Ni, Qianfu and Lu, Wen Fang and Yarlagadda, Prasad K. (2008) *An extensible product structure model for product lifecycle management in the make-to-order environment.* Concurrent Engineering: Research and Applications (CERA), 16(4). pp. 243-251.

An Extensible Product Structure Model for Product Lifecycle Management in the Make-to-Order Environment

Q.F. Ni[a], W.F. Lu[b,*] and Prasad KDV Yarlagadda[a]

[a]*School of Engineering Systems, Queensland University of Technology, PO Box 2434, Brisbane,Q4001 Australia*

[b]*Centre for Design Technology/Department of Mechanical Engineering, National University of Singapore, 10 Kent Ridge Crescent, Singapore 119260*

Abstract

This paper presents a product structure model with a semantic representation technique that make the product structure extensible for developing product lifecycle management (PLM) systems that is flexible for make-to-order environment. In the make-to-order business context, each product could have a number of variants with slightly different constitutions to fulfill different customer requirements. All the variants of a family have common characteristics and each variant has its specific features. A master-variant pattern is proposed for building the product structure model to explicitly represent common characteristics and specific features of individual variants. The model is capable of enforcing the consistency of a family structure and its variant structure, supporting multiple product views, and facilitating the business processes. A semantic representation technique is developed that enables entity attributes to be defined and entities to be categorized in a neutral and semantic format. As a result, entity attributes and entity categorization can be redefined easily with its configurable capability for different requirements of the PLM systems. An XML-based language is developed for semantically representing entities and entity categories. A prototype as a proof-of-concept system is presented to illustrate the capability of the proposed extensible product structure model.

Keywords: Product structure; product data management; product lifecycle management; semantic representation

* Corresponding author. Tel.: +65 6516 1228; fax: +65 0000 0000.

E-mail addresses: q2.ni@qut.edu.au (Q.F. Ni), mpelwf@nus.edu.sg (W.F. Lu) and y.prasad@qut.edu.au (P. KDV Yarlagadda).

1 Introduction

Product structure, which is a hierarchical tree representing the classification of components that compose a product and the interrelationships of the components, is key information widely used by various business activities performed at different stages [1-3]. PLM is a strategic business principle to make product information consistent and sharable throughout an entire product lifecycle, and associate other relevant information created at each stage with product structures to serve the needs of downstream stages [4].

To provide customers with tailor-made products faster, better and cheaper, manufacturers have shifted their production mode to mass customization to take advantage of mass production for small batch-size production [5]. For such an environment, a product initially consists of a common base and modularized functional subsystems to form a customization platform [6]. Accordingly, essentially needed is a product structure model capable of flexibly representing product families and product variants with attention to different business processes in a product lifecycle. A good product structure model should be able to synchronize a family structure and its variant structures. At present, research generally attends to structure and represent detailed data related to a single product, and many product structure models are specifically developed for the different products. Product structure models considering product family and capable of supporting PLM rarely exists [7].

In implementation of PLM systems, much customization work is needed to tailor a system for a particular enterprise through redesign and redevelopment. Customization is necessary mainly because entities and entities attributes of the same entity to be managed vary from one

company to another due to the different business strategies. In PLM, different functional departments may have different needs to product representation. Ideally, a product structure model should be developed by taking into consideration all the requirements of different business processes. However, it is almost impossible [7]. Furthermore, due to the need to change their business strategies, new requirements might be required. A good approach to fulfill the need of frequently changing requirements is to develop a PLM system that enables enterprises to reconfigure the system when needed. The flexibility of a PLM system relies heavily on extensibility of a product structure model underpinning the system. The focus of this research is on the modeling of PLM system for a make-to-order environment. The extensible product structure model presented in this paper is one of key outcomes of the research.

2 Related research review

One essential function of PDM systems is to manage product structure [3]. However, few available PDM systems are powerful to manage product structures for mass customization because of the limitations of product structure models at representation of product families [8]. In addition, most product structure models underlying PDM systems lack of the ability to support integration of other business processes, such as planning and production [9, 10].

A few reports can be found on product structure models for representing product families. Sudarsan [11] presented a Product Family Evolution Model (PFEM) to address product family representation for the product information modeling framework. However, PFEM pays little attention to representation of common characteristics of a family and particular characteristics of a variant. Du [12] reported a product structure model to represent product family in the mass

customization context with functional view, technical view and structural view. This model is helpful for companies to shift from the individual product development to family-based design by providing a systematic method to establish a building block repository and configuration rules. It lacks of the ability to support design process management. Fujita [13] proposed a product structure representation by decomposing a product into different subsystems. The model only puts its focus on maximizing product varieties using minimum building blocks to achieve optimized a customization platform. Janitza [8] also reported a product model for mass customization by incorporating product decomposition and part specification into one model. This model provides a highly flexible product model specification for the product designer and simpler configuration for the customer. The family representation and variant representation has not received enough attention and synchronization of two representations is not addressed.

Based on the literature review, some of main research gaps in product structure modeling were identified, including 1) explicit representations of common characteristics of product family and specific features of product variants; 2) synchronization of a family model and its variant models in the context of mass customization; 3) integration of production structure and other business object models; and 4) extensibility of product structure models for flexible PLM systems. The product structure model reported in this paper attempts to fill these gaps to support development of a flexible PLM system for entire product lifecycle.

3 Abstract Product Structure Model

3.1 Master-variant pattern

To enable the model to effectively represent the common features of a family and special

features of different variants, a master-variant pattern, as shown in Fig. 1, is proposed for establishing the product structure model. In the model, the interfaces *IMaster* and *IVariant* are modeled to represent common properties and behaviors of families and variants respectively. The interface *IMVLink* represents common properties and behaviors of associations between masters and variants. The cardinalities of the association between *IMaster* and *IVariant* imply that one master can have one or unlimited variants and a variant should have and only can have one master. A master cannot exist without a variant, and vice versa. In this pattern, attributes common to all variants should be defined in master classes. Attributes specific to variants should be modeled in variant classes. In this pattern, *IMaster* is an abstract for grouping variants and represents common characteristics of a family while *IVariant* represents the special characteristics of individual variants.

In the model, the attributes *id* and *name* are defined to uniquely identify individual families. The attribute *version* is used to differentiate variants in a family. The model implies that all variants can share the same id and name and each variant can have a special name by defining the attribute *variantName* in the class *Variant*.

The master-variant pattern offers three main advantages: 1) it provides a clear boundary between the family representation and the variant representation. At the same time, it offers the capability of maintaining the data integrity; 2) it is capable of representing common characteristics of families and specific characteristics of individual variants; and 3) it can flexibly meet different requirements of different business processes. Masters or variants can be explicitly used as inputs to a business process and associated information can be explicitly linked to masters or variants.

3.2 Product structure model

Based on the master-variant pattern, the product structure model shown in Fig. 2 is developed. In the model, product, part and subassembly are represented by three groups of classes respectively: *Product*, *ProductVariant* and *ProductMVLink*, *Part*, *PartVariant* and *PartMVLink* as well as *Subassembly*, *SubassemblyVariant* and *SubassemblyMVLink*. The classes *Product*, *Part* and *Subassembly* represents product families, part families and subassembly families respectively while the classes *ProductVariant*, *PartVariant* and *SubassemblyVariant* represent product variants, part variants and subassembly variants.

3.2.1 Family Structure

For clarity of presentation, the family structure model in Fig. 2 is presented specifically in Fig. 3. In the model, aggregation associations between *Product* and *Part*, *Subassembly* as well as *StandardPart* implies that a product can consist of non-standard parts, subassemblies and standard parts. *Part* and *Subassembly* are master classes that represents a family rather than a specific product. Hence, the model shown in Fig. 3 only reflects what part families, subassembly families and standard parts are involved in a product family. It does not provide information about which variant of a part family or a subassembly family is involved in a product variant. However, based on the master-variant link, all part variants and subassembly variants are clearly reflected. Therefore, the family model provides an overall view of a product family about product variants and all optional part variants and subassembly variants. Such a overview is called product family spectrum [10].

Fig. 4 shows the example of the spectrum view of a simplified car family based on the proposed model. A car family, represented by *Car*:*Product*, can consist of an audio subsystem,

represented by *Audio*:*Subassembly*, and an engine, represented by *Engine*:*Part*. Further, an audio subassembly consists of a radio subsystem, represented by *Radio*:*StandardPart*, and a media player, represented by *MediaPlayer*:*Subassembly*. From the spectrum, it can clearly see that three types of engines with different rated powers and three types of audio subsystems, which are cassette player, CD player and video player, are available for selection.

The spectrum can effectively assist designers to configure products for customers, amend design to reorganize existing functions into configurable subsystems, design new alternative subsystems, or develop new functional subsystems to enhance customizability of a family. It can also help customers to configure products during the preparation of orders.

3.2.2 Variant Structure

A variant structure should clearly reflect what part variants and subassembly variants are used. At the same time, the model should be capable of enforcing the consistency of the family structure and variant structures. To achieve this goal, the variant structure model is built on the top of the family structure model. As shown in Fig. 2, *FPPLink* and *FPSLink* respectively represent associations of a product family with a part family and a subassembly family, and *FSSLink* represents association of a subassembly family with other subassembly families. To further represent variant structures, three association classes, i.e. *PPVersionLink*, *PSVersionLink* and *SSVersionLink*, are defined to associate *FPPLink* with *PartVaraint*, *FPSLink* with *SubassemblyVariant* and *FSSLink* with *SubassemblyVariant*. *PPVersionLink*, *PSVersionLink* and *SSVersionLink* are called version links and its key attribute is version. The value of this attribute indicates which product variant or subassembly variant that the associated variant is used for.

To explain the variant structure model, the relationships between a car variant and engine variants are taken as an example. As shown in Fig. 5, the car family has three variants, i.e. *CarA*, *CarB* and *CarC*, and the engine family also has three variants, which are *Engine1.8*, *Engine2.0* and *Engine2.2*. *Car* and *Engine* are associated through *CarEngineLink*, an instance of *FPPLink*. *FPPLink* is incapable of providing information about which engine variant is used for *CarA*, *CarB* and *CarC* respectively. To reflect the associations between the engine variants and the car variants, three version link instances are introduced, i.e. *EngineVersionLink1*, *EngineVersionLink2* and *EngineVersionLink3* to associate *Engine1.8*, *Engine2.0* and *Engine2.2* with *CarEngineLink* respectively. The attribute *version* in the version link classes plays the role of specifying which car variant each associated engine variant is used. From Fig. 5, it is clear that *Engine1.8* is used for *CarA* as the value of the attribute version of *EngineVersionLink1* is *CAR.A*, which should be the same as that of the attribute version of *CarA*.

Compared to the variant structure model, a significant advantage of this model is that the family structure model and the variant structure model are integrated. As a result, product variant structures can be well controlled by the corresponding product family structure. For example, if the engine family in Fig. 4 was not associated with the car family, *CarEngineLink* would not exist. Consequently, no engine variants could be associated with any product variants. This feature is very significant to companies which manage multiple families and there exist multiple subsystems that provide same functions, but are not exchangeable crossover families. For instance, two engine families are maintained for two car families respectively without exchangeability. While configuring products, this model can effectively prevent from selecting incompatible variants based on the family structure.

4 Lifecycle Management Support

4.1 Product view model

Users with different disciplines usually look into products from different prospective. For example, purchasing staff are only interested in the components which are to be purchased from suppliers. A production manager may only concern the components which are to be made or assembled internally. In product lifecycle management, a product should be represented in different ways to fulfill different needs in addition to product structure. These representations should be consistent with product structure, which completely reflects product constitution and relationships of constitutional components from the prospective of functions and structures [14]. Hence, the proposed product structure model is extended to support product views. A product view is a hierarchical representation to associate some of components of a production in different ways to fulfill needs of a specific stage in a product lifecycle. In product view management, an essential requirement is that a product should be independent of its product structure. However, it should be easily synchronized with product structure. As shown in Fig. 6, a reference mechanism is adopted to realize product views. A product view, represented by the class *ProductView*, consists of a set of instances of *PartRef* and/or *SubassemblyRef* organized in a hierarchical structure. As they are constructed using part references and subassembly references, product views are independent of a product structure. However, the reference mechanism enables product views to be linked back to product structure. Synchronization between product views and the corresponding product structure can be achieved. A reference is a pointer which does not contain the actual data of a part or a subassembly. Therefore, no duplications of data exist and data consistency can be easily maintained. A product can have

multiple views, such as manufacturing view, bill of material view and engineering change view. Categorization of product views is realized based on view roles, represented by the link class *ViewRole*. The ability to support product views enables the model to better support PLM.

4.2 Integration with other processes

The proposed product structure model differentiates standard parts and non-standard parts. Standard parts are purchased from suppliers. Non-standard parts may go through other business processes, such as production process or outsourcing process.

The interfaces *IStockable*, *IPurchasable* and *IOutsourcable* in Fig.2 are modeled to enforce the implementing classes to comply with the processing rules of stock management, purchase management and outsourcing management. The implementation of *IStockable* by variant classes, i.e. *ProductVariant*, *PartVariant* and *SubassemblyVariant*, implies that common parts, subassemblies and even products are allowed to be made to stock. Further, it enables make-to-order and make-to-stock decision to be made at a variant level. As a result, in a part or subassembly family, variants commonly demanded can be made to stock while variants only demanded by a few of customers may be particularly made when being ordered. The model is able to support the main objective of mass customization by taking advantage of volume production and also able to deliver tailored products for customers.

5 Extensibility and Semantic Representation

5.1 Concept of semantic representation

The product structure model discussed above is rather abstract. To make it useful for PLM systems, the model needs to be extended according to industrial sectors. The object-oriented

approach to derive a specific model based on an abstract model is called generalization, which is a process to define subclasses by extending abstract classes to represent specific types. For example, the subclasses *Shaft* and *Gear* may be defined by extending the abstract class *Part* to represent shafts and gears. However, companies in different industrial sectors have different types of products, parts and subassemblies. Even companies in the same industrial sector may categorize these items in different ways due to the difference of business practices. The identification of subclasses and essential attributes of each subclass might be difficult at the stage of creating a specific product structure model. In addition, specific models are usually established at the design stage of system development and are built into a PLM system. In such a way, any changes to a model will cause redeveloping the system. Such a PLM system also lacks of the flexibility to support changes of business strategies.

It is imperative to develop extensible product structure model for developing flexible PLM systems that can be easily specialized for a particular company with minimized redesign and redevelopment. Essentially, an extensible product structure model should be represented in a semantic approach and loosely coupled with system codes. In such a system, a semantic product structure model serves as an instructor who guides and controls execution of system codes. In turn, system codes act as executives to carry out information processing by interpreting instructions in a semantic product structure model. When the models are changed or replaced, the codes can manipulate information according to new models. Therefore, PLM systems based on a semantic product structure model is highly flexible and reconfigurable and they can be easily deployed to different companies, even in different industrial sectors.

5.2 Semantic representation framework

Semantic representation for the developed product structure model is shown in Fig. 7. A set of attribute definition is aggregated to define the entities of products, parts or subassemblies. An attribute definition has a *AttributeSpecification* associated, which defines attribute name, type and title. An attribute can be further characterized by providing a default value, value sources, value validation rules and other constraints. A value source contains information for determining value candidates of an attribute. A value validation defines rules as an expression for verifying and validating a value to be assigned to an attribute. The model separates the default value configuration and descriptive title configuration from the attribute declaration. This enables users with different knowledge work together to compose semantic models. In system implementation, people who have detailed knowledge of the system can work on attribute declarations while users with general knowledge can define default values and titles. The model provides a text resource configuration for defining texts in key-value pairs. Text entries in the text resource configuration are referred by other configurations using keys.

5.3 XML-based representation language

Fig. 8 shows a simplified XML-based product entity model. The tag *entity* is used to declare an information entity. The tag *attribute*, which is nested to the tag *entity*, is introduced to declare entity attributes. Attribute name, data type and display title are compulsory information in the definition of attributes. Data type and display title are defined by the nested tags *type* and *title*. The tag *extended* is employed to indicate an attribute is extended if the tag value is “true”, or a built-in one if the tag value is “false”. The tag *attribute* can have a nested tag *deprecated*.

Value constraints can be defined for individual attributes and need to be defined in

accordance with data types. For example, legal constraints for the text type include: 1) maximum length of a value; 2) only upper cases or lower cases are accepted; and 3) whether spaces are permitted. Complicated validation conditions can be defined in value validation. A validation condition is an expression represents a validating logic. Three types of value sources are supported. The constant value source defines a set of constants as the candidate values of an attribute. The query value source provides information for construct SQL statements to query objects as candidate values. The navigation value source provides information for acquiring objects associated with the current object as the candidate values of an attribute.

5.4 Semantic category representation

The approach of creating different subclasses to represent different types of products, parts and subassemblies results in a rigid specific model. Therefore, this paper develops a semantic category representation for flexible categorization. As shown in Fig. 9, the semantic category representation represents categorization in a hierarchical format. The tag *category* defines a category using three parameters: *key*, *title* and *schema*. The attribute *schema* contains a keyword pointing to a group of attribute definitions associated with the category. A collective category, such as gear, can have sub-categories, such as cylindrical gear and conical gear, which are represented as nested elements of the collective category. The attributes defined for a collective group will be inherited by all its sub-categories. Apart from attributes defined in the class *PartVariant*, instances of *CylindricalGear* and *ConicalGear* also have attribute *teethNumber* which is defined for the category *gear*. At the same time, *CylindricalGear* and *ConicalGear* instances have specific attributes respectively to characterize cylindrical gears and conical gears. This approach does not require identifying all subclasses at a design stage as

categories and category-related attributes can be defined even after system development.

6 Prototype

A prototype system with functions for project management, product configuration management and inventory management has been developed based on the proposed product structure model and semantic representations. Fig. 10 shows the multi-tier and web-based architecture of the system. The kernel in the architecture is the system services, which are organized into three layers: foundation layer, functional layer and domain layer.

On the foundation layer, the entity service is responsible for managing information entities by taking into consideration semantically defined attributes and category-related attributes. While creating instances of product and entities, the service checks the semantic definitions and consolidates both entity-related and category-related semantic attributes defined. The relationship service provides functions for managing entity relationships. This service also ensures that a master at least has one variant associated and no variant can exist without a master. The persistence service acts as a gateway of database access. It maps information entities to corresponding tables while storing information entities and instantiates appropriate objects while retrieving information from database. These three services work together to make transparent the master-variant concept and semantic representations to other services.

The functional layer offers common functions, such as document management to support the domain layer. The document service is responsible for associating various documents, such as engineering drawings, with products and parts. Its main responsibility is to wrap documents as binary objects and associate the document with an object – a document owner by leveraging

the relationship service. The report service provides a template-based method to generate various reports for information exchange and sharing. By cooperating with the document service, reports can be associated with other objects, such as projects, products or parts.

The domain services are developed to provide functions to integrate and manage business processes. Functions for project management include project initialization, project schedule and progress tracking. In the make-to-order environment, there are internal projects and external projects. Internal projects are initialized to manage family design and plan the production of common parts, subassembly and functional subsystems. External projects are created based on customer orders to fulfill customer requirements by cooperating with the inventory service and the resource service. In general, internal projects are managed based on family structures while external projects work on variant structures. The product service provides the capability to manage product family structures, variant structures, part families, subassembly families and a standard part library to assist product configuration, process planning and workshop task generation. Fig. 12 shows the interface of a family structure view with two families, i.e. a car family and a truck family. The inventory service manages stocks of common parts and commonly demanded variants. The resource service manages capacities and capabilities of resources, such as machines, materials, and operators to support design task management, process planning and workshop task management.

7 Conclusion

PLM systems are complex and the implementation is costly and time-consuming with potential failures. Our research focuses on the modeling and designing a PLM system which

can be easily configured with minimum redesign and redevelopment for different companies and reconfigured for new business opportunities even after the system is developed. This paper presents the key outcome of the research, an extensible product structure model, which is critical to PLM systems with flexibility for the make-to-order environment.

In this paper, a master-variant pattern is developed to establish such a product structure model to represents common characteristics of a product family and particular characteristics of a product variant. This model is capable of maintaining a clear boundary between product family structures and variant structures as well as consistency of a family structure and its variant structures. The model is also able to support integration with other business processes.

To make the product structure model extensible, a semantic representation technique has proposed to represent entity definition and entity categorization. The semantic categorization representation enables to flexible define categories and a set of attributes can be defined semantically and associated with each category. This technique enables to extend the abstract product structure model to a specific model for developing PLM systems with flexibility.

A prototype system for proof-of-concept is developed to demonstrate the proposed extensible product structure model and the semantic representation techniques to support product configuration. In particular, this prototype with multi-tier and web-based architecture is developed to show the capabilities of the proposed modelfor the flexibility of a PLM system.

References

[1] W. He, Q. F. Ni, X. Ming, and W. F. Lu, "Product Structure Management for Enterprise Business Processes in Product Lifecycle," Proceedings of 11th ISPE International Conference on Concurrent Engineering, Beijing, China, 2004.

[2] T. Mannisto, H. Peltonen, A. Martio, and R. Sulonen, "Modelling generic product structures in STEP," *Computer-Aided Design*, vol. 30, pp. 1111-1118, 1998.

[3] B. Eynard, T. Gallet, P. Nowak, and L. Roucoules, "UML based specifications of PDM product structure and

workflow," *Computers in Industry*, vol. 55, pp. 301-316, 2004.

[4] G. Thimm, S. G. Lee, and Y.-S. Ma, "Towards unified modelling of product life-cycles," *Computers in Industry*, vol. 57, pp. 331-341, 2006.

[5] Q. Ni, X. Ming, and W. F. Lu, "Computer-Supported Collaborative Environment for Distributed Product Development," Proceedings of International Conference for Agile Manufacturing, Beijing, Chian, 2003.

[6] B. MacCarthy, P. G. Brabazon, and J. Bramham, "Fundamental modes of operation for mass customization," *International Journal of Production Economics*, vol. 85, pp. 289-304, 2003.

[7] Q. Shu and C. Wang, *Information Modeling for Product Lifecycle Management*, 183 ed, 2005.

[8] D. Janitza, M. Lacher, M. Maurer, U. Pulm, and H. Rudolf, "A product model for mass-customisation products," *Lecture Notes in Computer Science*, vol. 2774, pp. 1023-1029, 2003.

[9] A.-P. Hameri and J. Nihtila, "Product data management--exploratory study on state-of-the-art in one-of-a-kind industry," *Computers in Industry*, vol. 35, pp. 195-206, 1998.

[10] W. He, Q. F. Ni, and B. H. Lee, "Enterprise Business Information Management System based on PDM Framework," presented at IEEE International Conference on Systems, Man & Cybernetics, Washington, D.C., USA, 2003.

[11] R. Sudarsan, S. J. Fenves, R. D. Sriram, and F. Wang, "A product information modeling framework for product lifecycle management," *Computer-Aided Design*, vol. 37, pp. 1399-1411, 2005.

[12] X. F. Du, J. X. Jiao, and M. Tseng, "Architecture of Product Family for Mass Customization," presented at IEEE International Conference on Management of Innovation and Technology, 2000.

[13] K. Fujita, "Product variety optimization under modular architecture," *Computer-Aided Design*, vol. 34, pp. 953-965, 2002.

[14] F. Fuxin, "Configurable product views based on geometry user requirements," *Computer-Aided Design*, vol. 37, pp. 957-966, 2005.

[15] Z. Zhang, M. K. O. Lee, P. Huang, L. Zhang, and X. Huang, "A framework of ERP systems implementation success in China: An empirical study," *International Journal of Production Economics*, 2004.

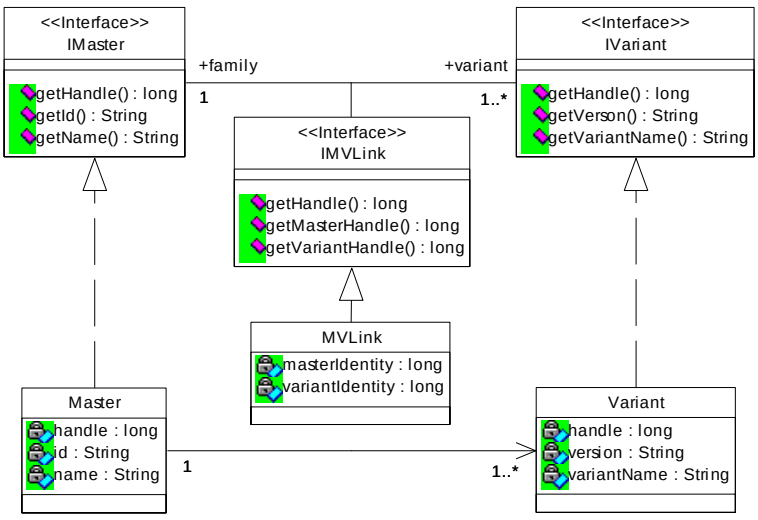

Fig.1 Master-variant pattern

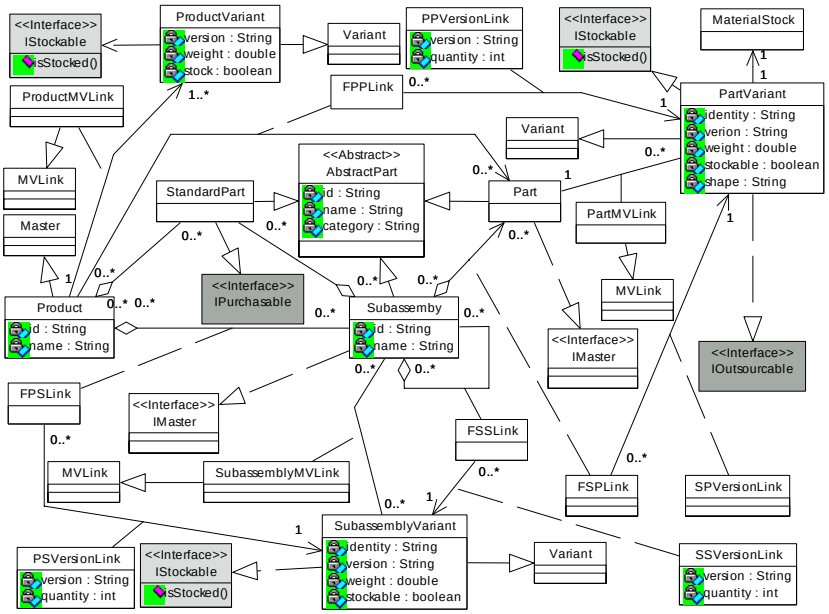

Fig.2 Product structure model

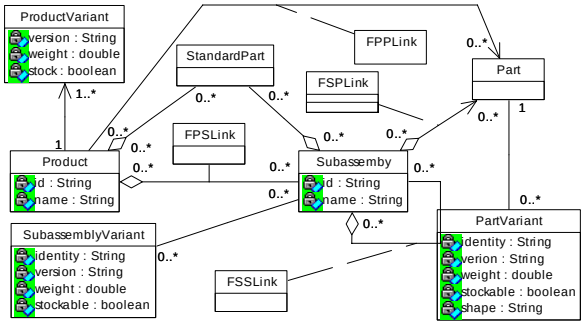

Fig.3 Product family model

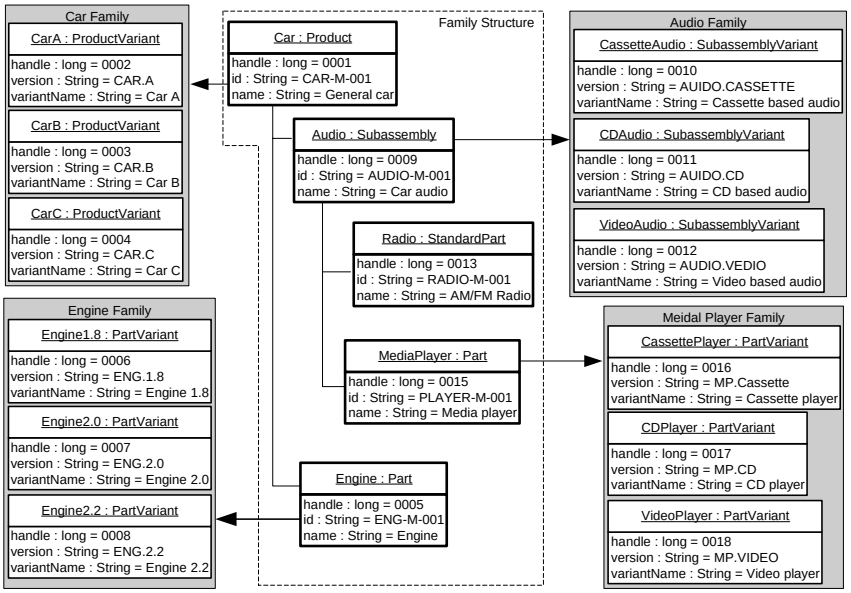

Fig.4 A simplified car family spectrum

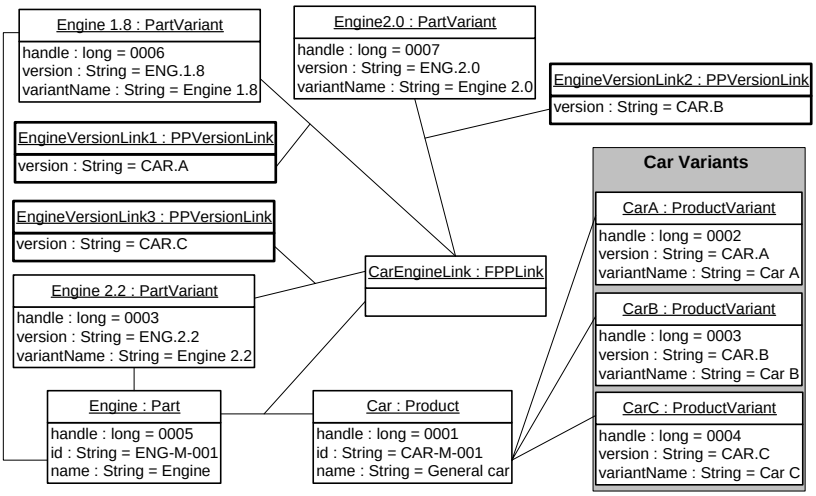

Fig. 5 Relationship between a car variant and an engine variant

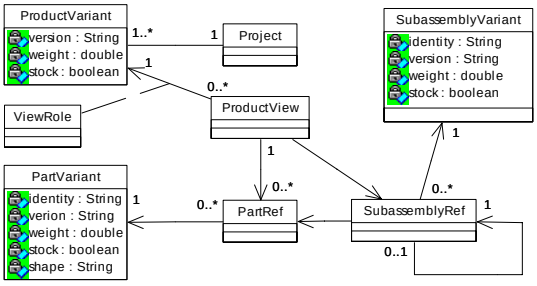

Fig. 6 Product view model

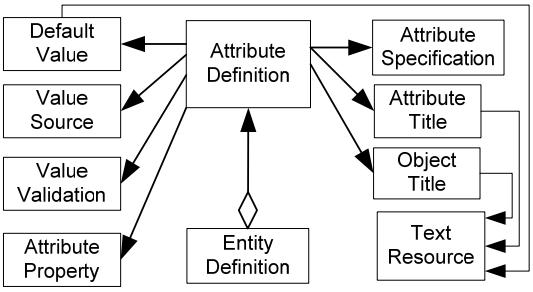

Fig. 7 Semantic representation

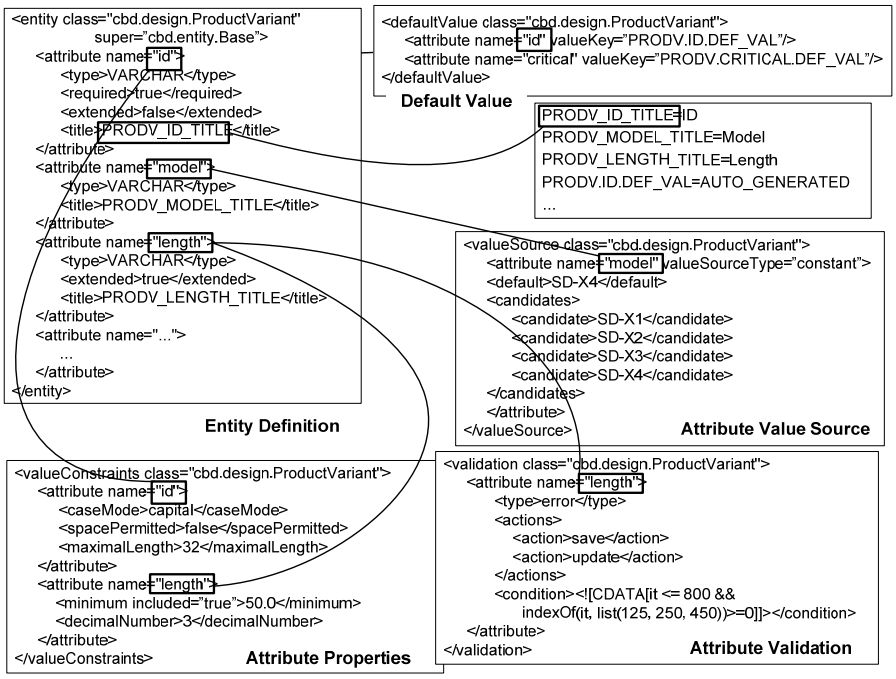

Fig. 8 Semantic product definition

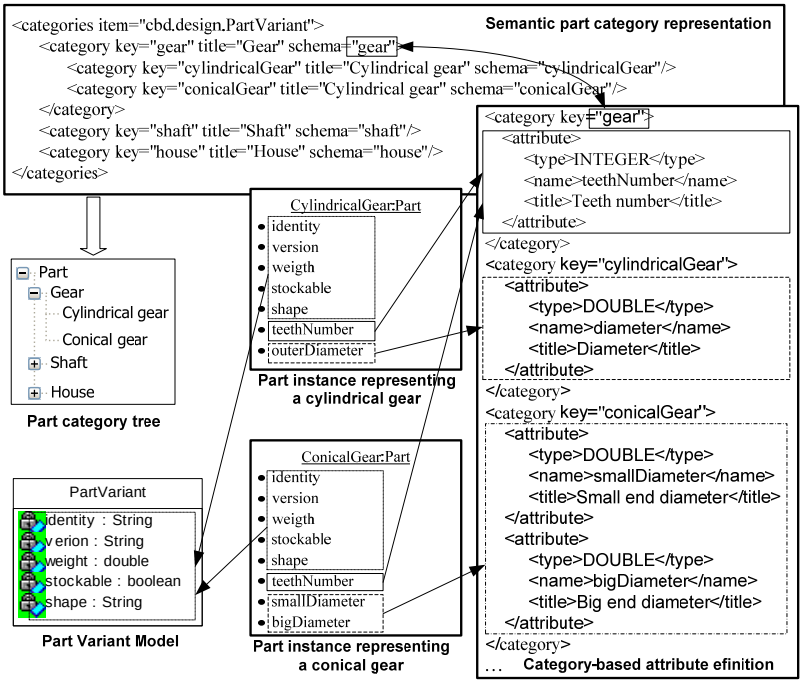

Fig. 9 Semantic category representation

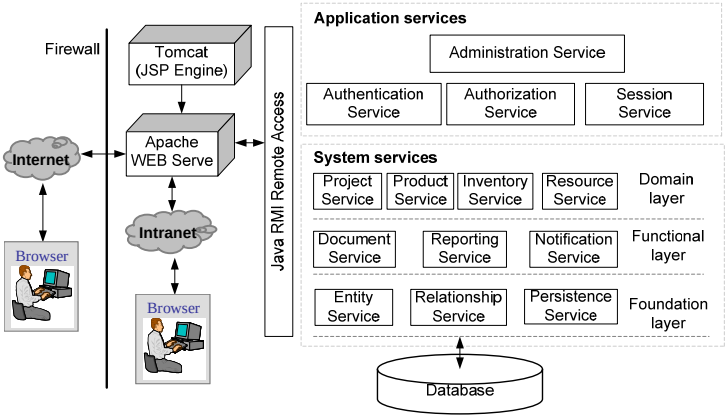

Fig. 10 Architecture of the prototype system

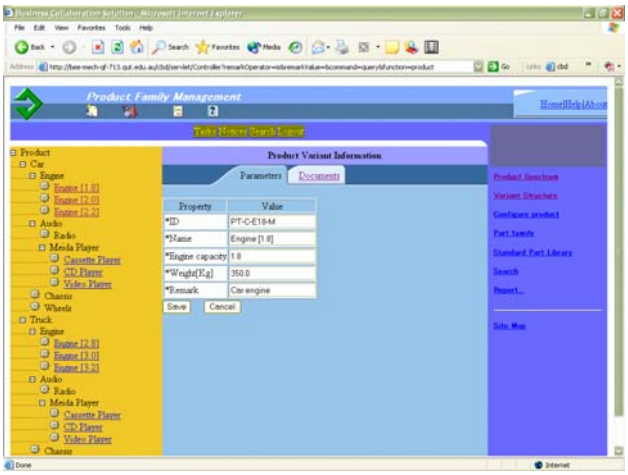

Fig. 11 The interface of a family structure view