EPackage(
(p1.isRacine())
else endif
res
nsURI <- 'http://velocompose/1.0', nsPrefix <- 'velocompose', name <- 'VeloCompose',
eClassifiers <- Sequence{}->append(rootClass)-
>union((p1.getAllComposant())->collect(d | thisModule.Composant2EClass(d))) -
>union((p1.getAllCompose().debug('tous'))->collect(f | thisModule.Compose2EClass(f))) ),
rootClass : Ecore!EClass( name <- p1.name,
eStructuralFeatures<-Sequence{}->union((p1.Constructedby)-
>collect(f|thisModule.Compose2EReference(f)))->
union((p1.include)->collect(d | thisModule.Composant2EReference(d))) )
} lazy rule Compose2EClass {
from comp : Compose!Compose (comp.isRacine()) to outClassCompose : Ecore!EClass (
name <- comp.name.debug('classCompose'),
eStructuralFeatures<-Sequence{}->union((comp.Constructedby)-
>collect(f|thisModule.Compose2EReference(f,outClassCompose)))->
union((comp.include)->collect(d | thisModule.Composant2EReference(d))) )
} lazy rule Composant2EClass {
from comp : Compose!Composant to outClassComposant : Ecore!EClass(
name <- comp.name.debug('classComposant')
)
} lazy rule Compose2EReference {
from comp : Compose!Compose (comp.isClassNormal()) to outEcore : Ecore!EReference(
name <- 'my' + comp.name, containment <- true, eType <- thisModule.getMyClass(comp.name)
)
} lazy rule Composant2EReference {
not
from comp : Compose!Composant to outEcore : Ecore!EReference (
)
}
name <- 'my' + comp.name.debug('nom'), containment <- true, eType <-
thisModule.getMyClass(comp.name)
-- @path Ecore=/Produit2Ecore/Ecore.ecore -- @path
Produit=/Produit2Ecore/Produit.ecore
module Produit2Ecore; create OUT : Ecore from IN : Produit;
--ce helper renvoie tous les articles liées a le structure fixe d'un produit
helper context Produit!Produit def : getAllArticles() :
```

```
Sequence(Produit!Article) =
if not self.Structure.oclIsUndefined() then let strF : Produit!StructureFixe =
self.Structure->first()
in
;
else
endif
strF.compose->asSequence() Sequence{}
--ce helper renvoie tous les articulateurs liées a le structure mobile d'un
produit helper context Produit!Produit def : getAllArticulateurs() :
Sequence(Produit!Articuleur) =
if not self.Articule.oclIsUndefined() then let strM : Produit!StructureMobile =
self.Articule->first()
in
;
else
endif
strM.fait->asSequence() Sequence{}
--ce helper renvoie la class d'arrivé d'une association.
helper def : getMyClass(s :  ) : Ecore!EClass = let allClass :
Sequence(Ecore!EClass)=
String
Ecore!EClass.allInstances()->asSequence() in allClass->iterate(c ; res :
Ecore!EClass = allClass-
>first() |
;
if c.name = s then c
else endif
res
)
--regle construit le package qui contient touos les classes.
rule produit2Package { from p : Produit!Produit to Epack: Ecore!EPackage (
nsURI <- 'http://veloproduit/1.0', nsPrefix <- 'veloproduit', name <-
'veloproduit', eClassifiers<-Sequence{}->append(produit)-
>union((p.getAllArticles())->collect(f|thisModule.Article2EClass(f)))- >
union((p.getAllArticulateurs())-
>collect(d|thisModule.Articulateur2EClass(d)))->
append(thisModule.StructureFixe2EClass(p.Structure->first()))-
>append(thisModule.StructureMobile2EClass(p.Articule->first()))
), produit : Ecore!EClass (
name<-p.name,
eStructuralFeatures<-Sequence{}-
>append(thisModule.Articule2EReference(p.Articule))->
append(thisModule.Structure2EReference(p.Structure->first())) )
}
--ce regle construit une classe à partir d'un article
lazy rule Article2EClass { from a : Produit!Article to class : Ecore!EClass (
}
}
)
)
name<-a.name
--ce regle construit une classe à partir d'un articulateur
lazy rule Articulateur2EClass { from ar : Produit!Articuleur to class :
Ecore!EClass(
name<-ar.name
--ce regle construit une Classe de la structure fixe.
lazy rule StructureFixe2EClass{ from strF : Produit!StructureFixe to class :
Ecore!EClass(
name <- 'StructureFixe',
eStructuralFeatures<-Sequence{}->union((strF.compose)-
>collect(f|thisModule.compose2EReference(f)))
)
}
--ce regle construit une Classe de la structure Mobile.
```

```
lazy rule StructureMobile2EClass { from strM : Produit!StructureMobile to class
: Ecore!EClass (
name<-'StructureMobile',
eStructuralFeatures<-Sequence{}->union((strM.fait)-
>collect(f|thisModule.fait2EReference(f)))
)
}
--Cette regle construit l'association entre un produit et son structure fixe
lazy rule Articule2EReference{
from strF : Produit!StructureMobile to ref : Ecore!EReference (
name <- 'Articule', containment <- true, eType <-
thisModule.getMyClass('StructureMobile')
)
--Cette regle construit l'association entre un produit et son structure Mobile
lazy rule Structure2EReference{
from strF : Produit!StructureFixe
to ref : Ecore!EReference ( name <- 'Structure', containment <- true,
eType <- thisModule.getMyClass('StructureFixe')
)
--Cette regle construit l'association entre la structure fixe et ses articles.
lazy rule compose2EReference {
}
from a : Produit!Article to ref : Ecore!EReference ( name <- 'My'+ a.name,
containment <- true, eType <-thisModule.getMyClass(a.name)
)
--Cette regle construit l'association entre la structure Mobile et ses
articulateurs. lazy rule fait2EReference {
from a : Produit!Articuleur to ref : Ecore!EReference ( name <- 'My'+ a.name,
)
containment <- true, eType <-thisModule.getMyClass(a.name)
}
}
```