Indian Journal of Science and Technology, Vol 9(46), DOI: 10.17485/ijst/2016/v9i46/107184, December 2016
ISSN (Print) : 0974-6846
ISSN (Online) : 0974-5645

Decomposition of Reusable Architecture Asset

Hanyong Choi[1], Sungho Sim[2*]

[1]Major of Computer Engineering, Shinhan University, Korea; hychoi@shinhan.ac.kr
[2]Department of Liberal Education, Semyung University, Korea; shshim@semyung.ac.kr

Abstract

Objectives: To disassemble design information into parts assets to solve the problem of software production method and to reduce duplicate development costs due to domain changes. **Methods/Statistical Analysis:** To solve this, the study tries to create parts asset for reusing domain design information in the design process that is independent to the development environment. **Findings:** Also in the design process, predesigned design information to apply reuse of parts assets and the reuse of this architecture level that can accommodate this is difficult. Therefore to reuse design information in the design process, there needs to be abstracted architecture information platform independent from development environment. Also it needs to be based on well-designed architecture that can support design of application domain. Therefore the study used a DMI structure that can formally express the design information of architecture level platform and application domain. Therefore the study defined a method of creating component assets of design information to store architecture design designed by designer in DMI as parts assets. **Improvements/Applications:** Thereby it was made possible that design information of designer is configured as PLE asset in the extractive method and also it was made possible to synthesize and reuse.

Keywords: Asset, Architecture, Component, DMI, PLE, Reuse

1. Introduction

Reuse and automated production of software has been very difficult to satisfy various demands[1]. Also effective development method to solve this has been continuously sought after and there are many researches for resolving these problems about software system reuse and automated production[2,3]. Also rather than the demand of reuse of code in the process of system development, design information reuse of the process of abstract design process has increased. Code reuse increased the problem of low rate of reuse in readability and dependency on development environment.

To solve this, the study tries to create parts asset for reusing domain design information in the design process that is independent to the development environment. However in the design process, there is need for parts needed in the architecture and application domain of domain that is the target of design. In the end, well defined and structured design information that can be reused by new designers is not provided and also there is no support for environment where provided design information can be synthesized to expand design information[4,5]. Also in the design process, synthesizing predesigned design information to apply reuse of parts assets and the reuse of this architecture level that can accommodate this is difficult. Therefore to reuse design information in the design process, there needs to be abstracted architecture information platform independent from development environment. Also it needs to be based on well-designed architecture that can support design of application domain[6].

Therefore the study used a DMI structure that can formally express the design information of architecture level platform and application domain[6-8]. DMI stores design information by disassembling into parts assets in the design process of high abstraction level and to enable synthesis of this, meta model was expressed using XMI. Therefore design information is stored as parts assets expressed as formalized meta model using XMI and it was made so that this can be synthesized and the design information reused to design. Also DMI structure enables design that is reusable and platform based architecture

*Author for correspondence

level. In production method, based on the problem of formalizing previous design information and reusing and platform, extractive method PLE structure that can solve problems of software design was chosen. Therefore the study, based on DMI structure, architecture level design is done in the design process and Also design information is a reusable part asset independent from platform that is reusable, considering reasonability and assembly, architecture components can be refined and stored, and part asset is flexible according to the design purpose of designer and independent parts synthesis is possible.

2. Background

2.1 PLE

To increase productivity of software, component parts are produced and assembled. However there are difficulties in maintenance for adapting to various platform changes and increasing productivity or in cases where there is need for reproduction. The most effective method for this has been continuously required and the method to solve this problem about Salter reuse and automated production method is Product Line Engineering methodology[9-11]. PLE is largely composed of two stages, domain engineering that creates product assets by analyzing similarities and differences and application engineering that creates certain products that the customer wants using PLA. Domain engineering is analyzing the similarities and differences of products included in a certain domain to make a product-line asset, and it is composed of architecture design and component design. In architecture design, broad and wide design decision is made and in component design, narrow design decision is made. In component design, considering reasonability and assembly, architecture component is defined and for the product-line, the planned product must be able to be assembled by reusable asset component.

PLE methods can be divided into reactive method, proactive method, and extractive method. Reactive method is a method of gradually expanding previous software product line and a new product as needed or if new requirements occur for the current product. With low cost, software product line expansion is possible and while it is possible to flexibly respond to new requirements, it has the characteristics of being difficult to apply when there are almost no similarities between new requirements and if many parts are variable. Is adequate for when it is difficult to predict requirements for new products within the software product line. Also it is an adequate method for a domain that needs to frequently and flexibly respond to similar and new requirements between new requirements nearly suggested between users. Proactive method is a method of analyzing, designing, and implementing a complete software product line to support all ranges predicted to be needed for future products and when S/W Product Line Asset is developed, it can minimize time and effort needed in developing a product adequate for the market but it dusty characteristics of requiring long analysis and design time for software product line as well as high costs and time expenditure. Because this method requires much time and high costs, it is inadequate for cases where it exceeds the allowable range, and it is adequate for large corporations that have plenty of capital. It is adequate for a stable domain with a well defined predicted requirement about the necessary product. Extractive method is a method of analyzing and extracting similarities and differences in a system product previously developed into a software product line and it is a method of extracting information from data such as previous source code, design, and domain analysis. Because compared to proactive method it requires less cost and time, effective software reuse is possible but it dusty characteristics of not being adequate for cases when it is difficult to find similarities and differences between previously developed system. It is useful when it is possible to reuse the previous product or system and it is a useful method there are many similarities and consistency in differences between systems.

2.2 DMI

As expressed in Figure 1, DMI system structure is designed into three layers, Design Layer, Mapping Layer, and Infra Layer. Design layer is a layer that expresses design information based on UML and expresses stored information and infra layer is a layer that stores architecture and component information made into component with XML. Also, mapping layer which is the middle layer got exchanges information between design layers and stored design information is layers that converts XML information and expresses UML, as does the role of searching and managing components, and creating code. In the mapping layer, modules that models design structure expressed with UML and services for storing meta-models that compose the design structure of infra layer provided. Also it is composed of a code creation module that creates XML code by mapping stored meta-models on code

```
<XMI xmi.version="2.1">
#HEADER#// Header
#CONTENT#// Content
#EXTENSIONS#// Extensions
</XMI>

// Header
<XMI.header>
<XMI.documentation>
<XMI.exporter>DMI</XMI.exporter>
<XMI.documentation>
<XMI.metamodel xmi.name="UML" xmi.version="2.1"/>
</XMI.header>

// Content
<XMI.content>
<Model xmi.id="#ID#">
<xmi.name>#ARCHITECTURE_NAME#</xmi.name>
<ownedElement>
[repeat CLASS]
#CLASS#
[/repeat CLASS]
[repeat ASSOCIATION]
#ASSOCIATION#
[/repeat ASSOCIATION]
</ownedElement>
</Model>
</XMI.content>
<Class xmi.id="#ID#">
<xmi.name>#CLASS_NAME#</xmi.name>
<feature>
[repeat OPERATION]
#OPERATION#
[/repeat OPERATION]
</feature>
</Class>

<Operation xmi.id="#ID#">
<xmi.name>#OPER_NAME#</xmi.name>
<visibility xmi.value="#VISIBILITY#"/>
</Operation>

<Association xmi.idref="#ID#">
<xmi.name>#ASSOC_NAME#</xmi.name>
<connection>
:
</connection>
</Association>

                                        // Extensions
```

Figure 1. Meta Model Structure.

creation template. Designers authenticate divided into component designer and architecture designer and component designer registers service components and limited to registered service components they received authority to edit and delete. Registered service components support facet search that has facet lists classified by previous component search words and classification by purpose and evaluation. To create an architecture design structure where reuse independent from certain tools and platforms as possible, certification from architecture designer is received and when registering design information, the structure is modeled with UML and configuration information is stored in the product line asset. Then, a UML based editor that can visually model design structures like these is supported. To reuse architecture structure in the future, configuration information stored in the database is mapped on a code creation template composed of XMI spec to create and store as an XML code.

3. Expression of Design Information

The study used meta-model to formalize and express design information structure independent from the main environment as shown in Figure 1. Four meta-data, XMI (XML Metadata Interchange) was used to express meta-models for design information.

For design information, design was done using meta models, architecture and expansion meta-models. The first model is a meta-model to express the architecture for managing as formalized asset and the second model is an expansion meta-model to express to component for servicing according to the characteristics of the application domain of the designer. Architecture meta-model represents provision as standard design information of formalized domain area and it is expanded combining the service component of the user and this domain architecture. To express the design information of the formalized architecture, XMI was used as seen in the next figure to define meta-model. (Figure 1)

Also as seen in Figure 2, for meta-models for expressing reused and expanded architecture, meta-models that expressed pre-defined architectures were used to include architecture information and relevant information to enable expression of a new location domain, domain architecture to define meta-model expressed with XMI. The <Header> part records the type and XMI information of meta-model. (Figure 2)

<XMI xmi.version="2.1">
#HEADER#
#CONTENT#
#EXTENSIONS#
</XMI>
<XMI.documentation>
<XMI.exporter>DMI</XMI.exporter>
<XMI.documentation>
<XMI.metamodel xmi.name="UML" xmi.
version="2.1"/>
</XMI.header>
<XMI.content>
<Model xmi.id="#ID#">
<xmi.name>#DOMAIN_ARCHITECTURE_
NAME#</xmi.name>
<ownedElement>
[repeat ARCHITECTURE]
ARCHITECTURE
[/repeat ARCHITECTURE]
[repeat ASSOCIATION]
#ASSOCIATION#
[/repeat ASSOCIATION]
</ownedElement>
</Model>
</XMI.content>
< ARCHITECTURE xmi.id="#ID#">
<xmi.name># ARCHITECTURE _NAME#</xmi.
name>
</ ARCHITECTURE >
<Association xmi.idref="#ID#">
<xmi.name>#ASSOC_NAME#</xmi.name>
<connection>
<feature>
#class#
</feature>
</connection>
</Association>
<Class xmi.id="#ID#">
<xmi.name>#CLASS_NAME#</xmi.name>
</Class>

Figure 2. Domain Architecture Structure.

The <Content> part records the XMI information that has the actual architecture meta-model information and finally, the <Extension> part is composed of parts to express expanded information of meta-model. Therefore according to the grammar to compose meta-model of the architecture assembled with component, architecture again, is composed

of components and sets of relationships and each component is again, composed of class and sets of relationships. Like this, design architecture about service domain, previous asset is utilized to compose new domain architecture.

4. Decomposition of Design Information

4.1 Decomposition Structure of MetaModel

To define architecture structure, XMI meta-model was used and to decompose the designed architecture, operator and the calculation method for decomposition calculation was defined in follow Figure. XMI decomposition calculation has the calculation function of removing elements composing metadata in metadata. (Figure 3)

4.2 Decomposition of MetaModel

Meta-data was composed so that architecture could be stored as asset by decomposing metadata combined into reusable asset in metadata model of the synthesized architecture. Here, the process of decomposing metadata, as seen in Figure 4, a method of separating one data element by one within metadata is used. Here, in the decomposition process, the architecture to be decomposed is selected and architecture ID and element ID is obtained. (Figure 4)

Then with decomposition calculation, architecture ID and element ID are decomposed and in content domain, metadata received from decomposition calculation is decomposed. Architecture name and metadata of ID separated in this process is decomposed and the metadata about the decomposed element of architecture is decomposed to give new architecture name and ID. (Figure 5)

XMI.decomposite
<!ELEMENT XMI.decomposite ANY> <!ATTLIST XMI.decomposite %XMI.element.att; %XMI.link.att; xmi.position CDATA "–1">
XMI.replace
<!ELEMENT XMI.replace ANY> <!ATTLIST XMI.replace %XMI.element.att; %XMI.link.att; xmi.position CDATA "–1">

Figure 3. Decomposite Architecture.

1: ***select*** architecture for decomposition;
2: ***if*** (interface) {
3: get architectures-ID, elements-ID;
4: decomposite metadata of architecture-ID, elements-ID;
5: decomposite metadata of architecture-name in content area;
6: decomposite metadata of architecture-element in content area;
7: create new metadata of architecture-name, architecture-ID;
8: }
9: else
10: can't decomposite this architecture;

Figure 4. Decomposition Procedure.

XMI.decomposite
<XMI.content> <XMI.differencehref="original"> [repeat architecture] <XMI.decompositehref="original/ architecture-ID> <XMI.decomposite xmi.id="#ID#" xmi.name="#ARCHITECTURE_NAME#"/> <XMI.decompositehref="original/association-ID> <XMI.decompositexmi.idref="#ID#" xmi.name="#ASSOC_NAME#"/> <XMI.decompositehref="original/Class-ID> <XMI.decomposite xmi.id="#ID#" xmi.name=#CLASS_NAME#"/> [/repeat architecture] <XMI.replacehref="original/Model/> <Model xmi.id="#New_ID#" xmi.name="#New_Architecture_Name#"/> </XMI.difference> </XMI.content>
<ARCHITECTURE xmi.id="#ID#"> <xmi.name>#ARCHITECTURE_NAME#</xmi.name> </ ARCHITECTURE > <Association xmi.idref="#ID#"> <xmi.name>#ASSOC_NAME#</xmi.name> <connection> <feature> #class# </feature> </connection> </Association> <Class xmi.id="#ID#"> <xmi.name>#CLASS_NAME#</xmi.name> </Class>

Figure 5. XMI Decomposition.

In order to decompose architecture according to XMI. decomposite calculation, architecture is selected using <XMI.decompositehref=”architecture“> and to obtain pattern to be decomposed and relevant information in XMI.decomposite calculation, each architecture is classified by xmi.id value and it is divided into elements and architecture. Using meta-model, the elements of architecture that are trying to decompose by repeating XMI. decomposite calculation are decomposed. Like Figure 5, architecture-ID and association-ID information is obtained and meta-model information is decomposed on XMI meta-model of architecture, then from the metadata with expanded architecture, the architecture can be decomposed. Here, a new architecture name is given by <XMI.replace> population and name of element is changed. The architecture being decomposed within meta-model has registered the name and ID of architecture given from content domain so this decomposed, and the decomposed architecture and relevant information is decomposed together.

5. Conclusions

The study conducted an evaluation on six systems to check the changes in complexities and completion in design information for when designing at the class level and designing based on domain architecture by using decomposed and stored parts assets. In the four design information cohesion, domain design information asset of DMI was used design based on design information architecture. Here, DMI based on parts asset is applied and when the target system is designed, because of design information uses UML in the class level and it can be seen that there is higher cohesion and designing reusing parts assets.

Also when reusing parts asset to design based on domain architecture, design complexity comparatively reduced compared to when designing at the class level. Therefore DMI asset design method is stronger organizational measurement of cohesion in respect to the complexity of the system. Each system can be seen that the complexity of the results obtained by applying the higher of organization domain architecture having a strong cohesion, because reusable asset increased abstracted designing level in systems. Therefore to set the optimal number of classes and relation according to design purpose when designing system, based on previous designed experience, using optimal obstructed domain architecture as parts assets, cohesion of design information can be strengthened. Also as size of systems increase and complexities increase, by applying domain architecture, cohesion can be strengthened and by strengthening organization, organization fit for the purpose of system design can be obtained. This can be seen as an effect that can be obtained from acquiring reusability in assembly when reusing PLE parts assets based on domain engineering and strengthening of organization by parts assets refined from architecture component.

6. References

1. Dao TM, Kang KC. Mapping Features to Reusable Components: A Problem Frames-Based Approach. Proc. 14th Int’l Conf. Software Product Lines: Going Beyond. 2010; p. 377-92.
2. Larsen G. Model-Driven Development: Assets and Reuse. IBM Systems Journal. 2006; 45(3):541-53.
3. Bosch J. Super-Imposition: A Component Adaptation Technique. Information and Software Technology. 1999; 41(5):257-73.
4. Kastner AS, Lengauer CC. Language-Independent and Automated Software Composition. The Feature House Experience. Software Engineering IEEE Transaction. 2011; 39(1):63-79.
5. Esmail A, Ali G. A New Architecture for Enterprise Resource Planning Systems Based on a Combination of Event-based Software Architecture and Service-oriented Architecture. Indian Journal of Science and Technology. 2015 Jan; 8(2):108-19.
6. Batory D, Lofaso B. Smaragdakis. JTS: Tools for Implementing Domain-Specific Languages. Proceedings of the 5th International Conference on Software Reuse. 1998; p. 143–53.
7. Choi H, Sim S. A Study on Software Development method based on DMI. International Conference on System in Medicine and Biology. 2015; 2(1):359-60.
8. Apel S, Kolesnikov S, Liebig J, Kastner C, Kuhlemann M, Leich T. Access Control in Feature-Oriented Programming. Science of Computer Programming. 2012; 77(3):174-87.
9. Pohl K, Bockle G, Linden FJVD. Springer: Software Product Line Engineering: Foundations, Principles and Techniques. 2005.
10. Thompson V, Heimdahl MPE. Structuring Product Family Requirements for N-Dimensional and Hierarchical Product Lines. Requirements Engineering. 2003; 8(1):42-54.
11. Altintas NI, Dogru CS, Oguztuzun AHH. Modeling Product Line Software Assets Using Domain-Specific Kits. Software Engineering, IEEE. 2011; 38(6):1376-402.