Automatic Model Translation to UML from Software Product Lines Model using UML Profile

Rizki Muhammad
Fakultas Ilmu Komputer
Universitas Indonesia
Email: r.rizkimuhammad@gmail.com

Maya Retno Ayu Setyautami
Fakultas Ilmu Komputer
Universitas Indonesia
Email: mayaretno@cs.ui.ac.id

***Abstract*—Software Product Lines (SPL) enable a software to have various products in single development. The products possess commonality and variability that should be defined in the problem domain. Abstract Behavioral Specification (ABS) is one of executable modeling language that supports SPL by implementing Delta Oriented Programming (DOP). In DOP, features that is related with the variability will be implemented in the delta modules (deltas). Deltas will modify a basic product to create (new) various products. Thus the various features and products will be managed well in delta modeling. On the other hand, there is Unified Modeling Language (UML), a standard and popular modeling language. UML is not designed to model SPL, but UML has a mechanism to extend their syntax and semantics by defining UML Profile. In this paper, we aim to bridge UML and SPL automatically by having an automatic traslation program. The program will produce UML model based on ABS model, that supports SPL, by using UML-DOP Profile. Besides connecting UML and SPL, the program can also help the developer to achieve coherency between design and implementation. As the results, the UML models produced by automatic translator are represented by XML Metadata Interchange (XMI) documents.**

I. Introduction

In software development life cycle, modeling phase is an important stage before the implementation. Requirement model is one part of this phase related with the user needs [1]. Any error that happen in this phase may produce inappropriate software because the requirement model is not in line with the user requirement. Unified Modeling Language (UML) is a modeling language that is widely used in software development. UML has two types of diagram, structure diagram and behavior diagram [2]. Each types of diagram consists of several diagrams, such as UML Class Diagram in structure diagram.

Requirement modeling in software engineering should not only consider to the correctness but the effectiveness and efficiency. Some research has been conducted to produce a new paradigm in developing a software, one of them is Software Product Lines (SPL). SPL creates a software based on the commonality and variability [3]. It can have various products in a single development. There are two processes in the SPL development, *domain enginering*, where the requirement commonality and variability are defined, and *application engineering*, where the applications are built based on requirement artifacts.

Delta Oriented Programming (DOP) [4] is a novel programming paradigm that supports SPL by defining features and products variation based on delta modeling. Software variations in DOP are defined in delta modules, or deltas for short. Commonality or basic products are defined in the *core* modules. Variability that is described by various features are implemented by deltas without modifying the existing feature defined in *core* modules. One of modeling languages that implements DOP is Abstract Behavioral Specification (ABS) [5]. ABS is an executable object oriented modeling language that was produced by HATS Project [6]. It has five language layers to model SPL that will be explained in Section II.

As a standard modeling language, UML notation support does not support SPL. However, Object Management Group (OMG) has a mechanism to extend UML syntax and semantics by defining UML Profile. In the previous research, UML Profile that supports SPL has been created based on DOP, called UML-DOP Profile [7]. Using a UML-DOP profile, each element in ABS is mapped to UML element so UML can visualize SPL model. The detail will be explained in Section III-A.

UML profiles may be used in the language transformations with a little work [8]. In this paper, we will use UML-DOP profile as a basic rules for automatic transformation from ABS model, that supports SPL, to UML model. The translation mechanism is done by modifying the ABS compiler to produce the XMI document that represent the UML models. The detail process of translation from ABS model to XMI model will be explained in Section IV.

In this research, UML-DOP profile is used as a general guide or rule to model SPL in UML. This paper presents an automatic approach for doing translation process from ABS, that supports SPL, to UML. The automatic translation program will help a software developer that models a system with SPL paradigm to create the equivalent UML diagram. Moreover, ABS is an executable modeling language that can be generated into running code, such as Java [9]. Therefore, by having the UML generator from ABS, the coherency between design in UML and the implementation code could be improved.

II. Abstract Behavioral Specification

Abstract Behavioral Specification (ABS) is an object oriented modeling language that implements DOP and supports

SPL. ABS is an executable modeling language because it is also equipped with ABS tool that can automatically generate running code such as Java, Haskell, and Erlang [9]. In order to develop various product in SPL, ABS has the following five languages layer [5]:

1) Core ABS
   Core ABS is a main layer defines software behavior based on object oriented modeling [9]. It has similar syntax with Java, excludes inheritance and overloading. However, code reuse in ABS can be achieved by delta modeling mechanism as in DOP concept. A core ABS has a module declaration which is followed by interfaces, classes, and optional main block [10].
2) Micro Textual Variability Language(μTVL)
   μTVL is a language derived from Textual Variability Language (TVL), where μTVL is used in ABS modeling framework to represents a features model. It has similar meaning with a feature diagram. It can provide boolean or integer constraints to the feature. ABS tool is also equipped by μTVL checker that helps developer to validate the products variation.
3) Delta Modeling Languange (DML)
   Delta Modeling Language (DML) supports ABS to represent DOP in developing SPL. In DOP, delta is defined to implement a specific feature. One feature can be implemented by or more deltas and one delta can be used by one or more features. Delta in ABS modifies the core ABS to generate various products. Delta can modify properties in core ABS, such as interfaces, classes, methods, or attributes.
4) Product Line Configuration Product Line Configuration Language (CL) manages the relationship between feature model and delta. Each feature can have one or more deltas and each delta may also relate to one or more features. CL defines that relationship to describe when a feature is chosen, which delta will be applied to Core ABS. The mechanism can help the user in the debugging process since the configuration of variability system is described in this CL module.
5) Product Selection Language
   Product Selection Language (PSL) defines the products that will be generated based on the specific features. To create a product, we define the product name and the list of features that is required. ABS tool help the verification of the selected product to guarantee that the product complies to the constraints in μTVL. Therefore, logical error in ABS can be check in compile time.

A. Code Generator in ABS Compiler

ABS compiler consists of two parts, front-end compiler and back-end compiler. It can be used to generate code from ABS model into another code. The mechanism for code generator in ABS compiler can be seen at Fig. 1.

Based on that figure, ABS front-end compiler is using ANTLR (ANother Tool for Language Recognition) to parse ABS Code into AST. ANTLR will parse ABS Code by

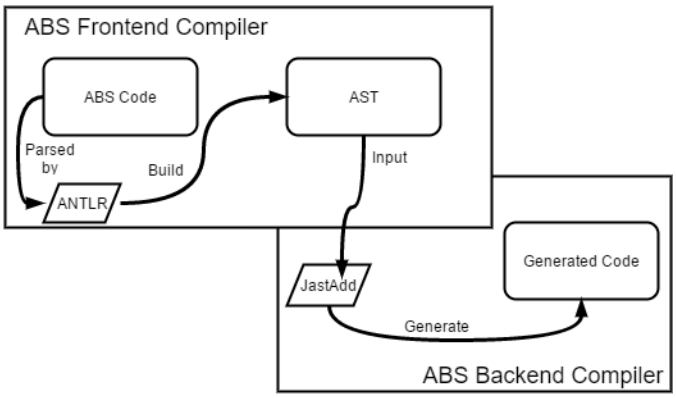

Fig. 1. ABS Compiler

matching its input with ABS Grammar and then ANLTR will build AST based on that. In ABS, its definition of AST can be seen at ABS.ast and mTVL.ast where the AST nodes is defined as an abstract class that later can be extracted by utilizing JastAdd aspects. The AST, that was built by ANTLR in ABS front-end compiler, then will be used by ABS backend compiler to generate code by utilizing JastAdd aspects to retrieve the information and process it. An example of ABS definition of AST and the information can be seen at Fig. 2. Based on the figure, ABS AST has a compilation unit that consists of a list of components that represent ABS element, such as *delta module*, *product line*, and *feature*. Each component can have several child component. For example, component *FeatureDecl* has a child component *AttrConstraint*.

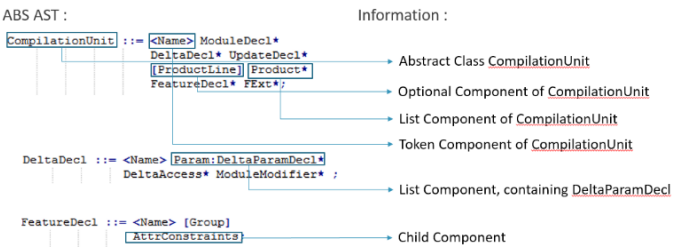

Fig. 2. Example of ABS AST Definition

III. Unified Modeling Language

UML is a modeling language that is widely used in software development often involve a variety of applications. The interaction and exchange between applications is done with using the XML Metadata Interchange (XMI). XMI provides standardization in representing the metadata of a UML model. It can be used in other applications that have the ability to process XMI using Extensible Markup Language (XML) to move the information.

Although it has been bridged by XMI, it does not mean UML can be used to model all problems and programming languages. UML is a general purpose notation, so it is not always able to model elements that specifically only in a particular programming language [11]. These problems can

TABLE I
UML-DOP PROFILE STEROTYPES [7]

DOP Element Name	UML Base Class	Stereotype Name
Module	Package	`<<module>>`
Delta Module	Package	`<<delta>>`
Delta Parameter	Property	`<<deltaParam>>`
Module Modifier	Association	`<<adds>>`, `<<removes>>`, `<<modifies>>`
	Class	`<<modifiedClass>>`
	Interface	`<<modifiedInterface>>`
Modifier	Operation; Property	`<<adds>>` `<<removes>>` `<<modifies>>`

be solved by defining UML Profile to specific concept. It can be done by modeling the elements on domain-specific language with the search for the most similar to the model representation UML which is then used as the base metaclass [12]. For example, OMG published a UML Profile based on Java syntax [13].

A. UML-DOP Profile

UML-DOP Profile is a UML Profile which is defined based on Delta Oriented Programming (DOP) [7]. By having this profile, UML can be used to model SPL. The UML-DOP profile is defined based on ABS syntax. Each DOP element is mapped to UML element, as seen in Table I. Each element in DOP will be mapped to element in UML metaclass as a stereotype. For example delta in DOP is modeled as UML package with stereotype `<<delta>>`.

Fig. 3 shows an example of UML Diagram with UML-DOP profile that models an Account ABS problem [10] for a `Checking Account` product. The core module `Account` is represented as a UML package with stereotype `<<module>>` that consists of interface `Account` and class `AccountImpl`. There are two delta modules, delta `DType` that implements feature `Type` and delta `DCheck` that implements feature `Check`. A delta `DCheck` is represented by UML package `DCheck` with stereotype `<<delta>>` and a feature `Check` is represented by UML component `Check` with stereotype `<<feature>>`. The last one is `CheckingAccount` product that is represented by UML component with stereotype `<<product>>`.

IV. ABS-UML TRANSLATION PROCESS

The translation process form ABS model into UML model is done by modifying the ABS back-end compiler, as seen in Fig. 4. We modified ABS compiler in Fig. 1 by adding the new JastAdd aspect that will act as a part of ABS backend compiler.

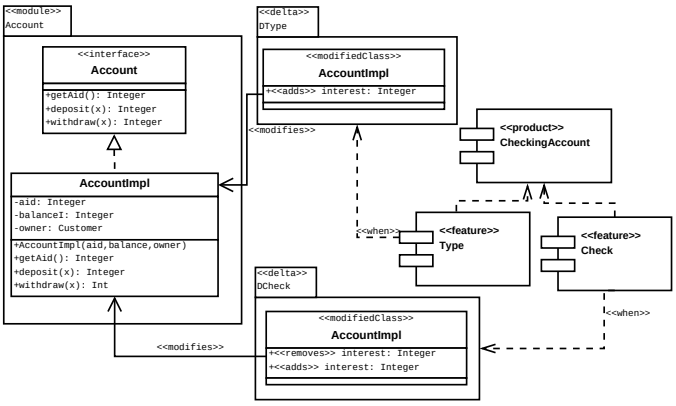

Fig. 3. Account UML Diagram with UML-DOP Profile

The new JastAdd aspect will process the AST generated by ANTLR in the ABS frontend compiler and then process it within the JastAdd aspect. The translation process that happens within the JastAdd aspect will apply rules and stereotypes defined in UML-DOP Profile. It will translate ABS model into UML model based on UML-DOP Profile, in the form of XMI document. We use Eclipse Modeling Tools that is equipped with UML diagram editor (Papyrus) and ABS compiler.

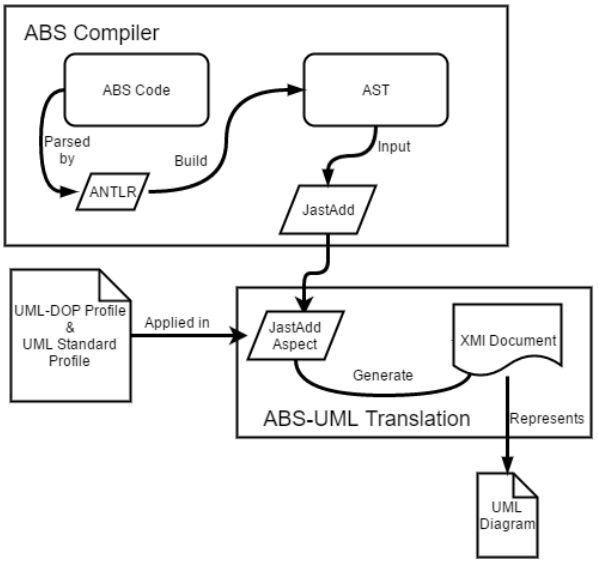

Fig. 4. Translation Process

Generally, there are four steps in developing the translation program, such as:

Step 1: Building XMI Template

The first step of the translation process is building the XMI template for the output. In this step, the program will define the order of the UML element in the document and any other configurations, such as applying UML profiles and defining namespaces used in the document. The XMI format follows the XMI document that is used in Papyrus UML Editor. A snippet code of XMI template can be seen in Fig. 5.

This template is used as the he result of this template will be the translation output (with `.uml` extension). It starts with

```
<?xml version="1.0" encoding="UTF-8"?>
<xmi:XMI xmi:version="20131001" [namespaces]">
  <uml:Model xmi:id="_absuml" name="abs-uml">
    ..... [profileApplications]
    ..... [core modules]
    ..... [delta modules]
    ..... [features]
    ..... [products]
    ..... [associations]
  </uml:Model>
  ..... [stereotypes]
</xmi:XMI>
```

Fig. 5. Template of Generated XMI

the XML version declaration, namespaces, profile application indicates that the UML should used the UML-DOP profile, and the ABS modules.

Step 2: Extracting Information from Abstract Syntax Tree (AST)

By using information from ABS AST definition, as shown in Fig. 2, the translation process can extract information from each component of abstract classes in the AST. The translation process begin by extracting the information from `CompilationUnit` that has the information of `ModuleDecl`, `DeltaDecl`, `ProductLine`, `Product`, and `FeatureDecl` as seen at Fig. 2. To extract that information, the program will be using getter methods of each abstract syntax (abstract classes and its component). For example, to extract information from ModuleDecl component the translation program will use `getModuleDecls` as seen at Fig. 6. It is possible to extract information of abstract classes from within the JastAdd aspect, because it supports*intertype* declaration for AST classes. *Intertype* declaration is a declaration that appears in an aspect file, but actually belongs to an AST class[1].

```
for(ModuleDecl d : getModuleDecls()){
        if(!Constants.BUILT_IN_LIBS.contains(d.
            getName()) && !d.getName().contains("ABS
            .")){
        d.generateXMI(stream, imports);
        modules.add("_M" + d.getName());
        }
}
```

Fig. 6. Example of Extracting Information

Step 3: Building XMI Document

After extracting information from the AST, the translation program will compose it as a valid document by following the template defined before. At this step of translation, the order of the UML element in the document is defined. The order of is core ABS module, delta module, product line configuration, product selection, feature model, and stereotypes based on UML-DOP Profile. The translation process will generate XMI document with a specific structure. The snippet of XMI structure can be seen at Fig. 7. In the front of UML component there are stereotypes that were already defined in UML-DOP Profile. For example, three packages with stereotype `<<module>>`, namely `MAccount`, `MCustomerIF`, and `MCustomerData` represent core modules in ABS.

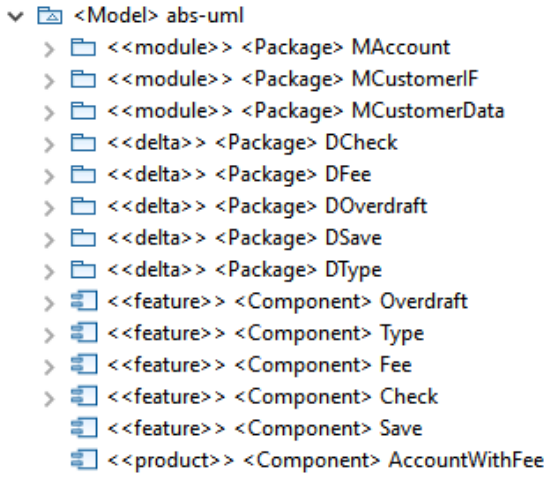

Fig. 7. Structure of The Generated XMI

Step 4: Applying UML-DOP Profile as Translation Rules

The last step is applying UML-DOP Profile as translation rules. In this step, the program will be assigning each ABS element into its corresponding meta-class and stereotype defined in UML-DOP profile. Therefore the ABS model can be modeled as UML model completely. An example can be seen at Fig. 8, the translation program will process ProductLine element from ABS model and applying few stereotypes, which is `<<feature>>` with base metaclass UML component, and also stereotype `<<when>>` and `<<after>>` that has base metaclass UML dependency.

```
public void Model.generateXMI(PrintWriter stream){

//List of all features in product line
ArrayList<String> features = new ArrayList<String>()
    ;
HashMap<String, ArrayList<String>> productline = new
     HashMap<String, ArrayList<String>>();
stream.println("<uml:Model xmi:id=\"_absuml\" name
    =\"abs-uml\">");

if(hasProductLine()){
getProductLine().generateXMI(features, productline);
stream.println("</uml:Model>");

for(String f : features){
stream.println("<ABSProfile:feature xmi:id=\"__feat"
     + f + "\" base_Component=\"_feat" + f + "\"/>")
    ;}

stream.print("</xmi:XMI>");
stream.close();}
```

Fig. 8. Snippet Code Applying UML-DOP Profile

V. Evaluation

Evaluation of automatic translation from ABS to UML is done by applying the translation program to case studies.

[1]http://jastadd.org/old/manual/aspects.php

We choose Bank Account case study that has been modeled in ABS [10]. We also choose AISCO (Adaptive Information System for Charity Organization) as a real world case study.

A. Bank Account Case Study

Bank Account case study modeled a various account product, such as Checking Account, Saving Account, Account with Overdraft, and Account with Fee. Basically that product has similar function to calculate balance of account based on deposit and withdrawal. Bank Account case study ABS model has 3 core modules, 5 delta modules, 1 feature model, 1 product line configuration, and 1 product selection language. Manually, we can create the UML model as seen in Fig. 3.

To evaluate the translation result, we compare the XMI document, that is created manually on the Papyrus UML Editor based on Fig. 3, with the generated XMI document that is produced from our translation process. As as result, we found that the XMI has similar structure with different order. The example of evaluation process by comparing the XMI result is showed in Fig. 9.

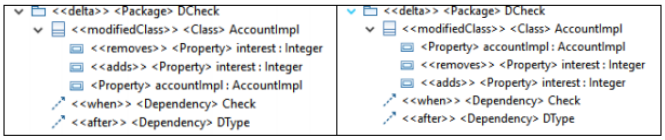

Fig. 9. Left (XMI based on Papyrus), Right (XMI Based on Translation Process)

B. AISCO Case Study

AISCO case study is bigger than Bank Account. Basically different charity organization system has a similar purpose to report their program and maintain the financial report. We modeled this case in the feature diagram Fig. 10 to get product variations. This case study already modeled in ABS https://gitlab.com/IS4Charity/aisco-abs . It consists of 6 core ABS modules, 16 delta modules, also a feature model, product line configuration, and product selection.

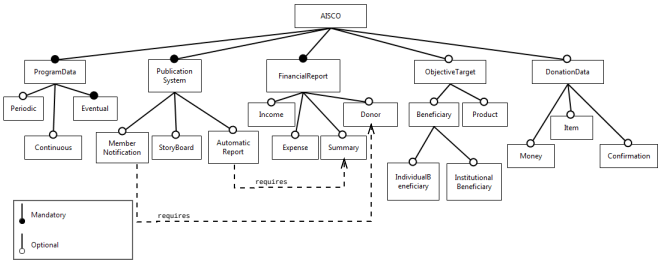

Fig. 10. Feature Diagram of AISCO Case Study

The ABS model is used as an input for the translation program. Based on the translation result, there is a XMI document that consists of all ABS modules. The example XMI result for delta module `DEventual` is shown in the snippet code Fig. 11. Delta is represented as a UML package that contains a modified class. Line 17-21 shows the profile application based on UML-DOP stereotypes. For example UML package DEventual uses stereotype *delta* indicating that the package represents the delta module.

```
1  <packagedElement xmi:type="uml:Package" xmi:id="_DDEventual" name="DEventual">
2    <packagedElement xmi:type="uml:Class" xmi:id="_modProgramImplDEventual" name=
         "ProgramImpl">
3      <ownedAttribute xmi:type="uml:Property" xmi:id="_propProgramImplDEventual"
           type="_ProgramImpl" association="_ascProgramImplDEventual"/>
4      <ownedAttribute xmi:type="uml:Property" xmi:id="_addpicDEventual" name="pic
           ">
5        <type xmi:type="uml:PrimitiveType" href="pathmap://UML_LIBRARIES/
             UMLPrimitiveTypes.library.uml#String"/>
6      </ownedAttribute>
7      <ownedOperation xmi:type="uml:Operation" xmi:id="_addsetPICDEventual" name=
           "setPIC">
8        <ownedParameter xmi:type="uml:Parameter" xmi:id="_rtnsetPICDEventual"
             name="Unit" direction="return" effect="read" type="_Unit"/>
9        <ownedParameter xmi:type="uml:Parameter" xmi:id="_paramsetPIC_person"
             name="person" effect="update">
10         <type xmi:type="uml:PrimitiveType" href="pathmap://UML_LIBRARIES/
               UMLPrimitiveTypes.library.uml#String"/>
11       </ownedParameter>
12     </ownedOperation>
13   </packagedElement>
14   <packagedElement xmi:type="uml:Dependency" xmi:id="_whenDEventual" client="
         _DDEventual" supplier="_featEventual"/>
15 </packagedElement>
16 <!--Profile Application-->
17 <ABSProfile:delta xmi:id="__DDEventual" base_Package="_DDEventual"/>
18 <ABSProfile:modifiedClass xmi:id="__modProgramImplDEventual" base_Class="
       _modProgramImplDEventual"/>
19 <ABSProfile:adds xmi:id="__addpicDEventual" base_Property="_addpicDEventual"/>
20 <ABSProfile:adds xmi:id="__addsetPICDEventual" base_Operation="
       _addsetPICDEventual"/>
21 <ABSProfile:modifies xmi:id="__ascProgramImplDEventual" base_Association="
       _ascProgramImplDEventual"/>
```

Fig. 11. XMI Output-Delta DEventual

We can compare the XMI result with the UML Diagram that is created manually. Delta DEventual is represented in the UML Diagram Fig. 12 below. There is a UML Package with stereotype `<<delta>>` DEventual that consists of UML Class with stereotype `<<modifiedClass>>` ProgramImpl. This result is similar with the XMI output that is produced by the translation program.

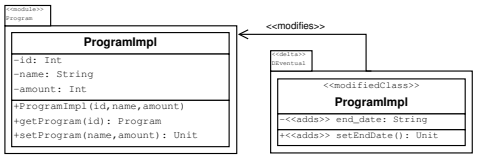

Fig. 12. UML Representation of Delta Module

Fig. 13 shows that there are 16 UML Package with stereotypes *delta* in the XMI output. This is in accordance with the number of ABS delta module in AISCO case study. Based on our evaluation, all ABS modules can be translated into UML that is represented by XMI document.

VI. Related Works

The idea to use a UML profile to support model transformation appears also in [8]. They introduce supporting profiles as first class language definitions within ATL transformations. The objective is to bridge the structural modeling part of AllFusion Gen (AFG) and UML Class Diagram. AllFusion Gens data model is based on ER modeling concepts such as Entity Types, Attributes, and Relationships. ATL focuses on model transformation.

```
<ABSProfile:delta xmi:id="__DDAutomaticReport" base_Package="_DDAutomaticReport"/>
<ABSProfile:delta xmi:id="__DDConfirmation" base_Package="_DDConfirmation"/>
<ABSProfile:delta xmi:id="__DDContinuous" base_Package="_DDContinuous"/>
<ABSProfile:delta xmi:id="__DDDonor" base_Package="_DDDonor"/>
<ABSProfile:delta xmi:id="__DDEventual" base_Package="_DDEventual"/>
<ABSProfile:delta xmi:id="__DDExpense" base_Package="_DDExpense"/>
<ABSProfile:delta xmi:id="__DDIncome" base_Package="_DDIncome"/>
<ABSProfile:delta xmi:id="__DDIndividualBeneficiary" base_Package="
    _DDIndividualBeneficiary"/>
<ABSProfile:delta xmi:id="__DDInstitutionBeneficiary" base_Package="
    _DDInstitutionBeneficiary"/>
<ABSProfile:delta xmi:id="__DDItem" base_Package="_DDItem"/>
<ABSProfile:delta xmi:id="__DDMoney" base_Package="_DDMoney"/>
<ABSProfile:delta xmi:id="__DDNotification" base_Package="_DDNotification"/>
<ABSProfile:delta xmi:id="__DDPeriodic" base_Package="_DDPeriodic"/>
<ABSProfile:delta xmi:id="__DDProduct" base_Package="_DDProduct"/>
<ABSProfile:delta xmi:id="__DDStory" base_Package="_DDStory"/>
<ABSProfile:delta xmi:id="__DDSummary" base_Package="_DDSummary"/>
```

Fig. 13. XMI Output-Delta Modules

In this research, UML-DOP profile, that has been defined in [7], is used as a basis rules for translating ABS model to UML Model. They evaluated the profile completeness by applying to AISCO case study manually. They mapping each element in DOP to element in UML and draw the UML Diagram with UML-DOP Profile. We use the same case study (AISCO) with [7] to compare the output generated by translation program.

VII. Conclusion and Future Works

The automatic translation from ABS model into a UML model can be performed by utilizing ABS AST and UML-DOP Profile. The information based on AST can be retrieved by JastAdd can be be processed by the translation program to produce a UML diagram that is represented as an XMI. The rules in the translation program are created based on the mapping element in the UML-DOP Profile. Based on case study Bank Account in Section V, all the ABS models can be mapped into XMI documents. Therefore the case study that is modeled based on SPL approach can have the equivalent UML models.

In this research, the UML models produced by the translation program are represented in XMI documents. The UML diagram that visualize the XMI document is not generated by the translation program. On the next research, this problem can be integrated into the translation program. It is also good idea if the translation program can be integrated with the UML editor. By having the integration, changes in ABS models will automatically change the UML models.

Acknowledgment

The authors would like to thank to Dr. Ade Azurat, Prof. Reiner Hähnle and Dr. Radu Muscheivici as supervisors in this research. This research is part of PRICES (Precise Requirement Change Integrated System) project.

References

[1] R. S. Pressman, *Software Engineering: A Practitioner's Approach, 7th Ed*. McGraw-Hill, 2010.

[2] *Unified Modelling Language: Superstructure*, Object Management Group, 2004.

[3] K. Pohl, G. Böckle, and F. Van Der Linden, *Software Product Line Engineering: Foundations, Principles, and Techniques*. Springer, 2005.

[4] I. Schaefer, L. Bettini, V. Bono, F. Damiani, and N. Tanzarella, "Delta-oriented programming of software product lines," in *Proc. of 14th Software Product Line Conference (SPLC 2010)*, Sep. 2010.

[5] E. B. Johnsen, R. Hähnle, J. Schäfer, R. Schlatte, and M. Steffen, "ABS: A core language for abstract behavioral specification," in *Proc. 9th Intl. Symp. on Formal Methods for Components and Objects FMCO)*, ser. LNCS, B. Aichernig, F. S. de Boer, and M. M. Bonsangue, Eds., vol. 6957. Springer, 2011, pp. 142–164.

[6] "Highly Adaptable and Trustworthy Software using Formal Models," Mar. 2009, http://www.hats-project.eu.

[7] M. R. Setyautami, R. Hähnle, R. Mushceivici, and A. Azurat, "Uml profile for delta-oriented programming to support software product line engineering," in *Proceedings of the 20th International Systems and Software Product Line Conference (SPLC 2016)*.

[8] M. Wimmer and M. Seidl, "On using UML profiles in ATL transformations," in *1st International Workshop on Model Transformation with ATL*, 2009.

[9] P. Y. H. Wong, E. Albert, R. Muschevici, J. Proença, J. Schäfer, and R. Schlatte, "The ABS tool suite: modelling, executing and analysing distributed adaptable object-oriented systems," *Journal on Software Tools for Technology Transfer*, vol. 14, no. 5, pp. 567–588, 2012.

[10] R. Hähnle, "The abstract behavioral specification language: A tutorial introduction," in *11th International Symposium, FMCO 2012*, 2012.

[11] L. Fuentes-Fernandez and A. Vallecillo-Moreno, "An Introduction to UML Profiles," *The European Journal for the Informatics Professional*, vol. V, pp. 6–13, 2004.

[12] J. Pardillo and C. Cachero, "Domain-specific language modelling with uml profiles by decoupling abstract and concrete syntaxes," *Journal of Systems and Software*, vol. 83, no. 12, 2010.

[13] *Metamodel and UML Profile for Java and EJB Specification*, Object Management Group, 2004.