Operational Framework Based on Modeling Languages to Support Product Repository Implementation

Muriel Pinel[1], Christian Braesch[1], Laurent Tabourot[1], and Aline Berger[2]

[1] SYMME, University of Savoy, Annecy, France
{muriel.pinel,christian.braesch,laurent.tabourot}@univ-savoie.fr
[2] THESAME, Annecy, France
ab@thesame-innovation.com

Abstract. Embracing Product Lifecycle Management approach involves integrating a product repository in the company information system. From customer's needs to disposal stage, several product representations exist. The product repository purpose is to secure consistency of one product representation with the others. This paper presents an operational modeling framework that supports product repository implementation. In order to ensure consistency, this framework identifies correspondences between entities of languages ("trade" languages and standard languages). The presented concepts are illustrated with correspondences between language entities of *product designed* and *product planned to be built* Bills of Materials.

Keywords: Product Lifecycle Management (PLM), Bill Of Materials (BOM), Model Driven Engineering (MDE).

1 Introduction

From customer's requirements to recycling or disposal stage, every product gets through several maturity stages: product and process designs, manufacturing, use, support, retire, etc. Each stage of the product lifecycle uses a "trade" product representation that meets specific stage objectives and needs.

For example, several Bills Of Materials (BOM) represent the product structure. Product design department defines *"the product designed BOM"*. This BOM meets the functional specifications. This BOM is composed of ***functional assemblies***, ***components*** and ***"is composed of"*** relationships. Basing on this BOM, process design department defines *"the product planned to be built BOM"*. This BOM meets the manufacturing, purchasing and workshop managing needs. This BOM is composed of ***purchased components and assemblies***, ***manufactured components and assemblies*** and ***"is made of"*** relationships. These two Bills Of Materials are product structures but theirs nodes and relationships do not refer to the same concepts.

Thus, numerous and various product representations exist throughout the product lifecycle. This multiplicity can be root of functional inconsistencies among stages. The purpose of this paper is to present an operational modeling framework that supports product repository implementation. This framework helps in translating a

L. Rivest, A. Bouras, and B. Louhichi (Eds.): PLM 2012, IFIP AICT 388, pp. 257–266, 2012.

representation described in a language into a representation described in an other language. This framework also helps in verifying that several representations describe the same product.

Industrial context is described in the first part (§2) in order to highlight the importance of consistency among product representations. These representations are managed into information system and are built basing on software models. Several methods exist to define software models. Information systems engineering methods and enterprise modeling results are first presented in the second part (§3). Then, an overview of product repository modeling approaches shows that existing methods are based on unified languages. These languages are difficult to implement in an extended enterprise context. So, a modeling framework is needed to manage correspondences between entities of several languages. The third part (§4) presents the ambivalence paradigm and a modeling framework. This framework is illustrated through correspondences between language entities of *product designed* and *product planned to be built* BOM.

2 Context and Identified Problems

Companies are facing a changing environment. PLM is one of the solutions adopted to meet new requirements and challenges.

2.1 Product Lifecycle Management

Environmental changes lead enterprises to define a PLM strategic approach based on existing PLM ideas, concepts ands tools. The PLM concept is recent and several definitions exist. According to different authors [1-5], PLM can be identified as a product repository with a set of functions used to manage information related to products. PLM's goal is to provide relevant information to stakeholders in each stage of the product lifecycle.

Every stage of the lifecycle takes part in the transformation of an idea into a physical product that can be used, maintained and recycled. The product lifecycle can be divided in two periods. First, the product only exists as a theoretical concept (product design, process design, etc.). Then, the product exists as a physical object (manufacturing, maintain, etc.). During the first period, each stage defines a theoretical representation of a more and more mature product. For example, the process design stage defines the *"product planned to be built BOM"* based on the *"product designed BOM"*. During the second period, one stage uses the requirements of representations provided by previous stages as input to manufacture or to modify the physical product. In order to meet traceability requirements, this stage may also record physical product properties into a representation. For example, the manufacturing stage builds a physical product based on the "*product planned to be built BOM*". The *"product built BOM"* can also be recorded.

Thus, the relationship between two lifecycle stages is a "Supplier - Customer" relationship. Every stage transforms input representation into its own product

representation (theoretical or record). This new representation is described in the specific world of discourse of the stage. We refer to this world of discourse as the "**trade language**" of the stage and we define the "**product repository**" as the set of product representations used throughout the product lifecycle.

The following figure illustrates possible distortions among product representations.

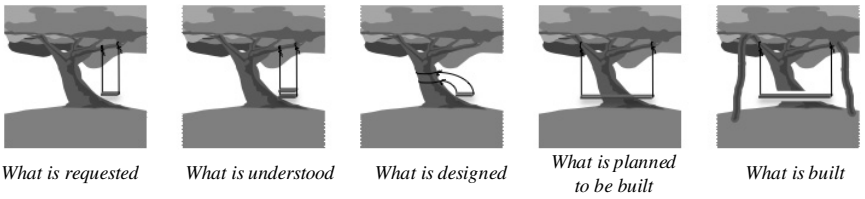

Fig. 1. Example of requirements distortions among the product representations

This figure identifies two categories of distortions.

The first category of distortion is related to the "Supplier - Customer" relationship between two stages. The "Supplier" provides the "Customer" a representation described in its specific "trade language". Before any transformation, the "Customer" has to translate the representation into requirements described in its own "trade language". Among the root causes of distortions, a first problem is identified (illustrated on the left in Figure 2): how can we ensure that the requirements described in the "trade language" (world of discourse) of one stage are well interpreted by the following stage which has another "trade language"?

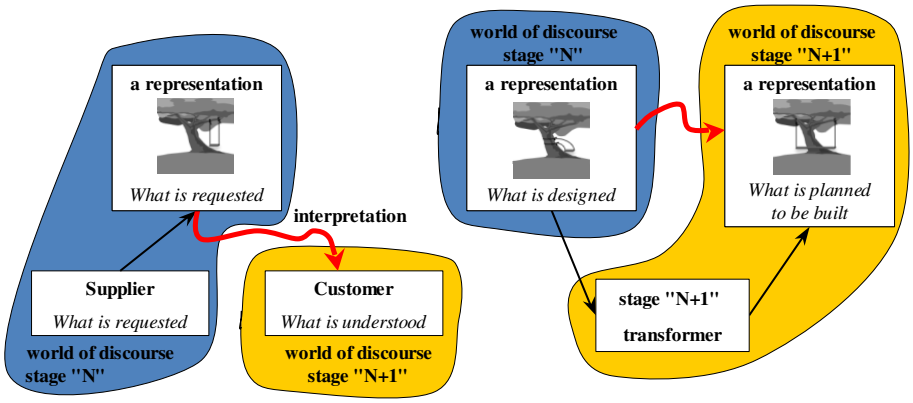

Fig. 2. Two distortion categories

The second category of distortion is related to the "transformation" of a representation. When a representation is built, new requirements are set out and a specific world of discourse is used. A second issue is identified (illustrated on the right in Figure 2): how can we ensure that two product representations with specific requirements described in two different "trade languages" (worlds of discourse) represent the same

finished product? that is to say: how can we ensure consistency among product representations described in several “trade languages”?

Thus, product repository has to provide mechanisms able to address these two issues.

2.2 Product Repository Models

A **model** represents a given aspect of a system (a product for example) and it is built with an intended goal [6-7]. Models are written using elements of an expression **language**. Symbols and syntax (organization of the symbols) of this language are represented in a **metamodel** [8]. The product repository objective is to provide users with product models suited to their specific needs. Products representations are defined by assigning values to model features; we call them **instances**.

In a company, all products representations of one stage are based on the same “trade model”. Figure 3 illustrates interactions between metamodel, language, model and instance. A product representation (**instance**) is built with the “**trade language” elements** of a product repository “**model**”.

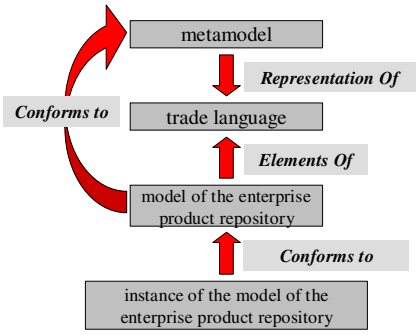

Fig. 3. Interactions between metamodel, language, model and instance (adapted from [8])

As a result, consistency among representations (instances) depends on consistency among product repository models. The second problem can be completed as follows: how can we ensure consistency among product repository models?

3 Information Systems Engineering or Enterprise Modeling?

Product repository models definition is based on methods of information systems engineering and on results of enterprise modeling. First, existing means to ensure interoperability among product repository models are presented. Then, an overview of existing methods for product repository modeling is given.

3.1 Information Systems Engineering and Enterprise Modeling

Product repository objective is to manage information related to products (cf. §2). Thus, product repository is an information system component of the company. Various time-tested methods of information systems engineering have been defined [9]. Most of these methods propose models adapted to a class of problems by integrating functional, structural and behavioral aspects of a field of study.

Identified problems in this paper relate to consistency among models, so Model Driven Engineering (MDE) [8], [10], [11] is particularly interesting. Models are more and more numerous and complex. MDE formalizes models and transformation rules. MDE objectives are to get a better understanding of the information system and to capitalize information system design knowledge. To do so, transformations between two models are done using model transformation language (ATL, ATLAS Transformation Language, for example) rather than programming language (Java for example). Therefore, MDE provide us a well-suited structure to address identified problems.

For each product lifecycle stage, one product repository model is defined in order to meet the stakeholder needs. Interactions among the product repository models can be defined basing on results of enterprise modeling [12]. Three major trends exist to ensure interoperability among models [13]. Firstly, specific enterprise architecture frameworks (for example, GERAM, Zachman, etc.) define relationships among models. Secondly, standards (for example, ISA95, STEP, etc.) define precisely concepts used in different models and ensure their interoperability. Thirdly, unified modeling languages for enterprise as UEML (Unified Enterprise Modeling Language) represent a company through different facets.

Standards are appropriate for exchanging data among the stakeholders involved in the product lifecycle in an extended enterprise context [14]. So, taking them into account in the modeling framework is essential.

3.2 Existing Approaches of Product Repository Modeling

Various methods exist to define product repository models of a company. Some methods use existing enterprise architecture frameworks (Zacham [15]) and some methods adapt existing frameworks to specific PLM needs (adaptation of GERAM framework [16]). Others define their own modeling framework to meet specific PLM objectives [17] [18]. Only one proposal [16] refers to a standard.

S. Zina [19] identifies two major ways to match several product representations, multi-views and multi-model approaches. Existing models [2], [16], [17], [18], [20], [21], [22] are multi-views approaches. These models provide a unified language to different “trades”. As a result, a “trade language” is a restriction of the unified language. Every stakeholder builds his own product representation with symbols and syntax available for him. However, in an extended enterprise context, product representations are spread among organizations. For example, a company defines the functional and “as designed” product representations. Its supplier defines the "as planned to be built" representation and records properties of physical products. In this context,

the adoption of a single unified language by all stakeholders working on a product is difficult. Moreover, unification involves the integration of all particularities in a "common mold" and this can cause ambiguities. For example, the word “item” can be used to describe an organ of the product designed, a raw material and a spare part. This can be source of confusion: when someone uses the word “item”, what is he talking about?

So, firstly, unified models have limitations and drawbacks to support collaboration among stakeholders from several trades. Secondly, proposed unified models do not use standards that ensure interoperability into the extended enterprise. Thus, the next paragraph presents an appropriate framework to address these issues.

4 Modeling Framework for the Product Repository

Product repository aggregates several trade models. Existing unified languages have limitations. In this paragraph, a modeling framework based on the ambivalence paradigm is presented to address the identified issues (cf. § 2). First, the paradigm and the framework are introduced. Then, their use is illustrated through the example of two trade product Bills Of Materials.

4.1 The Ambivalence Paradigm and the Modeling Framework

Ambivalence is the state of existing in two ways without ambiguity or opposition. The **ambivalence paradigm** considers that a representation can have several interpretations (problem 1). This paradigm considers also that an object can have several representations (problem 2).

Identifying correspondences among entities of “trade languages” avoid interpretation ambiguities (problem 1). These correspondences prevent also contradictions among product representations (problem 2).

Two ways exist to define correspondences between entities of two languages. In the first way, an interpreter translates a representation built in the source language into a representation built in the target language. In this case, every stakeholder takes only into account his own language. The interpreter masters source and target languages. In the second way, the translation is provided by a common language. In this case, every stakeholder translates his representations into the common language.

Modeling framework must be able to manage these two ways. Indeed, the framework has to ensure transformation between several stage models. Correspondences between entities of two “trade languages” have to be managed. In an extended enterprise context, common languages are used for exchanging data. So, the framework has to be able to manage correspondences between “trade language” and common language entities.

In order to build this framework, the first step is to identify the entities of the languages and the “trade” rules of consistency. These “trade” rules are specific to the company. Rules can be generic, for example: a component of the *product planned to be built BOM* can not be in a state “released” if the corresponding component into the

product designed BOM is not in the state "released". Rules can be also very specific to the company's products. The second step is to identify correspondences between languages, that is to say correspondences between vocabularies, syntaxes and semantics. The third step is to formalize metamodels of the "trade" and common languages. The entities of the metamodels must be choosen in order to be able to implement the identified "trade" rules into the Information System. The fourth step is to formalize correspondences into transformation rules between metamodels (cf. Figure 4). Finally, the framework is done and it can be used. The formalization of transformation rules among metamodels helps in translating a "trade model" into another "trade model".

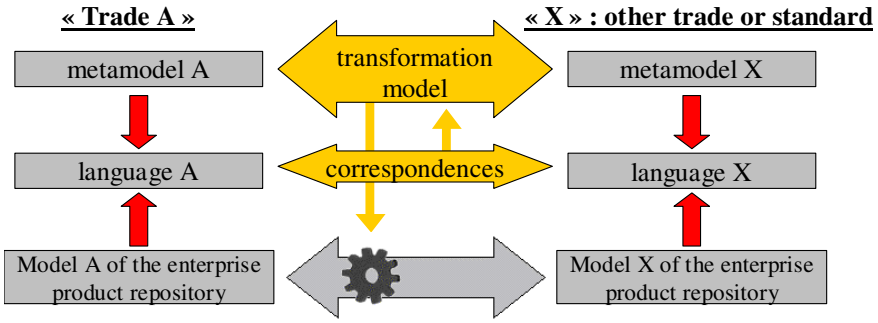

Fig. 4. Operational framework based on modeling languages

Using the same approach, formalization of transformation rules between models helps in translating a "trade representation" into another "trade representation". This formalization also helps in verifying that several representations describe the same product.

The framework is specific to each company. It can be implemented with a federerated architecture. Relationships between different Information System softwares (PDM, ERP…) can be done througth a MDE plateform as Eclipse [23]. Eclipse can be used to transform one product representation created with a software "A" into an other product representation able to be used by a software "B" or into a "standard" STEP representation [23]. On this platform, XML (Extensible Markup Language) is the format used for import and export and ATL is the language used to define the transformation rules.

4.2 An Example: Correspondences between Language Entities of Two BOM

In the introduction, two "trade" ways to describe a product structure were presented. The first one was the *"product designed BOM"*. This BOM meets functional needs and it is composed of functional assemblies, standard components and components. This product definition is usually managed into a PDM (Product Data Management) software. The second one was the *"product planned to be built BOM"*. This BOM

meets manufacturing, purchasing and workshop managing needs and it is composed of purchased components and assemblies, manufactured components and assemblies (in semi-finished or finished states) and raw materials. This product definition is usually managed into an ERP (Enterprise Resource Planning) software.

Correspondences to Avoid Ambiguities (Problem 1). To avoid ambiguity into the product repository, synonymies and polysemies among symbols of two trade languages have to be eliminated. Using a specific symbol (**vocabulary**) for each specific concept avoids confusion. For example, a functional assembly and a manufactured assembly do not have the same **semantic**. Specifying the symbol "assembly" adding the qualifiers "functional" and "manufactured" avoids confusion. Adapting ERP software data model is difficult. So specific symbols different from ERP ones have to be implemented into PDM software data model.

Correspondences to Avoid Contradictions between Requirements (Problem 2). The *"product designed BOM"* and the *"product planned to be built BOM"* have to represent the same finish product. Thus, correspondences have to be defined between organs, hierarchic relationships and quantities. These correspondences are used in two ways. The first one is to **support** the process design. For example, associations between generic "product designed" and generic "process and associate product planned to be built" can be done. This support can be implemented creating specific rules into tools as MPM (Manufacturing Process Management) software. The second one is to **verify** that the new built representation does not have any contradiction with the existing product representations. This verification can be done reworking and comparing BOM structures with specific algorithms. The transformations rules can be based on vocabulary, on semantic and on syntax of the languages. Possible associations (and associated cardinalities) between entities of the language describe rules based on syntax. Syntax rules can also be described literally.

Examples of Transformation Rules. The first example concerns "support" correspondences between organs based on vocabulary: *every standard component of the "**product designed BOM**" corresponds to a purchased component into the "**product planned to be built BOM**"*. An import program can create BOM into ERP database basing on a PDM export file. In such a case, identified rule can be implemented into the import program.

The second example concerns "support" correspondences between hierarchic relationships and quantities based on syntax rules described literally. In this example, the *"**product planned to be built BOM**"* is defined using a copy of the *"**product designed BOM**"* and reworking it. One transformation rule can be: *in the "**product planned to be built BOM**", if a relationship with a quantity **q** of component **X** is removed, others relationships must be created or modified to ensure that this quantity **q** of component **X** is preserved*. This rule can be implemented into a MPM software for example.

The third example concerns "verification" correspondences between hierarchic relationships and quantities based on syntaxic rules described literally: *if there are* ***n*** *times the elementary component* ***X*** *in the "****product designed BOM****", there must be* ***n*** *times this elementary component* ***X*** *in the "* ***product planned to be built BOM****".*

5 Conclusions and Future Work

Embracing Product Lifecycle Management approach involves integrating a product repository in the information system. This repository manages consistency among product representations used during the product lifecycle. Identifying and implementing consistency «trade» rules ensure this consistency. A modeling framework based on ambivalence paradigm and on model driven engineering (MDE) has been defined. Recent works [23-24] show that MDE approach seems to be a key factor to ensure product representations interoperability.

MDE provides mechanisms to implement consistency "trade" rules for vocabulary and syntax among several "trade languages". To complete the framework, our work is now focused on formalization of semantic relationships among "trade languages" by using ontologies and conceptual graphs.

Acknowledgements. The authors would like to thank Thesame, the French Competitiveness Cluster "Arve Industries Haute-Savoie Mont-Blanc" and the General Councils of Savoie and Haute-Savoie for supporting this research.

References

1. Stark, J.: Product Lifecycle Management: 21st century paradigm for product realisation. Springer (2005)
2. Sudarsan, R., Fenves, S.J., Sriram, R.D., Wang, F.: A product information modeling framework for product lifecycle management. Computer-Aided Design 37, 1399–1411 (2005)
3. Ameri, F., Dutta, D.: Product Lifecycle Management: Closing the Knowledge Loops. Computer-Aided Design & Applications 2, 577–590 (2005)
4. Terzi, S., Ball, P.D., Bouras, A., Dutta, D., Garetti, M., Gurumoorthy, B., Han, S., Kiritsis, D.: A new point of view on Product Lifecycle Management. In: Proceedings of the 5th International Conference on Product Lifecycle Management, PLM 2008, pp. 497–528 (2008)
5. Saaksvuori, A., Immonen, A.: Product Lifecycle Management, 3rd edn. Springer (2008)
6. Bézivin, J., Jouault, F., Rosenthal, P., Valduriez, P.: Modeling in the Large and Modeling in the Small. In: Aßmann, U., Akşit, M., Rensink, A. (eds.) MDAFA 2003. LNCS, vol. 3599, pp. 33–46. Springer, Heidelberg (2005)
7. Bézivin, J., Gerbé, O.: Towards a Precise Definition of the OMG/MDA Framework. In: Proceedings of the 16th Annual International Conference on Automated Software Engineering, San Diego, pp. 273–280 (2001)
8. Favre, J.M.: Towards a Basic Theory to Model Model Driven Engineering. In: Workshop on Software Model Engineering, joint event with UML 2004, Lisboa (2004)

9. Cauvet, C., Rosenthal-Saboux, C.: Ingénierie des systèmes d'information. Ed. Hermès in french (2001)
10. Caplat, G., Sourouille, J.L.: Considerations about Model Mapping. In: Workshop in Software Model Engineering, San Francisco (2003)
11. OMG: MDA Guide Version 1.0.1 (2003), http://www.omg.org/cgi-bin/doc?omg/03-06-01
12. Bernus, P., Nemes, L., Schmidt, G.: Handbook on enterprise Architecture, International Handbooks on Information Systems. Springer (2003)
13. Vallespir, B., Braesch, C., Chapurlat, V., Cretani, D.: L'intégration en modélisation d'entreprise: les chemins d'UEML. In: Proceedings of the 4th International Conference on Modeling, Optimization & SIMulation, Toulouse, pp. 140–145 (2003) (in French)
14. Rachuri, S., Subrahmanian, E., Bouras, A., Fenves, S., Foufou, S., Sriram, R.: Information sharing and exchange in the context of product lifecycle management: Role of standards. Computer-Aided Design 40(7), 789–800 (2008)
15. Bacha, R., Yannou, B.: New Approach for Building an Integrated Information System for Manufacturing Engineering Departments. In: Proceedings of MIM 2000: IFAC Symposium on Manufacturing, Modeling, Management and Supervision, Patras (2000)
16. Le Duigou, J., Bernard, A., Perry, N.: Framework for Product Lifecycle Management integration in Small and Medium Enterprises Networks. Computer-Aided Design and Applications 8, 531–544 (2011)
17. Gzara, L., Rieu, D., Tollenaere, M.: Product information systems engineering: an approach for building product models by reuse of patterns. Robotics and Computer Integrated Manufacturing 19, 239–261 (2003)
18. Terzi, S., Cassina, J., Panetto, H.: Development of a metamodel to foster interoperability along the product lifecycle traceability. In: Proceedings of the 1st Conference on Interoperability of Enterprise Software and Applications (ESA 2005), Geneva (2005)
19. Zina, S., Lombard, M., Lossent, L., Henriot, C.: Generic Modeling and Configuration Management in Product Lifecycle Management. International Journal of Computers, Communications & Control I, 126–138 (2006)
20. Noël, F., Roucoules, L.: The PPO design model with respect to digital enterprise technologies among product life cycle. International Journal of Computer Integrated Manufacturing 21, 139–145 (2008)
21. Labrousse, M., Bernard, A.: FBS-PPRE, an Enterprise Knowledge Lifecycle Model. In: Bernard, A., Tichkiewitch, S. (eds.) Methods and Tools for Effective Knowledge Lifecycle-Management, vol. 2, pp. 285–305. Springer (2008)
22. Matsokis, A., Kiritsis, D.: An ontology-based approach for Product Lifecycle Management. Computers in Industry 61, 787–797 (2010)
23. Iraqi-Houssaini, M., Kleiner, M., Roucoules, L.: Model-Based (Mechanical) Product Design. In: Whittle, J., Clark, T., Kühne, T. (eds.) MODELS 2011. LNCS, vol. 6981, pp. 548–562. Springer, Heidelberg (2011)
24. Moalla, N., Chettaoui, H., Ouzrout, Y., Noël, F., Bouras, A.: Model-Driven Architecture to enhance interoperability between product applications. In: Proceedings of the PLM-S4, pp. 380–392 (2008)