2014 Second International Symposium on Computing and Networking

A Product Domain Model based Software Product Line Engineering for Web Application

Takashi Nerome
Department of Computer Science
The University of Electro-Communications
Tokyo, Japan
Email: nerome@jp.ibm.com

Masayuki Numao
Department of Computer Science
The University of Electro-Communications
Tokyo, Japan
Email: numao@cs.uec.ac.jp

***Abstract*—Software product line engineering (SPLE) is a methodology for developing a diversity of software products and software intensive systems at lower costs, in shorter time, and with higher quality. SPLE is widely known to develop industrial products such as embedded system. However,for applying SPLE to develop Web applications which have to provide transactional logics for products such as financial products, the biggest issue is the lack of design method and application architecture to execute concurrently for a whole product line on the same runtime. Therefore development cost still tends to increase with frequent changes of the business requirements. We proposed two noble approaches to apply SPLE each for design method and application architecture.Firstly,for design method, we redefined the design scope of product as Product Domain Model, and designed a UML based meta-model which adds the notations of variability. Secondly, for application architecture, we adopted dependency injection technology to execute transaction logics for product line. We also defined a unit of logics for product line as Instance Product. To generate a plenty of resources regarding Instance Product, we created a generator which inputs Product Domain Model. In this paper, we introduce our approaches and evaluation by the pilot development of banking products. The results show that with our approaches, the issue of Web application development can be solved effectively, as well as the additional issue of maintainability.**

***Keywords*-Software Product Line Engineering; Product Domain Model; Instance Product; Dependency Injection; UML**

I. Introduction

Software product line engineering (SPLE) provides a systematic means of managing variability in a suite of products[1][2][3]. SPLE has been also recognized as one of the foremost powerful techniques for developing reusable and maintainable software within system families[4][5][6][7].

As for the application area of SPLE, industrial area is the most famous for the development of industrial products such as embedded system. However, for the development of Web applications which have to process the transactions with business logic such as banking and insurance applications, SPLE has not been used widely yet. Thus, we identified the issue to apply SPLE in this financial area.

For engineering activity, SPLE can be broken down into two core activities: domain engineering and application engineering[5]. The key task of domain enginering is to model the domain itself in order to lay the foundation for deriving individual products, which is the activity of application engineering. However, for Web application development, these dividing of activities are not the same. only domain engineering exists to develop variety of product without using the application engineering.

To solve the issue, we propose a new engineering approach to apply SPLE for Web application. We defined a model of domain engineering as Product Domain Model which express individual products, using a UML based meta-model with notations of variability.

For the runtime, we defined an application architecture to adopt dependency injection technology. The target of dependency injection is logics of a product. We defined the target as Instance Product. To generate a plenty of resources regarding Instance Products, we created a generator which inputs Product Domain Model.

The remaining of this paper is organized as follows. Section II discusses several related works on development regarding complexity of product's diversity. In section III, the details of the proposed approach is presented. Following this, the result of the pilot development based on the approach is provided in section IV. Finally, the conclusion and the future work of this paper are stated in section V.

II. Related Works

A. Domain-driven Design

Domain-driven design (DDD) focuses on what matters in enterprise applications: the core business domain[8][9]. Using object-oriented principles, developers can develop a domain model that all team members-including business experts and technical specialists-can understand. Even better, this model is directly related to the underlying implementation. DDD is effective to extract scope for target application but not include specific methods of diversity design.

978-1-4799-4152-0/14 $31.00 © 2014 IEEE
DOI 10.1109/CANDAR.2014.105

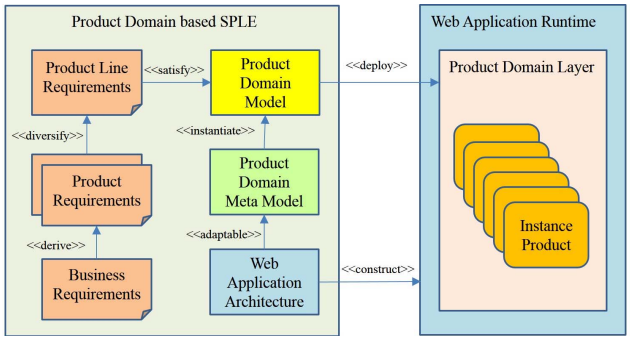

Figure 1. A concept of product domain based SPLE for Web application

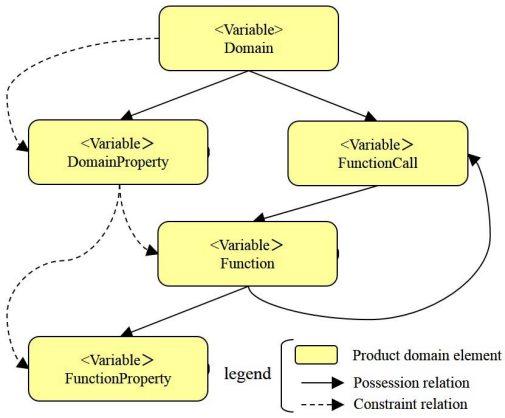

Figure 2. An overview of Product Domain Meta-Model

B. Dependency Injection

Dependency injection (DI) is a technology for reducing inter-component dependency in the application architecture with supporting aplication runtime[14]. DI enables loosely-coupled components, which are thereby highly reusable. The idea of dependency injection is to move the code for instantiating sub-components from the program of a component to a component of framework such as the Spring framework[12], which makes instances of sub-components specified by a separate configuration file and automatically stores them in the component. As for applying DI to process a lot of components, How to define relations among components are much important[13].

III. Proposed Approach

As for our approach, Figure 1 shows a concept of product domain based SPLE for Web application. Product domain based SPLE is domain engineering activity of general SPLE which can handle product line. Each logic for product without conditional branch of function call is executable as Instance Product in product domain layer of Web application runtime.

Product requirements derived from business requirement are diversified as product line requirements. Product Domain Model is designed to satisy product line requirements. Product Domain Model is also instantiated from product domain architecture adaptable with Web application architecture which is refered to contruct Web application.

Product Domain Model is essential in our approach to handle variability definition of product line. Instance Products are generated to execute based on Product Domain Model.

A. Product Domain Meta-Model

Figure 2 shows an overview of product domain meta-model that we defined based on DDD and feature model of SPLE. Domain means variety of product and a product is defined as variant of Domain. Domain also contains DomainProperty and FunctionCall.Domain and DomainProperty are able to contain variants more than 0. Domain's behavior can be defined as relations of FunctionCall, and FunctionCall is also able to contain variants more than 0. A characteristic of the product domain meta-model is easily enable to add product newly by definition updates for variants of DomainProperty, Function and FunctionCall. For adding product, reusable of Function is enable without influence to the existing products.

We implemented the product domain meta-model by UML and UML extension with stereotypes although we omit this explanation because of the page limitation.

B. Instance Product

For the diversity of product is expressed by a number of Variant of Domain and DomainProperty. Variability are defined as Variants of domain which is shown in Figure 2, and defines a number of Variants related to DomainProperty. We defined the combination which has no more choice to Variant, as Instance Product. For applying the DI technology, Conditonal branches are unnecessary to be included in the application business logic. Instance Product has ID of Instance Product, and able to run in product domain layer. Instance Product is transformed into XML for DI cofiguration and will be read by DI container.

C. Application Architecture

We adapted DI technology to application architecture which is shown in Figure 3. it makes possible to execute Instance Product at product domain layer within business logic layer. First, the business facade is called from presentation layer through service layer. Business facade request the ID extractor of Instance Product to get ID of Instance Product. Business Facade also request to instantiate Instance Product if it is not instantiated yet. the ID extractor for Instance

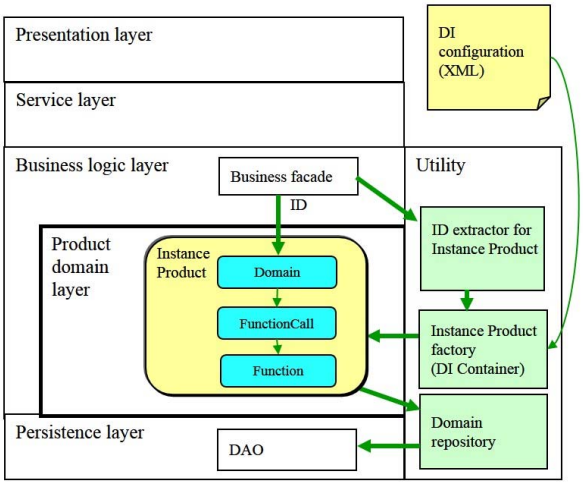

Figure 3. Application architecture

Product invoke Instance Product factory to instantiate the requested Instnace Product.Instance Product factory is the same as DI container which read information of Instance Product from DI configuration defined in XML format.

As the designed logic, the business facade invokes the Domain of Instance Product which is requested by ID.

Function's logic can invoke domain repository when it needs to access DB repository. Domain repository invoke DAO (Data Access Object) located at Persistence Layer to access the data.

In this way, To execute Instance Product in the application architecture is very familiar with Product Domain Model in the design method.

D. Design Method using Product Domain Model

Figure 4 shows a design method using Product Domain Model. Product Domain Model starts with considering product line from the whole application area. First, we define a structure for Domain and Function using class diagram using DDD based design. Next, we define Variant and dependency constraints using object diagram as Variability definition. Variants of Function are defined as a number of objects, and indicates dependency with alternative. Next, we define behavior for FunctionCall using activity diagram. To define FunctionCall using method invocation, it is possible to design control logic to determine if call method or not. The inputs of the determination refer the dependency contraint definition of object diagram. Then we refine the structure by behavior confirmation using sequence diagram according to activity diagram. This will also refine the whole UML model. Next we define implementation information such as class path and package name. From the product line view point, we can use the tool to generate Instance Product List to confirm and refine it. With the model-driven capability,

Table I
THE NUMBER OF EACH ELEMENT FOR 8 BANKING PRODUCT DOMAIN MODEL

Element type	Element name	Variant information
Domain	Banking product	8 products
DomainProperty	7 properties	totally 43 variants
FunctionCall	16 function call	totally 8 variants
Function	136 functions	totally 166 variants
FunctionProperty	1 property	totally 17 variants

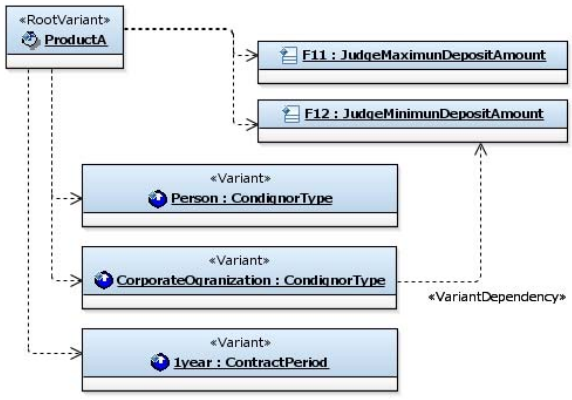

Figure 5. Object diagram for a part of variant and dependency

the tool can generate XML for DI, and the relational Java classes for Domain and Function.

IV. PILOT DEVELOPMENT AND EVALUATION

In this section, we explain the pilot development for an example of banking products and the evaluation of our approach. For the tool platform of integrated development environemnt, we used IBM Rational Software Architect[15]

A. Pilot Development

We analyzed each of requirements of banking products using general documentation, and created the derived requirement documents as product line requirements. Table I shows summarized information of banking product line. As stated in Table I, there was 8 products. The number of each element is shown in in Table I.

As for the design of th pilot development,We show a part of banking Product Domain Model using Figure 5 and Figure 6.

Then we also show the result of using the tool in Figure 8 and Figure 7.

B. Evaluation by Additional Products

To evaluate modifiability, We also designed 3 additional products sequentially in the Product Domain Model. To analyze values about Product Domain Model. we created the tool which input Product Domain Model.

Table II shows each of element number for 3 additional products. For the logics regarding them, it address there

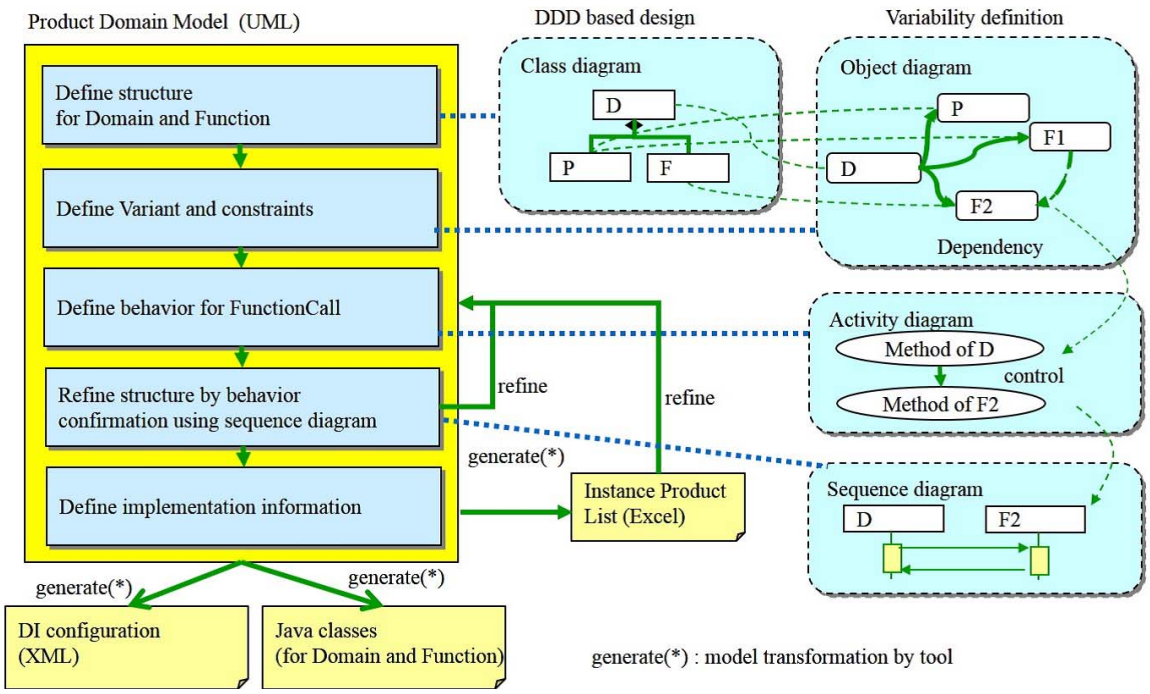

Figure 4. Design method for Product Domain Model

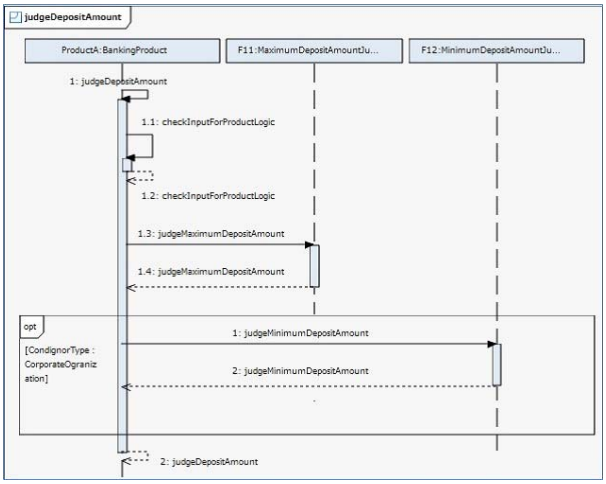

Figure 6. Sequence diagram as a part of behavior for banking Product Domain Model

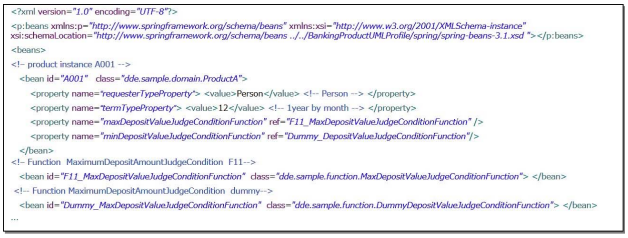

Figure 7. A part of the generated DI configuration of Spring framework

appear calculation process which is independent variants of domain properties.

Table II also shows the reusable rate of function element for 3 additional products and shows additional function number which needs to develop. Here the reusable rate is calcurated as multiplied by 100 the value divided by the whole number of model elements which is needed to develop.

Table II
THE ANALYSIS RESULT BY ADDITION OF THREE BANKING PRODUCTS

Additional products	A	B	C
Variants of DomainProperty	7	23	22
Functions	41	52	30
Variants of Function	41	22	30
FunctionProperties	0	1	1
Variants of FunctionProperties	0	17	17
Instance Products	2	36	17
Average reusable rate of all elements(%)	25.6	54.5	92.4
Reusable rate of Function(%)	9.8	78.9	83.3
Additional Functions(%)	37	11	5

According the the values in Table II , Product A requires

Element	Element value	Variant value	Product Instance			
			A001	A002	A003	A004
DomainProperty	CondignorType	Person	●		●	●
DomainProperty	CondignorType	Corporate Ogranization		●		
DomainProperty	ContractPeriod	1year	●			
DomainProperty	ContractPeriod	2year			●	
DomainProperty	ContractPeriod	5year				●
Function	MaximunDepositAmountJudgeCondition	F11	●		●	●
Function	MaximumDepositAmountJudgeCondition	dummy				
Function	MinimumDepositAmountJudgeCondition	F12		●		
Function	MinimumDepositAmountJudgeCondition	dummy			●	●

Figure 8. A part of the generated instance product list

newly calculation logics, and the reusable rate is lowest. On the other hand, the number of added domain attributes are 0, therefore existing products structure is not needed to be affected. Moreover, in added product B and C, the reuse rate are high, and the number of additional functions which are needed in new programming development will be lead to the minimum. This time, by the evaluation of pilot development which is adding insurance product derivation, it appears that we can correspond with minimum new functions. Moreover, when there are some affects to existing functions, we can afford by changing the XML for DI by switching the existing to the new function as all financial products. For example, when changing the logic of calculation at the tax calculation job, by using Product Domain Model function change, it is possible to make reflect to applications execution against multiproduct easier.

From the efforts point of view, The model transformation tool can generate the accruate DI configuration file which has 5 thousand lines. this can save much time and efforts to maintain DI configuration.

V. Conclusion

In this paper, we introduced our approaches and evaluation by the pilot development of banking products. The results show that with our approaches, it is much effective to apply for Web application which has logics of diversity of products. it is also possible to make cost and eforts lower for the maintenance of product. We are also working to create the import tool from the requirement definition. In the future work, we will focus on impact analsis with traceability in the lifecycle of development starting from requirements.

Acknowledgment

The authors would like to thank Tamotsu Sogoh of IBM Global Business Service for the support of requirement definition for Product Domain Model. The author is also grateful to Keiichi Matsuyama of Exa corporation for the help regarding the pilot development of this research.

References

[1] D. Weiss and C. Lai. Software Product-Line Eng.: A Family-Based Software Development Process. Addison-Wesley, 1999.

[2] Pohl, Klaus, Gunter Bockle, and Frank Van Der Linden. , Software product line engineering: foundations, principles, and techniques. Springer, 2005.

[3] Klaus Schmid, Rick Rabiser, and Paul Grnbacher. 2011. ”A comparison of decision modeling approaches in product lines. In Proceedings of the 5th Workshop on Variability Modeling of Software-Intensive Systems (VaMoS ’11)”. ACM, New York, NY, USA, 119-126.

[4] Ziadi, Tewfik, and Jean-Marc Jezequel. ”Software product line engineering with the UML: Deriving products.” Software Product Lines. Springer Berlin Heidelberg, 2006. 557-588.

[5] Clements,P., and Northrop,L.(2001). Software Product Lines: Practices and Patterns, 3rd edn. Reading, MA: Addison-Wesley Professional.

[6] K. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson. : Feature-Oriented Domain Analysis (FODA) Feasibility Study, Technical Report, Software Engineering Institute, Carnegie Mellon University, CMU/SEI-90-TR-222 (Nov. 1990).

[7] Buchmann, Thomas, and Bernhard Westfechtel. ”Mapping feature models onto domain models: ensuring consistency of configured domain models.” Software & Systems Modeling (2012): 1-33.

[8] Domain-Driven Design : http://domaindrivendesign.org/

[9] Evans, E. *Domain-Driven Design, Tackling Complexity in the Heart of Software*,Addison-Wesley (2003)

[10] Business Rule Mangement System : http://www.hartmannsoftware.com/pub/Enterprise-Rule-Applications/brms

[11] ILog and Business Rule Management System http://www-06.ibm.com/software/jp/websphere/ilog/business-rule-management/

[12] spring framework : http://projects.spring.io/spring-framework

[13] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. ”Design patterns: Abstraction and reuse of object-oriented design. In European Conference on Object-Oriented Programming Proceedings” (ECOOP ’ 93), volume 707 of Lecture Notes in Computer Science. Springer-Verlag, July 1993.

[14] Martin Fowler, ”Inversion of Control Containers and the Dependency Injection pattern” http://www.kakutani.com/trans/fowler/injection.html, 2004

[15] Leroux, Daniel, Martin Nally, and Kenneth Hussey. ”Rational Software Architect: A tool for domain-specific modeling.” IBM systems journal 45.3 (2006): 555-568.