Journal of Computer and Communications, **2014, 2, 85-90**
Published Online January 2014 (http://www.scirp.org/journal/jcc)
http://dx.doi.org/10.4236/jcc.2014.22015

Scientific Research

Conversion of Object Oriented System into Software Product Line with Delta Modeling Abstract Behavioral Specification

Ricky Timothy Gultom, Maya Retno Ayu Setyautami, Iis Solichah

Faculty of Computer Science, Universitas Indonesia, Depok, Indonesia.
Email: ricky.timothy@ui.ac.id; maya.retno21@ui.ac.id; iis@cs.ui.ac.id

Received November 2013

ABSTRACT

This article contains a system conversion from object oriented design into Software Product Line (SPL) using delta modeling of Abstract Behavioral Specification (ABS). ABS is a modeling language which targets system with high level of variety and supports SPL development with delta modeling. The case study of this thesis is a digital library system called Library Automation and Digital Archive (LONTAR). Originally, LONTAR only uses SOAP-based web service. With ABS, LONTAR will be converted into SPL and implement another web service called REST. The motivation of this conversion of LONTAR from object oriented into SPL is because it is easier to develop system with ABS than using regular object oriented. Product definition in ABS is relatively easier than creating a new subclass and do customization to make it works well.

KEYWORDS

System Conversion; Abstract Behavioral Specification; Delta Modeling; Software Product line; Object Oriented

1. Introduction

In software development scenario with high variety level, software modeling language and powerful tools to help the implementation process are among of most important things. Design-oriented languages such as Unified Modeling Language (UML) cannot fulfill this need because of their lack in executability. Implementation-oriented languages like Java Modeling Language (JML) are lacking on verifiability. This kind of lackness is tried to be solved by formal abstract language such as B-Method [1]. However, the usability is low because it is not easy to use.

The problem is one of the background of development of Abstract Behavioral Specification (ABS). One of the target of ABS is highly reusable software. Software reuse has been a common activity to simplify system design implementation. One of traditional approaches to software reuse is the Object Oriented (OO) System. It introduces the concept of abstraction, encapsulation, and polymorphism. The software reuse lies in its inheritance concept, which provides object type generalization. While the OO system is popular, it does not support effective reuse mechanism [2]. It cannot fully accommodate large system with frequent changes or has many variations of products. One of the reasons is the reuse mechanism has to maintain the object type consistency from parent classes to their subclasses. There is another mechanism to software reuse known as Software Product Line (SPL), which is more suitable for large system to implement than OO. It introduces the concept of core products and variant products [2].

This paper proposed a strategy to convert an OO system into SPL. We use the ABS delta modeling to form the conversion mechanism. Delta modeling is used by ABS as the main code reuse principal instead of inheritance. Therefore, when we convert OO to SPL, the inheritance in OO will be replaced with delta modeling ABS.

To verify the conversion mechanism proposed, a case study has been conducted. The criterion of appropriate case study is relatively large object oriented system and has need for variations. This paper chooses digital library system in Universitas Indonesia, called Library Automation and Digital Archive (LONTAR), as a case study because it fulfills the criterion. The issue that we try to discuss in this paper is the web service provided by LONTAR. Since its inception, LONTAR only provides SOAP-based web service. Whereas, REST web service has been commonly used nowadays. Therefore, there is also a need for LONTAR to provide REST. With ABS, we try to facilitate

LONTAR to fulfill those needs. Moreover, we also try to show advantage of using ABS rather than object oriented.

This paper is divided into six sections: Introduction, Literature Study, Conversion Steps, Experiment and Observation, Concluding Remarks, and finally Acknowledgments. Conversion Steps section contains steps to convert an object oriented system into SPL using ABS. Content of each steps is elaborated more clearly in Experiment and Observation section.

2. Literature Study

2.1. Abstract Behavioral Specification

There are many known modeling languages such as Unified Modeling Language (UML), Java Modeling Language (JML), and Behavior Tree. Each of those languages has certain flaw or weakness. For example, UML cannot be executed directly and cannot be verified. On the other hand, implementation level languages such as JML are lacking on verifiability aspect. Mathematical approaches try to solve the problem but they are too complicated, thus making the usability low.

ABS tries to cover those weaknesses mentioned above. ABS is a software modeling language which located between realistic and abstract language. The definition of realistic language is language in implementation level or language closely similar to how people think to solve problems. Meanwhile, abstract language is formal language and very mathematical. ABS adds executability for design-oriented modeling language such as UML and verifiability for realistic language and tries to improve usability for abstract language.

According to [3], ABS targets concurrent, distributed, object oriented software. Moreover, software is developed with several components and with a high variable level. To achieve the last one mentioned, ABS implements delta modeling as the main paradigm to develop various systems. Further information about ABS can be accessed in http://www.hats-project.eu/.

2.2. Software Product Lines

Software Product Lines (SPL) is a set of software which have several similar features [4]. Each element of the set however has unique features to comply with market demand or special mission. Products in SPL pertain to the same business goal or application domain, they share the same architectures; they are also built from the same components and services [5]. Core assets of SPL are the architectures, components, and services aforementioned. They are called core assets because they are the main resources to create new products in SPL.

Eventually, new products will not be created from scratch again because they can be built simply by reusing and reprocessing the core assets which already are there. Therefore, the process of creating new products will be faster and more efficient. In addition to that, the new products will be easier to maintain because they have similar components.

One way to develop SPL is with Delta Oriented Programming (DOP) which is a programming language specifically designed for implementing SPL approach [6]. The purpose of DOP is to provide expressive and flexible programming language for SPL. An SPL in DOP is implemented with a core module and a set of delta modules. Core module is a starting point of other products which will be created by application of delta module. Core module has a number of classes as the base of other products. The definition of base here is a product to be modified by delta to make a new product.

Delta modules specify changes in core module to implement other products. Those changes are in class level such as adding, removing, and modifying class. They can also be in class structure level such as adding and removing field or method.

2.3. LONTAR

Library Automation and Digital Archive (LONTAR) is a digital library system in Universitas Indonesia. This system manages common activities in library such as looking data about a collection, borrowing collection, returning collection, etc. LONTAR also provides web service to increase its interoperability for various libraries. The architecture of LONTAR can be seen in **Figure 1**.

Currently LONTAR only provides SOAP web service because LONTAR uses Apache Axis as shown in circle in **Figure 1**, which is by default only implements SOAP web service. In its development, there are needs for digital libraries to implement REST or other web service.

This paper tries to explore possibility for using ABS to make LONTAR has another web service such as REST. The plan is to take one specific LONTAR's module which is the web service module and convert it to SPL. With it, hopefully it will become easier to implement another web service. As illustrated, we tried to adjust the module in circle in **Figure 1** according to **Figure 2**.

3. Conversion Step

Steps from **Figure 3** can be generally used for every object oriented system. Meanwhile, steps in **Figure 4** are specifically used for LONTAR conversion from an object oriented system into an SPL one. In **Figure 4**, first two steps are the same as two first steps in **Figure 3**. Significant differences for steps in LONTAR are in third and fourth step. In **Figure 4**, step “process and generate” is elaborated more specifically. The triangular shapes

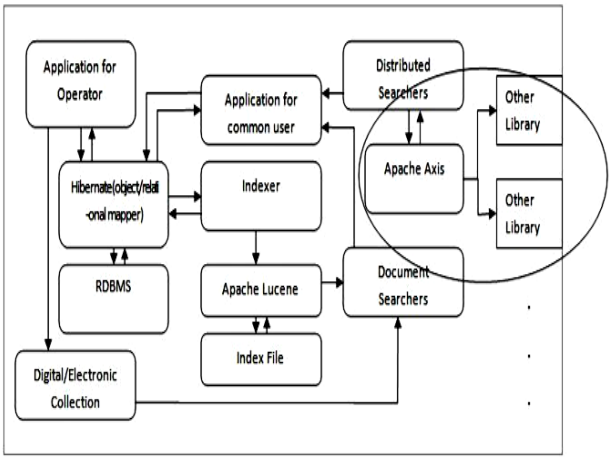

Figure 1. Architecture of LONTAR.

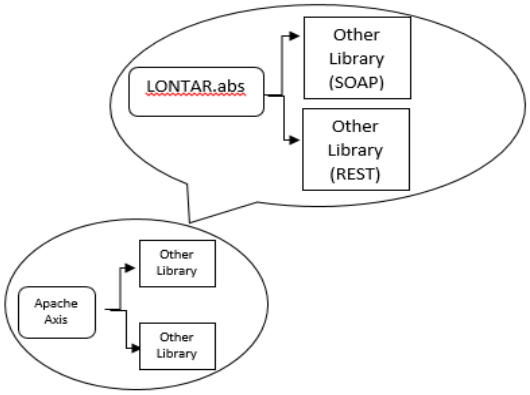

Figure 2. Replacement for one of LONTAR's module; Development plan for LONTAR.

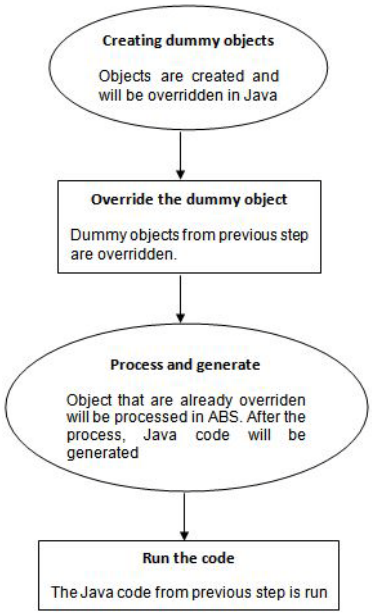

Figure 3. Conversion steps from object oriented system into SPL.

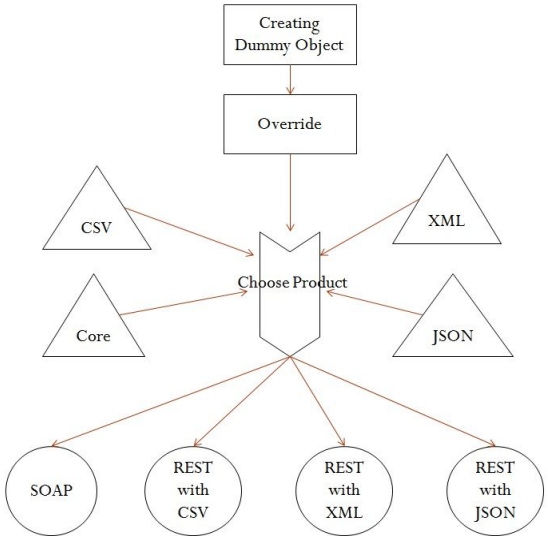

Figure 4. LONTAR's conversion from object-oriented into SPL.

symbolize existing deltas, while circles show products that can be created by applying those deltas. If no deltas are chosen SOAP product will be made. If CSV delta is chosen then the product will be REST with CSV and so on. Fourth and the last step in **Figure 4** is to run the product. Each step will be explained more thoroughly in the next section.

4. Experiments and Observation

4.1. Creating Dummy Object

Object is an instance of a class. Therefore, to create an object, a class must already be made. Dummy object serves as a purpose to allow LONTAR's data in Java to be processed further in ABS. After the class exists, dummy object can be created. It is called dummy object because it will be overridden eventually. The explanation of how it will be overridden is on the next subsection.

4.2. Override

In override phase, the first idea is directly using library of retrieval module to access and call LONTAR's index file. The purpose of this is to get the content of the index file. After received the contents, they are given back to ABS for the next step.

Nevertheless, this way is not running smoothly. ABS version used in this research per April 25th 2013 has not supported foreign libraries. ABS can only recognize libraries that are built-in from Java such as java.util and java.io. As a temporary solution, the genuine codes from LONTAR are copied to a Java program. This program's purpose is to write to a file content of the first ten index after doing query "software engineering" on LONTAR.

This file will be read, created object based on it, and given to ABS.

4.3. Choose Product

In this phase, features that are going to be realized in a product will be decided. These features are created from application of delta to core product.

To achieve that, the first thing to do is to create a list of string so a delta can be selected to choose appropriate message format. The names of variables with the content of them are passed to the aforementioned list. Method which this list is going to be printed is going to be modified by delta.

There are three deltas that can be chosen. They are DCSV to make a message with CSV format, DXML to create a message with XML format and DJSON for JSON format. All of them need a parameter in the form of list of string. This list is going to be processed so the message can be formed with appropriate format. Another similarity between the three deltas is that they create a list of string that its content is the string after being processed in delta.

Take one of the delta which is DJSON that can be seen in **Figure 5**. Delta DJSON takes all the elements of a list of string one by one and craft it to be in JSON format. After the deltas are made, next step is to create a file to control the product line. **Figure 6** contains the file to control the product line. It defines all the features such as PrintJSON, PrintXML, and PrintCSV with deltas that are applied for those features. DJSON for PrintJSON, DXML for PrintXML and DSCV for PrintCSV. The last step is to create a file which contains two things. First is the name of products that are going to be made. Second is the name of features each particular products are using. The example can be seen in **Figure 7**.

There are three products UseJSON, UseXML, and Use CSV. PrintJSON feature is used for product Use JSON, PrintXML is used for product UseXML and feature PrintCSV is used for product UseCSV. As the name suggests, product UseJSON will use feature PrintJSON and apply delta DJSON to core product and eventually print the data in JSON format. Same as the two products before, product UseCSV will use feature PrintCSV and print in CSV format.

Figure 8 shows the core product which none of the deltas is chosen. The message is printed in SOAP format. The format of the message conforms the format described in printing method in ABS before any deltas are applied. The result of using UseJSON product can be seen in **Figure 9**. The format of the printed message now becomes JSON format. This is happened due to delta DJSON has modified the printing method in ABS.

```
delta DJSON;
uses Lontar;

modifies class PrintMethodImpl {
    modifies List<String> print(List<String> l) {
        List<String> str = Nil;
        Int i=0;
        Int n = length(l);
        str = appendright(str,"{");
        str = appendright(str,"\n");
        while(i<n) {
            str = appendright(str,"\"");
            str = appendright(str,nth(l,i));
            str = appendright(str,"\"");
            str = appendright(str," : ");
            str = appendright(str,"\"");
            str = appendright(str,nth(l,i+1));
            str = appendright(str,"\"");
            str = appendright(str,"\n");
            i=i+2;
        }
        str = appendright(str,"}");
        str = appendright(str,"\n");
        return str;
    }
}
```

Figure 5. Delta LONTAR.

```
productline Lontar;
    features PrintJSON, PrintXML, PrintCSV;
    delta DJSON when PrintJSON;
    delta DXML when PrintXML;
    delta DCSV when PrintCSV;
```

Figure 6. LONTAR Product Line.

```
product UseJSON(PrintJSON);
product UseXML(PrintXML);
product UseCSV(PrintCSV);
```

Figure 7. LONTAR Products.

```
Console  Tasks
Java Output (coba_lontar)
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
<soap:Header>
</soap:Header>
<soap:Body>
        <m:GetidResponse>
                <m:id>3575</m:id>
        </m:GetidResponse>
        <m:GetcallnumberResponse>
                <m:callnumber>629.8.G.III </m:callnumber>
        </m:GetcallnumberResponse>
        <m:GetauthorResponse>
                <m:author>Francis H.Raven </m:author>
        </m:GetauthorResponse>
        <m:GetpublisherResponse>
                <m:publisher></m:publisher>
        </m:GetpublisherResponse>
        <m:GetlanguageResponse>
                <m:language></m:language>
        </m:GetlanguageResponse>
        <m:GettitleResponse>
                <m:title>Automatic control engineering </m:title>
        </m:GettitleResponse>
```

Figure 8. Core product.

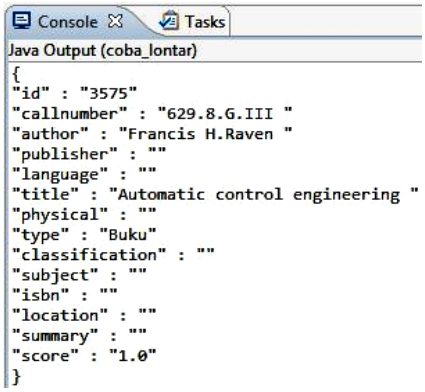

Figure 9. UseJSON product.

4.4. Observation

Table 1 consists of general comparison in LONTAR development between using ABS and using Object Oriented Approach.

Without ABS, to add other web service in LONTAR, new classes are needed and the functionalities are similar with classes that already in LONTAR. These new classes are used for implementing REST. New classes can be made as subclasses from the old ones. The difference is the new classes do not use Apache Axis because it only can be used for SOAP. The format of REST message should be defined in the new classes.

Meanwhile, the implementation of code reusability in ABS is not with inheritance but delta modeling. One of the reason to do this is because ABS uses strong encapsulation while inheritance can weaken encapsulation [7].

ABS uses strong encapsulation to avoid the data encapsulation weakening. In object oriented three kinds

of data encapsulation are known which are public, private and protected. In ABS all data are private. Encapsulation is even stricter than in object oriented where two objects even in the same class cannot access each other.

Another advantage in delta modeling is its abilities to do plug and play where users do not have to do complicated configuration when adding new classes. Users just have to select appropriate delta to apply to the core module. We can see the example in previous section. By just applying DJSON delta, the message is formatted in JSON format.

With SPL and ABS, to develop LONTAR is relatively easier. The development process in ABS becomes easier because developers can be more focused in modification with deltas. However, in object oriented system, customization for the new classes to work well with the old ones without ruining the system is necessary.

5. Concluding Remarks

Based on the experiment, OO System can be converted into SPL using delta modeling of ABS. This approach can be done in four steps, which are creating the dummy objects, override the dummy object, process and generate, and run the Java code from the previous step. On the LONTAR case study there was a problem with the current version of ABS plugin in step "process and generate" since LONTAR uses Apache Library while ABS only can process Java Library. Nonetheless, that is only technical issue. Theoritically, ABS has no problem just like OO. To resolve the problem, we have found a temporary workaround. Further development of ABS however is required.

Table 1. Table General Comparison between ABS and Object Oriented.

No	Object Oriented	ABS
1	Create new class	Create new product
2	Inheritance	Delta Modeling
3	Data Encapsulation can be public, private or protected	Every data is private
4	Class defines object type	Class does not define object type

Delta modeling has another opportunity to be used in another aspect of OO which is refactoring, one of technique for restructuring internal structure without changing its external behavior [8]. Refactoring can be used in restructuring code, like Java Code, or restructuring model, like UML Class Diagram. Delta modeling is used to modify code in ABS without changing its core ABS code. Thus, refactoring and delta modeling have similar purpose on modifying code or model. However, delta modeling can be verified but refactoring naturally cannot. If refactoring can be represented in delta modeling, it will have an opportunity to be verified.

Currently, we are developing translation mechanism from UML Class Diagram to ABS. This mechanism can help to create delta modeling representation of refactoring. We will create the complete process as the future work to explore another use of delta modeling in Object Oriented System.

Acknowledgements

This paper is part of research entitled "Development of Highly Adaptable and Reliable Software Development Tool to Support Requirement Changes and Variation of User Needs" which is led by Ade Azurat and fully funded by Penelitian Unggulan Perguruan Tinggi (Hibah Internal Fasilkom UI BOPTN Tahun 2013). Author also would like to give special thanks for Reiner Hahnle, Richard Bubel, Yannick Welsch and Radu Muschevici for their helpful advises on this research.

REFERENCES

[1] D. Cansell and D. Mery, "Foundations of the B Method," *Computing and Informatics*, Vol. 22, 2003, pp. 221-256.

[2] M. Eriksson and A. Hagglunds, "An Introduction to Software Product Line Development," *Proceedings of Ume's Seventh Student Conference in Computing Science*, 2003.

[3] R. Hahnle, "The Abstract Behavioral Specification Language: A Tutorial Introduction," TU Darmstadt, Darmstadt, 2012.

[4] P. C. Clements and L. Northrop, "Software Product Lines: Practices and Patterns," SEI Series in Software Engineering, Addison-Wesley, Boston, 2001.

[5] L. Northrop, "Software Product Line Essentials," Carnegie-Mellon University, Pittsburgh, 2008.

[6] I. Schaefer, *et al.*, "Delta-Oriented Programming of Software Product lines," 2010. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.221.4371

[7] A. Snyder, "Encapsulation and Inheritance in Object-Oriented Programming Lanugages," ACM, New York, 1986.

[8] M. Fowler, "Refactoring Improving the Design of Existing Code," Addison-Wesley, Boston, 2011.