Adaptable Component Model for Product Line Engineering

김정아*, 배제민**

Abstract

Adaptable Component Model for product line Engineering

Adaptable component model (ACM) is proposed based on the product line analysis. The ACM helps revealing variable parts in the system and tailoring them in real time based on pre-defined rules as such enhancing and maintaining an enterprise level application system are effectively managed. With the rule components, we can make necessary changes easily to adapt to new business context. A pilot study was conducted at an enterprise-level application. The productivity improvement performance was resulted in 6 times effective in average in terms of LOC and 2 times decreased workload in terms of M-day for one new product addition. The quality improvement was resulted in 5 times less in average in terms of the number of defects and 3 times decreased workload. Accordingly, this paper (1) proposes a component architecture based on ACM (2) defines the rule components to define variable parts of business, and (3) introduces a case study which applies product line based ACM.

Key Words: Product Engineering, Component, Varaibility, Adaptation

* 관동대학교 컴퓨터교육과 교수, 컴퓨터공학 박사, clara@kd.ac.kr
** 관동대학교 컴퓨터교육과 교수, 컴퓨터공학 박사, gemini@kd.ac.kr

1. Introduction

Business is continuously changing with market. An information system must be used to produce products needed for the business during the life cycles of business and products timely, properly, high quality, and cost effectively as needed. Indeed, it is important to determine which parts of the information system are for reuse, elimination, and change without spending much time and effort, to accomplish these, and to produce demanded products for the expectation.

The product line1 engineering has been developed and used to analyze and define a family of products to be provided for customers of a segment of market [1,2] Because it is a process to analyze a set of products, to find reusable components that are configured the family of the products, e.g., the product domain, the engineering should be used for developing a new product in the product domain. This approach eliminates redundancy of efforts and resources across the products of a product line and results in the high quality of products.

This concept is applied to developing and maintaining a software intensive system or information system with the component based software engineering paradigm that is based on the object technology [3,4]. It provides a powerful foundation in designing and implementing with objects, in extending component based domain modeling, and in enabling the abstraction of software system at the component level that is one level higher object system.

It should be expected that the architecture of the components identified through analyzing a product family ought to be mapped at the architecture of relevant components of the information system that is used to produce products in the product family [4,5]. In another word, the invariable (common) parts and variable (non-common) parts of a product line (e.g., family) should be mapped to the structure of the elements of the information system used in relevance of producing the products.

Then the life cycle of the product family and the system are exactly and correctly supported, based on their own rights.

Secondly, the reuse improvement of a software component is helped if the common parts and the variable parts of a component are identified during the development for reuse according to the component domain model [5,6,7]. This analysis depends on the interpretation method of the component model that is imposed by the product line analysis view point. This reuse of a component can be enhanced for real time if a mechanical or automated interpretation method of the component model is introduced. In this paper, adaptable component model (ACM) is proposed based on the integration of the product line analysis and the component based software engineering. The ACM provides mechanical tailoring capability for the variable parts of a component. Then this helps revealing variable parts of components in the system, viewing the derived variability, and expressing solutions as business rules, and separating them from the class codes. In addition, following are described in this paper:

(1) A component architecture which separates rule components carrying components and elements of variability, allowing reuse in its original definition,

(2) Rule syntax necessary to define rules, and

(3) A case study of applying the rule-based component development.

Chapter 2 explains the motivation of this research and chapter 3 proposes ACM model for rule-based component architecture and methods of rule definition. Chapter 4 illustrates a case study applied to a development of a software system and defines the effect while chapter 5 gives a conclusion.

2. Problems and Approach

The quality of an information system is critical to the system user, organization' management that system's services. In the National-Defense applications, there are several products which have a lot of commonalities and little difference. The various departments or subunits of national defense developed their own systems supporting these to manage products that were same or similar. The whole or partial redundancy in the products caused problems because of the uncontrolled redundancy, in development and implementation because different interface and unique policies were implemented.

This redundancy problem brings serious risk that may cause business errors due to misunderstanding of the business rules, resulting in inefficient management activities and developing a new product. According to our experiences, we can easily find the following problems in many organizations.

- Absence of an integrated architecture: An integrated architecture is a set of the concepts that define the overall structure and behaviors to implement the whole set of business rules. It should be a high level description of business rules, components, the interfaces, protocols to external systems and between the components, and control mechanisms followed by the system and its all elements. Because no high level description of the architecture was presented, integrated view point was not easily achieved. This leaded up to not being able to know and understand the service scopes, rules of various applications, the operations, and information implemented on the system' platform.

- Incomplete standardization of components: No standardization was recognized in the system implementation. Frequently similar or duplicated elements were identified over multiple applications and versions. In other words, there was inefficiency such that each business applications develop similar or same service modules repeatedly.

- Inefficiency in operations and maintenance: Redundancy of rules and functions and the inadequacy of implementation strategy for business application resulted in many similar but different operational problems. Even if a large resource was committed to change requests to a business system, the project could not meet with the planned schedule and the quality objectives.

Re-engineering of business rules and information system is related to desired changes to the features that various business applications what National Defense department uses. The product line engineering was applied to the product line: the set of the business rules and determined the component architecture and the invariable and variable parts of the components respectively as solution to the target problems.

Because of the recognized complexities in the products, there are three important constraints identified showing why the product line engineering was needed[1,2,5,7,8]:

- A function is implemented at multiple places in the system.
- A change needed to fix defect must be applied at the multiple places where the same functionality is implemented.
- A feature implemented at multiple places behaves differently depending on its context of a particular product.

The software product line approach is unique in coping solutions for these problems. Indeed, a software product line defines the requirements, the architecture, and a set of reusable components sharable among multiple products (e.g., releases, versions, or product family) which implement a considerable part of the products'functionalities. Our approach should reduce the time and costs and should increase the quality of the production of products in the family because of the usage of the reusable components (core assets) which have already been tested and secured. The requirement documents, conceptual diagrams, architectures, codes (reusable components), test procedures, and maintenance procedures are also the common development artifacts for sharing and managed as asset. Revealing the variability of the products in a product line view point, subject of this study, is a very important success factor. The variability analysis of a product line is to show how differences among the products contained in the product line are allowed and facilitated. Many approaches of how to identify and manage the variability have been proposed but they do not work effectively particularly when new products are engaged in or when existing products are evolving (scalability).

The variability of components plays a central role in the entire software product line development process, e.g., the development life cycle from the requirements elicitation to the architecture, the component design, the coding, and the tests. Also the variability is connected with the corresponding feature, as varying, under specific conditions that should be determined appropriately for effective management of the product line.

3. Extending the component model for variability management

Architecture that dynamically adapts at runtime to new user requirements is called Adaptive object-model (AOM)[9,10]. In this paper, the AOM and business rule principle are extended for handling the product-line engineering and the variability of components is formulated. This new capability is called as Adaptive Component Model (ACM).

3.1 ACM (Adaptable Component Model)

This paper proposes the Adaptable Component Model (ACM) to deal with the variability at component level. Fig.1 shows a basic idea of the ACM which provides the business concept used in building a business domain component [12]. ACM represents components, attributes, and strategies in the form of meta model and business rule, e.g., product rule model, described at the instance level rather than the class specification level, as shown in Fig 1. Users change the metamodel and the rules to reflect desired changes in the domain.

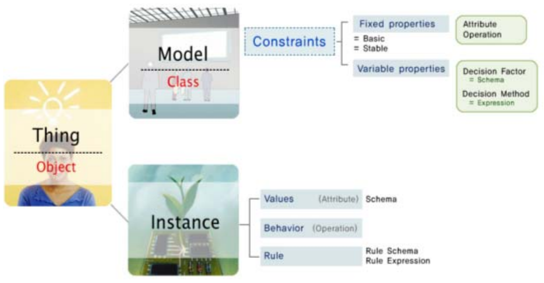

Fig 1 Basic Idea of ACM

The model defines constraints (the properties of domain and operations) to describe the thing, e.g., the objects. The constraints are divided into essential parts and variable parts. The essential parts mean the attributes to describe the thing and the basic services which manipulate the attributes. The variable parts mean those parts with continuously changing properties.

It is hardly possible to predict what variability is made available or what variability will happen. The variable parts are defined in detailed such elements specified to decide which variable ones should be selected and their interpretation methods for their customization. Actually, variable parts are recognized not as general components but as rule components in this study.

Here is the definition of business rule. A business rule is a statement that defines or constrains some aspect of the business [11,12]. It is intended to assert business structure for business domain or to control or influence the behavior of the business process. From the business domain perspective, it pertains to any of the constraints that apply to the behavior of people in the enterprise, from restrictions on sales to procedures for filling out a purchase order. It is a variability of business strategy. From the information system perspective, it pertains to the facts that are recorded as data and constraints on changes to the values of those facts. That is, the concern is what data may, or may not, be recorded in the information system. These are reflected to variable parts of business component.

3.2 Architecture with ACM

With ACM, we can develop an architecture such as shown in Figure 2. With business modeling, the rule components to manage the variability are identified such as the user interface layer, application component layer, business component layer, and persistent layer. Beneath these, there is BRAIN (Business Rule-engine for Adaptable and Intelligent eNterprise) layer for the rule engine to interpret the variability of component expressed in rule format and to execute the rule when invoked.

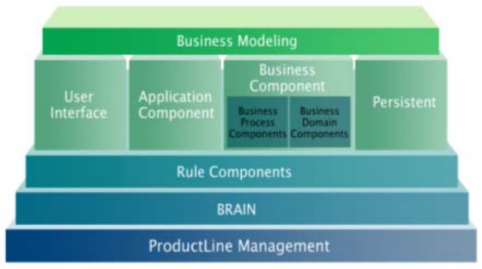

Fig 2 Architecture based on ACM

The BRAIN layer can be, for example, Jlog, Blaze Advisor which are commercial rule engine and OpenRule and Doors which are open source rule engine. These are java-based rule engine and follow JSR94 standard [14]. These rule engines might have similar architecture as illustrated in Fig 3. The business components interact with the client API.

A rule modeler is an actor and uses the rule editor to define rule models and to generate business rules. The rule model defines, like a class, rule schema and rule operation. Rule is an instantiation of the rule model. Such rules are stored in the rule repository with the rule model, while some rule engines do not store rules with rule model. We prefer to store both for the purpose of helping understanding and instantiating the rule model and of achieving higher flexibility in meeting business strategy and generating rules.

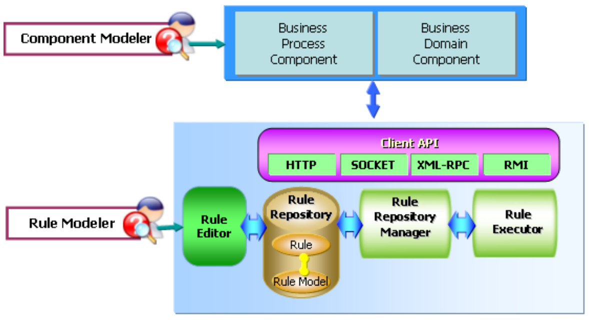

Fig 3 Rule-based component architecture

After integrating current business model and components, the rule models are constructed which consists of business domain component and business process component. The business domain component is an entity to define domain features and information needed for the process handling. The business process component defines flow and conditions of the business processing. These two components interact in such that the domain component identifies and determines if the business process component need to be invoked according to its environment.

The rule component is to define the variable parts of business rules and policies, separated from business domain component and business process component. With the rule component, extendibility and flexibility of component are increased.

3.3 The construction of the rule components

Variable parts in a software component are related to future changes in the business model and its interpretation. The scope of the variability of a component includes addition of attributes, change or addition of operation, change in operation implementation. If generalizing such variability, they can be regarded as changes in the domain model, changes in interpreting ways to the elements constituting the model, and changes to the workflow. This paper proposes ACM to manage such variability in the model, the model interpretation, and the associated workflow. The rules for the variability are classified into the next two groups: 1) the business process rules to define variability in the business processing, and 2) the business domain rules in their structure and interpretation, which is shown below:

Business rule = Business Process Rule + Business Domain Rule,

where the business domain rule is a rule to interpret the value of the object. The business process rule is a sequence of rules and conditions to process a business task.

The implementation of domain rules uses XML. For example, insurance names, interest, insurance type, entry limit, age at the entry, so on, are the elements of XML in the case of insurance domain application. Let us consider two domain rule examples. One is the ‘ge at entry’check and the other is ‘entry limit’check.

The implementation of process rule uses the rule syntax shown in Table 1.

Table 1 Syntax for process rule

```
ReturnType Rule_Heading {
        IF ( condition )
        THEN
        ELSE
        ADDITIONALINFO
}
```

As shown in Figure 4, the PrimitiveRule is a bare rule and the RuleExecutionSets is a set of one or more process rules to form a sequential business workflow.

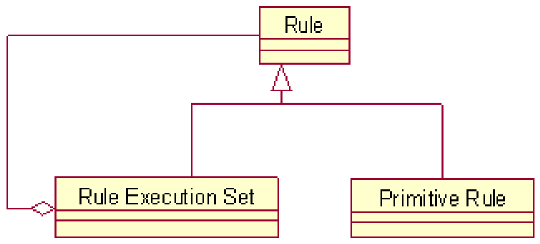

Fig 4 The construction of rule execution set

A business process can be implemented through the execution of the RuleExecutionSet. Table 2 describes an example of the RuleExecutionSet for checking the entry conditions. A rule is classified into the two types: the generic rule and specific rules. An insurance product can reuse the generic rules.

Table 2 Example of rule execution set

```
    Boolean MainInsuranceEntryCheckRule() {
      IF(FIRE("InsuranceProduct_MainInsurance_MaximumEntryAmountCheckRule")
           and FIRE("InsuranceProduct_MainInsurance_MinimumEntryAmountCheckRule")
           and FIRE("InsuranceProduct_MainInsurance_MinimumEntryAgeCheckRule")
           and FIRE("APlanInsurance_MainInsurance_MaximumEntryAgeCheckRule") )
      THEN Collection(Boolean true)
      ELSE Collection(Boolean false)
      ADDITIONALINFO ELSE "Not Pass EntryCheckRule"}
```

4. Product Line Development

4.1 Development Process

Accordingly, KobrA[15], a method of component-based product line engineering, was adopted for the product line development in this study. This method unifies the product line paradigm and the component based paradigm and supports both of "reuse in large" and "reuse in small." A KobrA splits the development cycle into two parts, the first part is to deal with development of a framework to allow a reusable set of software artifacts which are commonly embedded within all the products for the company. The second part is to carry out application development through instantiating, adapting, and extending the framework to meet the needs of specific requirements from users or customers. The framework engineering encompassed the activities and artifacts related to the development and maintenance of the framework. The application engineering encompassed the activities and artifacts related to the development and maintenance of an application. During the framework engineering, variability and commonalities were captured and the characterization of the product line was accomplished. The application engineering includes activities dealing with the instantiation of the framework.

4.2 Component-Based Product Line Engineering

A software component is an independent unit with the well-defined interface to perform its functions and assembled with other components to produce an entire application [16]. Our components are the logical representation and logical building blocks to be reused in a software system development, because these components in the KobrA framework do not include any attributes specific to the component implementation technologies (e.g. J2ED/EJB, CORBA, .NET/COM+).

The construction steps follow the context analysis, component modeling, and realization process, where the most important task is to establish ready-for-reuse components in the domain. The application building process through the component-based product line engineering is now divided into the next three steps:

Step_1: establish the business components, e.g. ready-for-reuse components, for the product line development, Step_2: design a targeted business application, assuming the ready-for-reuse components,

Step_3: produce a targeted business application, by assembling the ready-for-reuse components

4.3 Application Engineering

We reused the business components and customized the rule components according to the business context. With these business components and rule components, different product could be generated. A new system is produced by integrating, e.g., reusing components selected from the different presentation technologies and the application components.

The rule modeler uses the rule editor to customize rule components and verifies the correctness of the rules regarding the changes in rule components by the rule simulator. The modified business rule is immediately ensured not affecting to the business components.

5. Benefits of Engineering of Insurance Product Line

Because the variable parts are localized in the rule components, new requirements or requirement changes are reflected to the business rules by modifying the associated rule components.

We applied this ACM idea to the enterprise application in one company for one year. During this period, new 153 products were registered and 178 insurance products were updated using the rule components. In table 3, we summarized the effectiveness of the rule components in terms of LOC and M-day.

Table 3. Effectiveness of Rule component

	Without rule component		Applying rule component	
	LOC	M-day	LOC	M-day
Add a new product	1,500~ 2,500	10~22	250~ 660	5~15
Customize a Existing product	700~ 1,200	7 ~ 15	100 ~ 150	1.5 ~ 5
Defect/KLOC	80~100		15~22	
Defect Fix time	6 M-day		2.5	
Dynamic Change	X		O	

Before applying rule concept, each product was developed separately and repeatedly. Of course cut&paste reuse by individual programmer without any planed reuse strategy. Since it was neither an organizational approach nor a planned reuse, the productivity of individual programmer and the quality of the generated programs were not improved.

After applying the rule components, in other words, the planned reuse by product line was enhanced very much and the productivity and quality were improved. The productivity improvement performance was resulted in 6 times effective in average in terms of LOC and 2 times decreased workload in terms of M-day for one new product addition. The quality improvement was resulted in 5 times less in average in terms of the number of defects and 3 times decreased workload.

The testing effort can be decreased for each new product addition because the business components and the rule components were reused. Also the testing was localized to the modifiable rule components. Verification for each channel was not necessary any more.

The layered architecture of the product line established a framework better, compared with the previous version, for developing the core assets (invariable and variable parts,) because a significant part of the asset is independent from implementation. The asset includes code, definition, and all other key documentation and helps the use, reuse, and enhancement of the asset.

Another benefit was organizational restructure. After applying the product line, the organization was restructured and the three departments were defined: 1) the component engineering department responsible for maintaining the core assets, 2) the application engineering department responsible for customizing the core assets by modifying the rule components, and 3) the each department responsible for maintaining the presentation layer. The business experts are assigned to the application engineering department, so, even new channel comes out, there is no necessity to build new team. Therefore, the IT cost regarding skilled human resources was saved.

6. Conclusion

The most important factor in the construction of business system is to establish the components along with product line engineering, e.g., the business components for common and stable parts and the rule components for variable parts. In the business point of view, with a cope of requirements, environments and business processes that are continuously and rapidly changing, business organizations must need to adapt their business system as quickly as possible. The product-line based ACM is expected to work for this approach. Business system must be maintained and sustained to be adapted to necessary modification of the existing process to create new product by reengineering of the processes and existing products.

The case study has demonstrated these properties and the time-to-market such that the company’ new products have been provided just in time with effective IT costs investments.

References

1. Jan Bosch, 『Design and use of software architecture adopting and evolving a Product-line Approach』, Addison-Wesley, 2000
2. Paul Clements and Linda Northrop, 『Software Product Lines: Practices and Patterns, Addison-Wesley』, 2001
3. Jean-Christophe TRIGAUX and Patrick HEYMANS, 『Software Product Lines: State of the art,』, 2003
4. J. Bosch, “Software Product Lines: Organizational Alternatives” in Proceedings of the 23rd International Conference on Software Engineering, Toronto, Canada, pp. 91–100, 2001.
5. M. Aksit. “Composition and Separation of Concerns”in the Object-Oriented Model. ACM Computing Surveys, 28(4), December 1996.
6. J. Bayer. “Separation of Concerns in Product Lines Engineering” In K. Mehner, M. Mezini, E. Pulverm□uller, and A. Speck, editors, Aspektorientierung - Workshop der GI-Fachgruppe 2.1.9 Objektorientierte Software-Entwicklung. Technischer Bericht der Universitt Paderborn tr-ri-01-223, May 2001.
7. P. America, H. Obbink, R. van Ommering and F. van der Linden, “oPAM: A Component-Oriented Platform Architecting Method Family for Product Family Engineering” in Proceedings of the First Software Product Line Conference, Denver, USA, pp. 167–180, 2000.
8. J. Bosch, “roduct-Line Architectures in Industry: A Case Study” in Proceedings of the 21st International Conference on Software Engineering, November 1998.
9. Joseph W. Yoder, Federico Balaguer, Ralph Johnson. “rchitecture and Design of Adaptive Object-Models” in Proceedings of Object-Oriented Programming Systems, Languages, and Applications (OOPSLA ’1); ACM SIGPLAN Notices, ACM Press, December 2001.
10. Joseph W. Yoder, Ralph Johnson. “mplementing Business Rules with Adaptive Object-Models” Business Rules Approach. Prentice Hall. 2002.
11. Brian Foote, Joseph W. Yoder. “etadata and Active Object Models” Proceedings of Plop98. Technical Report #wucs-98-125, Dept. of Computer Science, Washington University Department of Computer Science, October 1998.
12. Business Rule Group, Defining Business Rules ~ What Are They Really?, Technical paper, www.businessrulegroup.org
13. Burkhard Peuschel,Wilhelm Schäfer, “oncepts and Implementation of a Rule-based Process Engine” in Proc. of the 14th International Conference on Software Engineering, 1992
14. JavaTM Rule Engine API Specification 1.0 http://www.jcp.org/aboutJava/communityprocess/review/jsr094/
15. Colin Atkinson, Joachim Bayer, Christian Bunse, Erik Kamsties, Oliver Laitenberger, Roland

Laqua, Dirk Muthig, Barbara Paech, Jürgen Wüst, and Jörg Zettel. Component-based Product Line Engineering with UML. Component Software Series. Addison-Wesley, 2001

16. J. Bayer, O. Flege, P. Knauber, R. Laqua, D. Muthig, K. Schmid, T. Widen, and J.-M. DeBaud. PuLSE: A Methodology to Develop Software Product Lines. In Proceedings of the Fifth ACM SIGSOFT Symposium on Software Reusability (SSR'9), pages 122–31, Los Angeles, CA, USA, May 1999. ACM.