Product Configuration Strategy Based On Product Family Similarity

Heejung Lee

***Abstract*—**To offer a large variety of products while maintaining low costs, high speed, and high quality in a mass customization product development environment, platform based product development has much benefit and usefulness in many industry fields. This paper proposes a product configuration strategy by similarity measure, incorporating the knowledge engineering principles such as product information model, ontology engineering, and formal concept analysis.

***Keywords*—**Platform, product family, ontology, formal concept analysis.

I. Introduction

IN a today's highly competitive market, mass customization is of great importance for companies to achieve greater success as in Fortune Magazine declared that "*mass customization* will do for manufacturers the 21st century [1]." The challenge facing the realization of mass customization is to offer a large variety of products while maintaining low costs, high speed, and high quality. To cope with it, many companies consider platform-based product development, sharing components, modules, assemblies, or parts in a product family or even similar product family groups. In general a product family is a set of related products deriving from a platform to satisfy the mass customization, and product family can be defined as: "a set of common components, modules, or parts from which a stream of derivative products can be effectively developed and launched [2]."

The benefit and usefulness of platform-based product development has been proved in many industry fields and a considerable amount of platforms have been defined during the decade, and the primary benefit in platform-based product development is providing economical product variety, that is, product and process excellence to achieve the cost advantage. A platform-based product development approach, however, looks like heavily dependent on the subjective experience and skills of company's individual designers and engineers due to the intrinsic characteristics of platform design.

Therefore the purpose of this paper is to provide the designer and engineers with the appropriate information and knowledge, and takes the knowledge engineering methods to find the correspondent modules between semantically related entities of the products. These correspondences can be used for leveraging product family and platform-based product development.

Heejung Lee is with the Department of Applied Systems, Hanyang University, South Korea (e-mail: stdream@hanyang.ac.kr).

II. Theoretical Background

A. Product Platform

For many years companies have exploited opportunities to create product families by developing and coordinating modular components. The term "platform" can be reviewed in three distinct fields: product development, technology strategy, and industrial economics [3]. Product development researchers introduced the term platform to describe the product family that "meet the needs of a core group of customers for easy modification into derivatives through the addition, substitution, or removal of features [4]", technology strategists used the term platforms as valuable points of control in any industry [5], and industrial economists used the term platforms to characterize products, services or organizations solving the transaction mediation problem [6]. Because the concept of platform has been widely used in various fields, this paper adopts the platform concept as product development flavor which has a root in engineering design.

Simpson et al suggested two approaches to product family design [7]: top-down and bottom-up approach. A top-down approach means that the company will develop the core or common product and its derivatives from the platform, while the company will fix up similar products to standardize components and modularize in a bottom-up approach. Both approaches have effort to define the appropriate product platform from which many products should be derived in an efficient and effective way.

On the other hand, optimizing method to define appropriate platform has been widely used for decades during product development and there are also many methods for optimizing product family and product platform design. There are not appear to be a preferred algorithm, however, among classical optimization algorithm – Branch and Bound, Simulated Annealing, Genetic Algorithm, and Pattern Search [7], while artificial intelligence techniques for product family and product platform design have been relatively successfully employed. Rosen developed the product module reasoning systems, which reasons about product architectures and suggests the all feasible module combinations [8]. Other researchers attempted to develop the reasoning techniques or systems such as product family reasoning systems [9], agent-based systems [10], and case-based reasoning systems [11]. Recently, knowledge-based approaches to infer platform design were attempted by combination of formal concept analysis and ontology [12]. ,

B. Ontology

In today's highly knowledge society, myriad of information systems use many different individual schemas to represent the product configuration. An ontology is the one of promising solutions for represent product knowledge in a formal way. The word ontology means a particular theory of the nature of being or existence, and is used with different meanings in different applications [13], [14]. Gruber originally defined an ontology as a "explicit specification of a conceptualization [13]," Borst defined an ontology as a "formal specification of a shared conceptualization [15]," and Studer, Benjamin, and Fensel merged these two definitions stating an ontology as a "formal, explicit specification of a shared conceptualization [16]."

Therefore, an ontology specify the semantics of terminology systems of a domain of interest and the meanings of domain data formally and explicitly, thereby providing a shared understanding of a domain of interest to support communication among human beings and applications. One main advantage of applying ontologies is the ability to support the sharing and reuse of formally represented knowledge by explicitly stating concepts, relations, and axioms in a domain. Ontologies have been widely applied in a variety of domains to represent information or knowledge models, such as product data models, owing to the fact that its formal semantic can be unambiguously interpreted by humans and machines.

In general, an ontology provides a taxonomy describing a domain of interest of any things in a formal language. In addition, ontologies can be practical means to represent and conceptualize the product data models in a computer format of today's digital or internet era.

C. Formal Concept Analysis

Formal Concept Analysis (FCA) has been introduced by Wille [17], and used for analyzing data and modeling semantic structures in many different research areas. In this section, we briefly describe important terminologies for FCA. *Formal Context* is a triple (C, P, R), where C is a finite set of objects, P is a finite set of properties and R is a binary relation between C and P, i.e. $R \sqsubseteq C \times P$. given two sets $C1 \sqsubseteq C$ and $P1 \sqsubseteq P$, we can consider the dual sets C1' and P1' such as the sets defined by the properties applying to all the objects belonging to C1 and the objects having all the properties belonging to P1, respectively, that is:

$$C1' = \{p \in P \mid \forall c \in C1: (c, p) \in R\}$$

$$P1' = \{c \in C \mid \forall p \in P1: (c, p) \in R\}$$

Formal Concept is a pair (C1, P1), such that $C1 \sqsubseteq C$, $P1 \sqsubseteq P$ and the following conditions holds: C1' = P1, P1' = C1. The sets C1 and P1 are referred to as the *extent* and the *intent* of the formal concepts (or briefly concept), respectively. Therefore, a concept is a pair consisting of two parts, the *extent* and the *intent*. In general, a concept (C1, P1) is a subconcept of a concept (C2, P2) if the extent C1 is a subset of the extent C2 or equivalently if the intent P1 is a superset of the intent P2.

III. Product Platform Design

A. Types of Heterogeneity

The first step of product platform design is to reduce heterogeneity among the languages or terminologies for engineering, marketing, and product functions. There have been many different types of heterogeneity: i) syntactic heterogeneity, which occurs when two or more product concepts are not expressed in the same language, ii) terminological heterogeneity, which occurs due to variations in names when referring to the same entities in different products, e.g., car vs. automotive, and iii) semantic heterogeneity, which stands for the differences in product configuration or modeling for the same concept or function. Ontology-based approach is a promising solution to this heterogeneity situation problem by finding correspondences among different products but having semantically related modular components.

B. Product Information Model

In a heterogeneous environment, the collaborative product development activities involve proprietary product information and the major barrier to effective collaboration is the lack of formal and explicit semantics in the product information model (PIM) that would facilitate semantic interoperability. Over the years, a wide range of researches have been conducted and the ontology-based approach is likely to be the most suitable for integrating diverse heterogeneous engineering applications, as the semantic of the product structure data built in formal logic-based ontology languages, such as Description Logic, can be specified in a well-defined and unambiguous manner.

Description logic (DL) [18], restricted subsets of First Order Logic, is one of the knowledge representation languages that can be used to capture the knowledge of an application domain in a structured and formally well-understood way. The name description logic is motivated by the fact that the important notions of the domain are described by concept descriptions, i.e., expressions that are built from atomic concepts (unary predicates) and atomic roles (binary predicates) using the concept and role constructors provided by the particular DL. In general, a concept denotes the set of individuals that belongs to it, and a role denotes a relationship between concepts.). Consider the main layers of the DL family bottom-up, ALC is a basic and simple language, permitting concept descriptions via $C \sqcap D$, $\neg C$, $\forall$ P.C, and $\exists$ P.C where C, D are concepts and P is a property. Augmented by transitive properties, ALC becomes ALC R+ in the following denoted by S. SI is an extension to S with inverse properties, followed by SHI with property hierarchies. It becomes SHIF if extended by functional restrictions, SHIN if extended by cardinality restrictions, and SHIQ if extended by qualified number restrictions. Support for data type predicates (e.g., string, integer) leads to the concrete domain of D, and using nominals O allows to construct concepts from singleton sets.

On the other hand, the OWL (Web Ontology Language) [19] is an ontology language designed for use by applications that need to process the content of information instead of just presenting information to humans. Notations or names of OWL about the same notions are different from DLs, although OWL is mainly based on DLs. In OWL syntax, class is referred to concept, individual to constant and property to role. Also, DL inferences including concept classification, concept satisfiability, and realization are implemented by DL reasoning systems providing classification, consistency checking, and realization. Therefore, this paper refers to them interchangeably because originally the different names indicate the same notion in both OWL and DL.

A PIM can be defined as a DL-based logical language consisting of a set of concepts and their relations. The concept *Part* is defined as *Thing*, and *Product_Part* is defined as the set of something that is *Part* and has no *isDirectComponentOf* relation with *Part*. Likewise, *Primitive_Part* is defined as the set of something that is *Part* and has no *hasDirectComp* relation with *Part* individual. In contrast, *Assembly_Part* is defined as the set of something that is *Part* and has at least two *hasDirectComp* relations with *Part* individual. Except the previous three concepts, the rest just have has super-concept relation because their definitions are not critical in this paper. In Table I, there are some basic relations which account for product structural information. In addition, the properties have some typical axioms such as ‘transitivity’, ‘inverse(-)’, ‘symmetric’, ‘inclusion’, etc. The property *hasComponent* has ‘transitivity’ characteristics and the ‘inverse(-) property of *isComponentOf*, and the property *hasDirectComp* has ‘inclusion’ relationship with is the sub-property of *hasComponent* and the ‘inverse(-)’ property of *isDirectCompOf*. The property *isCompatibleWith* is used when two parts have compatible parts to each other so that they can be assembled. The property *isNotCompatWith* has the opposite meaning. The two properties, *isCompatibleWith* and *isNotCompatWith* have ‘transitivity’ and ‘symmetric’. Table I shows the above mentioned relations.

TABLE I
PRODUCT STRUCTURE RELATIONS

Name	Domain	Range	Type
hasComponent	Part	Part	Object
hasDirectComp	Part	Part	Object
isComponentOf	Part	Part	Object
isDirectCompOf	Part	part	Object
isCompatibleWith	Part	Part	Object
isDirectCompatWith	Part	Part	Object

In addition to the product structure relations, we should also consider the product attribute relations for each specific product. The range of this product attribute relations can be another class (e.g. a range of an attribute relation *hasFeature* is a class *Chipset*), and also be a primitive data type (e.g. a range of an attribute relation hasRAMSize is an Integer). We define the following primitive datatypes: String, Integer, Float, and Boolean.

C. Basic Techniques

An important success factor of a platform development strategy is how effectively and efficiently new products can be developed from the platform. The goal of this section is to provide how to find the relations between different PIMs and suggest the platform design and customization alternatives. For this we present here the basic method for assessing the similarity between different.

When analyzing similarities between PIMs, we examine at the property levels that are directly related to these concepts. The following is at the heart of the approach: Given a set (Class, Property, Relation), we can consider that the classes and properties are referred to as the product and specification in the product development application, respectively. Therefore, a product group can be defined as the concept consisting of a set of products and a set of specifications.

For instance, consider a PIM set called Project X where,

C = {*C1, C2, C3, C4*}
P = {*P1, P2, P3, P4, P5, P6, P7, P8, P9, P10*}

and R is specified by Table II. In this set, four products are provided, each corresponding to some of four specifications. A sample product group is, for instance, the pair ((*C3*, C4), (*P1, P4, P8, P9*)). We can also consider another product group ((*C4),* (*P1, P4, P8, P9, P10*)), as a sub-set of pair ((*C3*, C4), (*P1, P4, P8, P9*)). We can notice here that by adding the specification *P10* to the former product group, the cardinality of its extent decreases, and by adding products to a product group the cardinality of its intent also decreases.

For this example, the following description logic formulae can be derived from the *Project X.*

$C_1 \equiv \exists\ hasFeature.P_1 \sqcap \exists\ hasFeature.P_2 \sqcap \exists\ hasFeature.P_3 \sqcap \exists\ hasFeature.P_5 \sqcap \exists\ hasFeature.P_6 \sqcap \exists\ hasFeature.P_7.$
$C_2 \equiv \exists\ hasFeature.P_1 \sqcap \exists\ hasFeature.P_2 \sqcap \exists\ hasFeature.P_5.$
$C_3 \equiv \exists\ hasFeature.P_1 \sqcap \exists\ hasFeature.P_4 \sqcap \exists\ hasFeature.P_8 \sqcap \exists\ hasFeature.P_9.$
$C_4 \equiv \exists\ hasFeature.P_1 \sqcap \exists\ hasFeature.P_4 \sqcap \exists\ hasFeature.P_8 \sqcap \exists\ hasFeature.P_9 \sqcap \exists\ hasFeature.P_{10}.$

These formulae were implemented in Protégé 3.5 (http://protege.stanford.edu), which is a widely used ontology development tool. Ontology classification reasoning is one of the most commonly performed activities and Pellet is a built-in reasoner in Protégé 3.5. There are many tools implementing algorithms for Formal Concept Analysis and we can obtain the same FCA solution by doing the classification with a Pellet reasoner. Using the running example, Fig. 1 shows the ontology classification reasoning results, which are the same as FCA solutions, that is, $C1 \sqsubseteq C2$ and $C4 \sqsubseteq C3$.

TABLE II
PRODUCT-PROPERTY RELATIONS EXAMPLE

Products	Properties									
	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10
C1	x	x	x		x	x	x			
C2	x	x			x					
C3	x			x				x	x	
C4	x			x				x	x	x

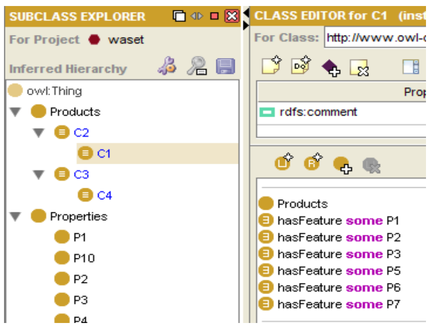

Fig. 1 FCA reasoning using Protégé 3.5

Given this, we apply the product family similarity [20] between two product groups such as:

Sim(PGi, PGj) = w |Ci ∩ Cj| / a + (1-w)*|Pi ∩ Pj| / b.*

where the PGi stands for the two different PIM sets such as PGi = (Ci, Pi), PGj = (Cj, Pj), Ci and Cj are extents of PGi and PGj, Pi and Pj are intents of PGi and PGj, respectively, and *a* is the greatest number of cardinalities of the Ci or Cj, and *b* is greatest number of Pi or Pj. Finally *w* is a weight such that $0 \le w \le 1$, that can be established by the user.

For instance, take two PGs of our running example and assume that *w* = 1/2. Let us start by evaluating the *Sim* randomly, that is, PG1 = ((*C3*, C4), (*P1, P4, P8, P9*)) and PG2 = ((*C4), (P1, P4, P8, P9, P10*)). Since *a* = 2, *b* = 5, *Sim(PG1, PG2) = 0.5*1/2 + 0.5*4/5 = 13/20.*

Now we consider another product family, PG3 = ((*C1, C2*), (*P1, P2, P5, P6*)). Since *a* = 2, *b* = 5, *Sim(PG2, PG3) = 0.5*0/2 + 0.5*1/5 = 1/10.*

This result shows that PG1 and PG2 groups are more directly related each other than PG2 and PG3. Note that *Sim* is always a value between zero and one and, for any pair of product groups, PGi and PGj, and *Sim*(PGi, PGj) = *Sim*(PGj, PGi). Of course, *Sim* increases in the case of product groups that are related, and vice versa, *Sim* decreases in the case of product groups that are not directly related.

D. Determining Product Configuration

From the basic techniques we will have the similarity database for platform configuration strategy. Now questions that need to be answered are: what is the optimal configuration of product group? And what criteria should be used to decide on product configuration design?

The underlying idea in answering this problem is to maximizing the average of total product family similarity. The product group is considered as product *context*, thus the average for our configuration will be: ∑ *Sim(PGi, PGj) / number_of_context.*

IV. CONCLUSION

This paper incorporates the knowledge engineering principles, which provide product configuration guideline based on formal semantics, and proposed method of measuring similarity for assessing product families will be often the starting point when designing the new product (families or groups) or when analyzing the existing family. The well-designed product family will improve the design to achieve better similarity in the family and reduce costs and lead-times. This works provides the guidelines on designing the better product family.

ACKNOWLEDGMENT

This research was supported by the National Research Foundation of Korea funded by the Korea government (NRF-2010-0022827)

REFERENCES

[1] D. J. Gardner, http://masscustomization.wordpress.com/
[2] M. H. Myer and A. P. Lehnerd, *The Power of Product Platforms: Building Value and Cost Leadership*, Fress Press, 1997.
[3] A. Gawer, *Platforms, Markets and Innovation*, Edward Elgar, MA, 2009.
[4] S. C. Wheelwright and K. B. Clark, “Creating project plans to focus product development”, *Harvard Business Review*, 70(2), 67-83, 1992.
[5] T. F. Bresnahan and S. Greenstein, “Technological competition and the structure of the computer industry”, *Journal of Industrial Economics*, 47(1), 1-40, 1999.
[6] J. C. Rochet and J. Tirole, “Platform competition in two-sided markets”, *Journal of the European Economic Association*, 1(4), 990-1029, 2003.
[7] T. W. Simpson, “Product platform design and customization”, Artificial Intelligence for Engineering, Design, Analysis and Manufacturing, 18, 3-20, 2004.
[8] D. W. Rose, “Design of modular product architectures in discrete design spaces subject to life cycle issues”, Advances in Design Automation, No. 96-DETC0DAC-1485, 1996.
[9] Z. Siddique and D. W. Rosen, “ On combinatorial design spaces for the configuration design of product families”, *Artificial Intelligence for Engineering Design, Analysis and Manufacturing* 15(2), 91–108, 2001.
[10] W. Shen, D. H. Norrie, and J. –P. A. Barthès, *Multi-Agent Systems for Concurrent Intelligent Design and Manufacturing*, New York: Taylor & Francis, 2001
[11] D. Sabin and R. Weigel, “Product configuration frameworks—A survey”, *IEEE Intelligent Systems* 13(4), 42–49, 1998.
[12] J. Nanda, H. J. Thevenot, T. W. Simpson, R. B. Stone, M. Bohm, and S. B. Shooter, “Product family design knowledge representation, aggregation, reuse, and analysis”, AI EDAM: Artificial Intelligence for Engineering Design, Analysis, and Manufacturing, 21(2), 173-192, 2007
[13] T. R. Gruber, “A Translation Approach to Portable Ontologies”, *Knowledge Acquisition*, 5(2), 199-220, 1993
[14] N. Guarino and P. Giaretta, Ontologies and Knowledge Bases: Towards a Terminological Clarification. In N. Mars, editor, Towards Very Large Knowledge Bases: Knowledge Building and Knowledge Sharing, 25-32. IOS Press, Amsterdam, 1995
[15] W. Borst. Construction of Engineering Ontologies. PhD thesis, Institute for Telematica and Information Technology, University of Twente, Enschede, The Netherlands, 1997.
[16] R. Studer, R. Benjamins, and D. Fensel. “Knowledge engineering: Principles and methods”, *Data & Knowledge Engineering*, 25(1–2):161–198, 1998

[17] R. Wille, Restructuring lattice theory: an approach based on hierarchies of concepts. In: Rival, I. (ed.) *Ordered Sets*. Dordrecht-Boston, Reidel, 1982

[18] F. Baader and W. Nutt, *Basic description logics in The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge, U.K.: Cambridge Univ. Press, 2003.

[19] D. L. McGuinness and F. V. Harmelen, *OWL Web Ontology Language Overview*, Available from: http://www.w3.org/TR/owl-features/, 2004

[20] J. C. Lee and H. Lee, An Ontological Approach to Measure Similarity between Different Product Information Models, Proceedings of the Asia Pacific Industrial Engineering & Management Systems Conference 2012 V. Kachitvichyanukul, H.T. Luong, and R. Pitakaso Eds.