International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:6, No:12, 2012

The Hybrid Knowledge Model for Product Development Management

Heejung Lee, and Hyo-Won Suh

***Abstract*—**Hybrid knowledge model is suggested as an underlying framework for product development management. It can support such hybrid features as ontologies and rules. Effective collaboration in product development environment depends on sharing and reasoning product information as well as engineering knowledge. Many studies have considered product information and engineering knowledge. However, most previous research has focused either on building the ontology of product information or rule-based systems of engineering knowledge. This paper shows that F-logic based knowledge model can support such desirable features in a hybrid way.

***Keywords*—**Ontology, rule, F-logic, product development.

I. Introduction

THE collaborative product development environment involves product information models, domain-specific engineering knowledge, different domain terminologies, and diverse systems. The product information model includes semantics about properties of product related data such as product structures, work breakdown structure, organizational roles, and so on, while the engineering knowledge is about the rule-base including domain-specific knowledge along with product engineering such as product configuration, process planning, cost targeting, and so forth [1].

The critical issue for effective collaboration in heterogeneous environment is the lack of formal and explicit semantics in the product information model, which could facilitate semantic interoperability. Over the years, a wide range of basic and applied research activities on product data exchange and integration of semantics have been conducted with the aim of semantic interoperability among partners [2]-[5]. The ontology-based approach out of many studies has been considered as the most suitable for integrating diverse engineering semantics, since the product semantics built in ontology language – such as the Web Ontology Language (OWL) [6] – can be specified in a well-defined and unambiguous manner. The other critical issue for knowledge-intensive product engineering is capturing domain-specific knowledge and experience, and reusing them in decision making procedure. To cope with this challenging problem, rule-based systems incorporating the knowledge of the experts in the form of If-Then rules have been suggested [7]-[9].

Normally, ontologies and rules are expressed in different knowledge representation formalism, which makes it difficult to interact between them. This is also likely to cause information to be lost during interactions. Therefore, the integration of ontology-based and rule-based approaches is an essential theme in the knowledge-based product development management.

This paper presents the hybrid way to integrate the ontology-based and rule-based approaches without losing the information, by applying the F-logic approach.

Heejung Lee is with the Department of Industrial and Management Engineering, Daegu University, South Korea (e-mail: 2ssol@daegu.ac.kr).

Hyo-Won Suh is with the Department of Industrial and Systems Engineering, KAIST, South Korea.

II. Preliminary

F-logic approach combines the advantages of logic programming and object-oriented programming paradigm, and extends and subsumes predicate calculus. F-logic is also known to have two flavors: the first-order flavor and the logic programming flavor. In particular, F-logic integrates the paradigms of logic programming and deductive data-bases with the object-oriented programming paradigm. Most of the applications of F-logic are for the intelligent information systems based on the logic programming paradigm, and F-logic has been used to represent ontologies due to its direct support for object-oriented concepts, its frame based syntax, and extensive support for meta-programming. We provide here with an overview of the main features of F-logic briefly [10].

A. Objects

F-logic integrates object-oriented parading and accordingly, objects are basic syntactic elements. Objects can be accessed by *object identity* (OID); for instance, John, man, and son(John) are possible.

B. Methods

Methods are specified using *data-F-atoms* consisting of a host object, a method object, and a result object. Any object is allowed to be appeared in any location of a data-F-atom. Methods may either be single-valued or multi-valued, which is represented by "→"; for instance, "John[son → tom]", which expresses that tom (result) is the son (method) of john (host). In a similar way, "John[son → {Tom, Hans}], which states that Tom and Hans are sons of John, and John may have additional sons.

C. Class Hierarchy

An *Isa-F-atom* of the form **o:c** states that an object **o** is a member of class **c**. A *subclass-F-atom* of the form **sc::c** states that the class **sc** is a subclass of the class **c**; for instance, Tom:man denotes that Tom is an instance of the class man, and man::person defines that the class man is a subclass of the class

person.

D. Signature

Signature-F-atoms specify the schema of a class. In particular, they declare the methods on the classes and give types of the arguments used by those methods, and the ranges of the methods. To distinguish *signature-F-atoms* from *data-F-atoms*, the arrow "*=>" is used instead of "→"; for instance, person[son *=> man]. In F-logic, we use "*=>" to declare a signature of an inheritable method. When a method is non-inheritable, "=>" is used.

E. F-molecules

It is possible to collect several *F-atoms* into a single and convenient, *F-molecule*, such as, Jonh:man[son → Tom:man, daughter → Jane:woman].

F. Rues

One of the best features of F-logic is the usage of rules to derive new information from a given knowledge base. A rule consists of a *head* and a *body*, which are separated by ":-", where the head of the rule is an *F-molecules* and the body is a Boolean combination of *F-molecules* or negated *F-molecules*. In particular, F-logic supports the c*losed world assumption*, which is not explicitly known, is assumed to be false. This is also called *non-monotonic*.

III. Hybrid Knowledge Model

This paper presents a hybrid knowledge representation and reasoning approach based on F-logic, to integrate the ontology-based product information model and rule-based engineering knowledge model. The product information model includes the product semantics while the engineering knowledge model includes engineering-specific knowledge during the product development process. Most product semantics are represented by frame-based approach and some are defined using rules, and engineering-specific knowledge is mostly represented in rules. The adopted F-logic supports this integration.

A. Generic Product Development Process

The generic product development process can consists of four phases. The process begins with a *concept development*, during which the needs of the target market are identified, alternative product concepts are generated, and one or more concepts are selected. The *system-level design* phase includes the production architecture definition and the decomposition of the product into subsystems and components. The *detail design* phase includes the complete specification of the product features such as the geometry, materials, and tolerances of all of the parts in the product. The *testing and refinement* phase include product verification and validation, the construction and evaluation of multiple prototypes of the product. In summary, the product development process begins with the definition of customer requirements and proceeds thorough these four phases. The knowledge model enables the product development process to be efficient and accordingly the company to be competitive. Fig. 1 shows the relations between product development stages, product information model and engineering knowledge base.

We will now describe a project in the conceptual design of car air purifiers. Fig. 2 illustrate rough process for car air purifier design. There are three major tasks such as *Target Spec. Decision*, *Fan Type Selection*, and *Fan Design*. Although each task could have more than one sub-tasks recursively, we do not focus on those sub-tasks but mainly on the three tasks. Almost every time there is a need to design a new air purifier, the target specifications should be decided to satisfy the customer requirements. The two most important factors among the target specifications are noise (dB) and air flow rate (m^3/hour) and they have great impact on selecting fan type. Therefore, as seen in Fig. 3, the second task is to select proper fan type to meet required noise and air flow rate. Once a project manager makes a choice for fan type, other detailed design tasks start. Each detailed design tasks can be performed by geographically distributed partners and then sharing design information among participants is of significance.

The procedure of application scenario has five steps: step1 is for the knowledge building by knowledge engineers, step2 is for the customer requirements setting, step 3 is for the product structure building by adding a new instance based on *hasDirectComp* method, step 4 is for product type selection with reasoning about both product information model and engineering knowledge to realize the instance from the customer requirements, and step 5 is for process planning of the selected feature.

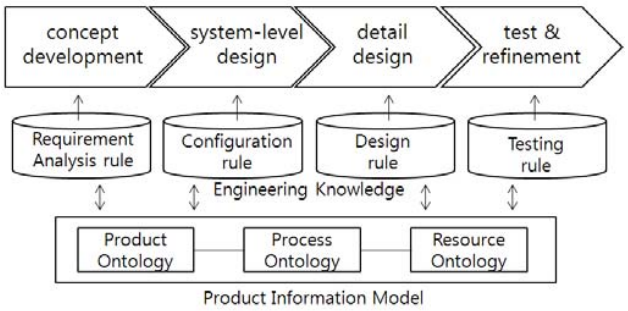

Fig. 1 The interaction between stages and models

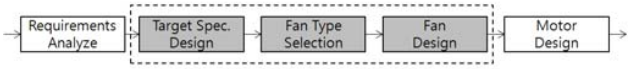

Fig. 2 Generic design process of a car air purifier

B. F-logic as a Hybrid Knowledge Model

F-logic has been viewed as a natural candidate for an ontology language thanks to its support for object-oriented concepts and its frame-based syntax. A typical ontology includes three main components: a taxonomy of classes, definitions of concepts, and definitions of instances. In F-logic, class taxonomies are represented directly using the *subclass-F-atom*; for instance, *compositePart::part*. Concept definitions are represented using *signature-F-atoms*; for instance, *compositePart[hasdirecComp{2:*} *=> part]*, and

finally instance definitions can be specified as facts using *Isa-F-atom* or *data-f-atoms*; for instance, *Part001:part* or *CompositePart002[hasdirectComp → {Part004, Part006, Part007}]*. In addition, derived class can be defined using rules. For instance, if the concepts of part are already defined, we can define a new concept, *smallPart* using the following statements:

?X:smallPart :- ?X:part, ?X[size → Small], where ?X is a variable.

C. Hybrid Knowledge Model for Car Air Purifiers

A distinctive product information model required for the conceptual design of car air purifiers is addressed in Table I, and methods can be also defined using rules; for instance, transitive method *hasComponent* can be defined as follows:

?X[hasComponent → ?Z] :-
?X[hasComponent → ?Y], ?Y[hasComponent → ?Z].

Domain-specific engineering knowledge is built regarding major decisions pertaining to the conceptual design of car air purifiers, which include target specification decision, fan type selection, and fan design. The rule formulae for each task are included in Table II.

TABLE I
PRODUCT INFORMATION MODEL FOR CAR AIR PURIFIERS

Index	Formulae
I1	*productPart::part.*
I2	*primitivePart::part*
I3	*assemblyPart::part[hasDirectComp{2:*} *=> part]*
I4	*carAirPurifier::productPart[hasComponent{1:1}*=>capFanAssy , hasComponent{1:1} *=> capCaseAssy, isInstalled{1:1}*=> position].*
I5	*capFanAssy:assemblyPart[hasDirectComp{1:1}*=> capFan, hasDirectComp{1:1}*=> capMotor].*
I6	*capCaseAssy::assemblyPart.*
I7	*capFan::assemblyPart[hasDirectComp{1:1}*=> blade, hasFanType{1:1} *=> fanType].*
I8	*capMotor::assemblyPart.*
I9	*blade::primitivePart.*
I10	*capAxialFan::capFan[hasFanType{1:1} *=> axial].*
I11	*capCrossflowFan::capFan[hasFanType{1:1} *=> crossflow].*
I12	*capSiroccoFan::capFan[hasFanType{1:1} *=> sirocco].*
I13	*capDualFan::carAirPurifier[hasComponent {2:2}*=> capFan].*
I14	*capSingleFan::carAirPurifier[hascomponent{1:1}*=> capFan]*
I15	*capSingleSirocco::capSingleFan[hasComponent{1:1}*=> capSiroccoFan*
I16	*capSingleSiroccoIn::capSingleSirocco[isInstalled{1:1}*=>inside].*
I17	*capSingleSiroccoOut::capSingleSirocco[isInstalled{1:1} *=>outside].*
I18	*axial::fanType.*
I19	*crossFlow::fanType.*
I20	*sirocco::fanType.*
I21	*inside::position.*
I22	*outside::position.*

TABLE II
DOMAIN SPECIFIC ENGINEERING KNOWLEDGE FOR CAR AIR PURIFIERS

Index	Formulae
R1	*?X [noiseLevel → small] :- ?X:capFan[requiredNose → ?Y], ?Y > 0, ?Y < 30.*
R2	*?X [noiseLevel → medium] :- ?X:capFan[requiredNose → ?Y], ?Y > 30, ?Y < 40.*
R3	*?X [airflowLevel → weak] :- ?X:capFan[requiredAirflow → ?Y], ?Y > 10, ?Y < 30.*
R4	*?X [airflowLevel → medium] :- ?X:capFan[requiredAirflow → ?Y], ?Y > 30, ?Y < 50.*
R5	*?X [airflowLevel → strong] :- ?X:capFan[requiredAirflow → ?Y], ?Y > 50, ?Y < 100.*
R6	*?X[fanType → axial] :- ?X:capFan[noiseLevel → small, airflowLevel → weak] .*
R7	*?X[fanType → crossflow] :- ?X:capFan[noiseLevel → small, airflowLevel → medium] .*
R8	*?X[fanType → sirocco] :- ?X:capFan[noiseLevel → medium, airflowLevel → strong] .*
R9	*?X[installed → outside] :- ?X:carAirPurifier[hasCapFan → {?Y:capFan[noiseLevel → high]}].*
R10	*?X[installed → inside] :- ?X:carAirPurifier[hasCapFan → ?Y:capFan], not Y[noiseLevel → high].*
R11	*?X[hasMaterial → steel] :- ?Y:carAirPurifier[hasComponent → ?X:blade, isInstalled → inside].*
R12	*?X[hasMaterial → plastic] :- ?Y:carAirPurifier[hasComponent → ?X:blade, isInstalled → outside].*

D. Product Structure Building and Type Design

Suppose we have the following requirements for a car air purifier with a single fan 1) 35 decibels as noise requirements, 2) 70 cube meters per hours as air flow requirements.

Following the index *I4* in Table I, we are asked to define *capFanAssy* and *capCaseAssy*. *Cap01:carAirPurifier* to satisfy the definition of *carAirPurifier*, *FanAssy01:capFanAssy* and *CaseAssy01:capCaseAssy* are guided to be added and they are linked with *Cap01* using method *hasDirectComp*. Consequently, for *FanAssy01* to be an instance of *capFanAssy*, it is guided to have one *capFan* and one *capMotor* as its direct component as shown in index I5. Thus, *CapFan01* and *CapMotor01* are guided to be created and they are then linked with *FanAssy01* using method *hasDirectComp*. In the same way, *Blade01* is also linked with *CapFan01* using method *hasDirectcomp*. The overall product structure until now is depicted in Fig. 3.

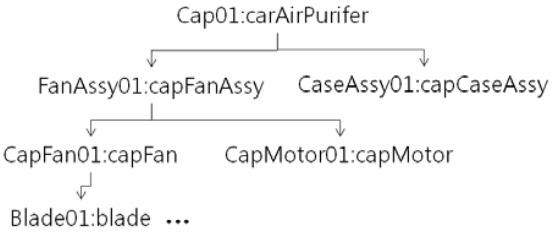

Fig. 3 Product structure of car air purifier

Next, we need to select fan type to satisfy the requirements of noise and air flow. The factual knowledge of fan type, "*noiseLevel → medium*" and "*airFlowLevel → strong*" can be derived from index *R2* and the type of *CapFan01* is determined to be *Sirocco* from index *R8* in Table II. From these results, C*apFan01* is re-conceptified into *capSiroccoFan* because

capFan01 has *Sirocco:FanType* satisfying the definition of *capSiroccoFan* as shown in index *I11*. Additionally, *hasDirectComp::hasComponent* and the transitivity axiom of *hasComponent* conclude that *Cap01* also has components *CapFan01* and *CapMotor01*. From this, C*ap01* is re-conceptified into *capSingleFan*. At this moment, C*ap01* is also re-conceptified into *capSingleSirocco* concept since *Cap01* has one fan and its type is *Sirocco*. Installation place of *Cap01* is decided by its noise, that is, if its noise is high, it should be installed *outside*. We know that the *Cap01* is installed *inside* the car because its noise is *not* proved to be high, *i.e.*, *negation under closed world assumption*. By re-concept reasoning, *Cap01* is re-conceptified into *CapSingleSiroccoIn*, the most specific concept to which *Cap01* belongs. The summary result is shown in Fig. 4 and Table III.

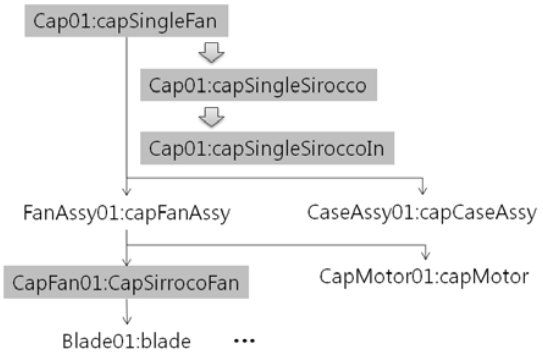

Fig. 4 Product type design of car air purifier

TABLE III
HYBRID REASONING FOR PRODUCT TYPE DESIGN

reasoning	Answer	Justification
Conversion	*?X [noiseLevel → medium], ?X [airflowLevel → strong]*	*R2, R5*
Fan Type Selection	*?X[fanType → sirocco]*	*R8*
Re-Concept	*CapFan01: capSiroccoFan*	*I12*
Re-Concept	*Cap01: capSingleFan*	*I14*
Re-Concept	*Cap01: capSingleSirocco*	*I15*
Installation Place Selection	*?X[installed → inside]*	*R10*
Re-Concept	*Cap01: capSingleSiroccoIn*	*I16*

IV. CONCLUSION

In the collaborative product development environment, the two main types of product knowledge are the product information model and the domain-specific rule model. The product information model includes ontological semantics regarding the properties of the product, such as its structure, material, and features. The domain-specific rule model, on the other hand, includes specific knowledge (expressed as specific rules) along with product engineering tasks such as the configuration, material selection, and process planning of a product.

Normally, these two models have been expressed in different knowledge representation formalisms, which make it difficult to interact between them. This paper presents a hybrid way to integrate the ontology-based and rule-based approach by aping the F-logic approach.

ACKNOWLEDGMENT

This research was supported by the National Research Foundation of Korea funded by the Korea government (No. 2010-0022827)

REFERENCES

[1] J. D. Noh, H. W. Suh, and H. J. Lee, "Hybrid knowledge representation and reasoning with ontology and rules for product engineering," ASME, IDETC/CIE, 2009.
[2] S. R. Gorti, A. Gupta, G. J. Kim, R. D. Sriram, and A. Wong, "An object-oriented representation for product and design processes," Computer-Aided Design, vol. 30, no. 7, pp. 489-501, 1998.
[3] 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, no. 13, pp. 1399-1411, 2005.
[4] L. Patil, D. Dutta, and R. D. Sriram, "Ontology-Based Exchange of Product Information Semantics," IEEE Trans. On Automation Science and Engineering, vol. 2, no. 3, pp. 213-225, 2005.
[5] M. Dong, D. Yang, and L. Su, "Ontology-based service product configuration system modeling and development," Expert Systems with Application, vol. 38, pp. 11770-11786, 2011.
[6] M. K. Smith, C. Welty, and D. L. McGuinness, OWL Web ontology language guide, W3C recommendation, http://www.w3.org/TR/2004/REC-owl-guide-20040210/, 2004.
[7] K. Y. Kim, H. Yang, and D. W. Kim, Mereotopological assembly joint information representation for collaborative product design," Robotics and Computer-Integrated Manufacturing, vol. 24, no. 6, pp. 744-754, 2006.
[8] R. Studer, V. R. Benjamins, and D. Fensel, Knowledge Engineering: Principle and methods," Data&Knowledge Engineering, vol. 25, no. 1-2, pp. 161-197, 1998.
[9] S. Staab and R. Studer (eds.), Handbook on Ontologies, International Handbooks on Information Systems, Springer-Verlag, Berlin, Heidelberg, 2009. pp. 45-70.
[10] K. T. Ulrich and S. D. Eppinger, Product Design and Development, McGraw-Hill, 2000.