Extending Unified Modeling Language to Support Aspect-Oriented Software Development

Rehab Allah Mohamed Ahmed
Computer Science Department,
Faculty of Computers & Information,
Helwan University, Cairo, Egypt

Amal Elsayed Aboutabl
Computer Science Department,
Faculty of Computers & Information,
Helwan University, Cairo, Egypt

Mostafa-Sami M. Mostafa
Professor of Computer Science, HCI Lab Member
Faculty of Computers & Information,
Helwan University, Cairo, Egypt

***Abstract*—Aspect-Oriented Software Development (AOSD) is continuously gaining more importance as the complexity of software systems increases and requirement changes are high-rated. A smart way for making reuse of functionality without additional effort is separating the functional and non functional requirements. Aspect-oriented software development supports the capability of separating requirements based on concerns. AspectJ is one of the aspect-oriented implementations of Java. Using Model Driven Architecture (MDA) specifications, an AspectJ model representing AspectJ elements can be created in an abstract way with the ability to be applied in UML, Java or XML. One of the open source tools which support MDA and follows the standards of the Object Management Group (OMG) for both UML and MDA is Eclipse providing an implementation of MDA through Eclipse Modeling Framework (EMF). This paper focuses on creating a UML profile; a UML extension which supports language specifications for AspectJ using EMF. Our work is based on the latest UML specification (UML 2.5) and uses MDA to enable the inclusion of aspect-oriented concepts in the design process.**

Keywords*—Aspect-Oriented Software Development; Model Driven Architecture; Eclipse Modeling Framework; Object Management group; UML; AspectJ

I. Introduction

Nowadays, software development complexity is continuously increasing and the need for non-functional requirements has become mandatory. This has led to various problems concerning the code of existing systems. Examples of such problems are redundancy and maintainability. Aspect-oriented software development came with solutions to such problems. A key concept in AOSD is by redefining the concerns into separate aspects where each aspect supports an individual concern. Typical examples of aspects are logging, security, and persistence [1]. Using AOSD, the need to include logging and security validation in each functionality is not necessary anymore. Both of the log and security aspects scan the code to perform what each of them is created for.

AOSD has its own terminology. An *aspect* refers to a specific concern. A *pointcut* specifies the condition that will be used to execute an aspect. A *joinpoint* refers to the program segment that satisfies the pointcut condition. An *advice* is a function that defines the behavior to use when a specific joinpoint is executed. The *weaving process* defines the manner in which the aspect code is combined with the base code so that they can be run together [1] [2].

Aspect-oriented processing can be applied on starting projects as well as existing projects. Various studies have focused on how to combine the aspect-oriented process with the software development process at different stages. Early research covered the requirement gathering process and how to perform separation of concerns. A formal way to convert the requirements to concerns and find the link between concerns has been presented [3]. In the design phase, aspects can be supported through formal languages such as UML and related tools [4]. A number of research projects have been conducted in the area of incorporating aspect-oriented concepts in the implementation stage of the software development process. In this respect, a number of programming languages such as C++ and Java have been extended to support aspect-oriented implementations yielding new languages such as AspectC++ and AspectJ [5] [6] [7]. Adding aspect-oriented concepts to the software development design process requires some changes in the design process for aspect identification and design as well as program naming standards leading to a new generic aspect-oriented design process [2].

This paper presents an aspect-oriented representation using the UML extension mechanism with the abstraction of MDA. Section II introduces Model Driven Architecture abstraction mechanism and UML standards related to it. Section III presents previous work on UML extension mechanism supporting aspect-oriented development. Section IV presents our proposed UML extension. Finally, section V presents the conclusion and future work.

II. MDA Abstraction Mechanism

As software systems increase in complexity, the demand for abstraction increases. Moreover, the need for separation of the business domains, implementation, and the platform dependency becomes mandatory. MDA supports such abstraction on three levels: one representing business context, second is a platform independent model (PIM), and the third is a platform specific model (PSM) [8].

Model driven architecture (MDA) is an approach to system specifications and system interoperability based on the use of formal models introduced by OMG (Fig. 1). MDA separates the specifications of a system from platform technology. Computation Independent Model (CIM) specifies the function of the system without getting into the construction details (Fig. 2). Platform independent model (PIM) specifies the construction of the system without implementation details.

Platform Specific Model (PSM) expresses the system details related to implementation platform using Domain Specific language (DSL) to transform it into different languages. MDA aims to develop modeling specifications once and target multiple technology implementations [12] [9] [13] [22].

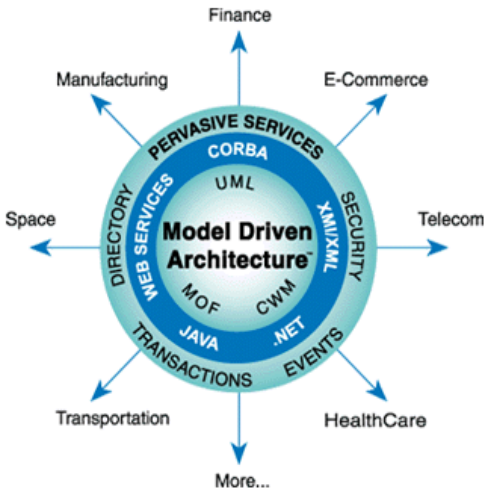

Fig. 1. Model Driven Architecture [9]

Fig. 2. Model Driven Architecture Layers [8]

A. OMG and UML

Object Management Group (OMG) is a nonprofit organization that develops technology standards for UML, MDA and other software engineering concepts. The UML standard specifications facilitate exchange between different tools * [9] [10].

OMG UML2.X specifications consist of four parts [11]:

1) Superstructure which defines the elements of the diagrams.

2) Infrastructure which defines the core metamodel of the superstructure.

3) Object Constraint Language (OCL) which defines rules for model elements.

4) XML MetaData Interchange (XMI) which defines an XML format for the exchange of UML models.

UML architecture is built up using the Meta Model Library called Meta Object Facility (MOF) based on a 4-layer Metamodel Architecture as shown in Fig. 3.

Infrastructure is used at both M2 and M3 levels of Fig.3. UML and MOF are both built based on the infrastructure with additional properties to UML. MOF defines how UML models interchange between tools using XMI. The superstructure is specified by UML to deal with structural and behavioral modeling [11].

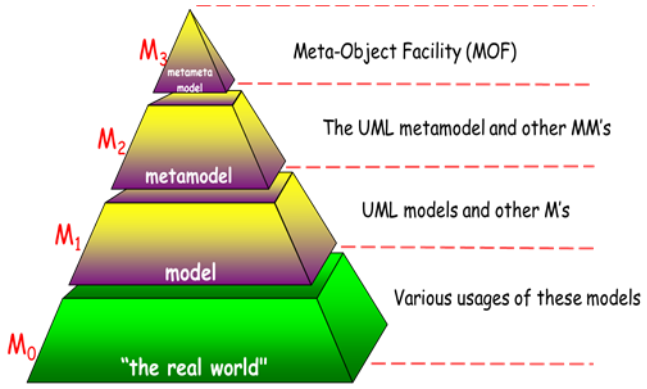

Fig. 3. 4-Layer Metamodel Architecutre [15]

UML provides an extensibility mechanism in two ways. The first is by creating *profiles* to customize the language for particular platforms and domains. The second is to create a new language related to UML using the Infrastructure library (define new Metamodel) but this requires specific environment modifications and handling. [11]

UML Profile cannot change the semantics of UML elements; it is used when customizations of UML are required for specific application domains. OMG have standardized several existing UML profiles for specific domains like CORBA, EJB. UML Profiles define both PIM and PSM in MDA, as in the CORBA UML profile, which defines the mapping from a PIM to a CORBA-specific PSM.

B. Eclipse Modeling Framework (EMF)

EMF is a framework and code generation facility that enables the definition of a model in any of these forms (Java, XML, and UML) and generates it in any of the three forms as in Fig 4. It is a technology moving in the direction of MDA as it is used to define the specification and separate it from the platform and the language representation. EMF is considered to be an MDA implementation supporting metamodel, but it has no Workgroup support, and does not fully comply with MOF standard. It has its Ecore which is close to MOF [14].

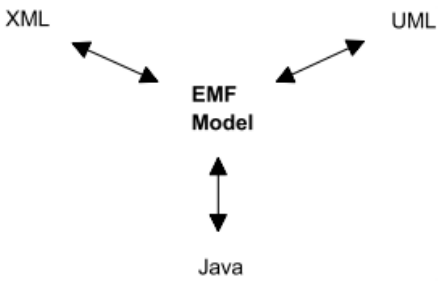

Fig. 4. EMF unifies UML, Java, and XML [14]

III. Previous Work

As a challenging concept in software development, AOSD has been under focus in various phases of the software

* OMG UML Specification accepted by ISO/IEC 19505-1:2012, ISO/IEC 19505-2:2012

development process starting from requirement gathering to design and implementation. A lot of work has been performed in the direction of creating new language extensions which support aspect-oriented concepts as well as developing compilers for such languages. On the other hand, other research focuses on the support of aspect-oriented concepts in the design phase in a formal way supporting standards.

UML profile is an extension technique for customizing UML to a specific domain or language implementation. Metamodel is a parallel implementation to UML. Both UML profile and Meta model support aspect-oriented concepts. This section focuses classifying UML extension mechanism based on whether it is a profile or metamodel, the used tool is open source or commercial, the UML standard version, and whether it supports an aspect-oriented language or is language independent. An AspectJ profile supporting aspect-oriented concepts in Java using one of the commercial tools and UML 2.0 extension mechanism has been developed [23]. Enhancements of such research to support UML2.4 have also been conducted [30]. A metamodel supporting aspect-oriented concepts and which is independent of language implementation and platform has been proposed [25]. The research does not rely on aspect-oriented elements related to a specific language but relies only on the basic elements (Aspect, Pointcut, Advice, joinpoint). Another work proposes a tightly coupled AspectJ metamodel with Java created code based on Java metamodel [24]. It is simple, but not considered as an extension to UML as Java metamodel is a linear version of it. In [26], another metamodel has been proposed representing both behavioral and static structures. This model supports the class and interaction diagrams by creating a tool based on the basic elements of aspect-oriented software development.

A comparative study of the extension mechanisms proposed to support aspect-oriented modeling approach is found in [28] [29] up to studies performed in 2010. An updated study, including more recent research is shown in Table 1. Our comparison is based on six criteria; UML version, extension mechanism, diagram support, tool support, code generation and language support.

UML Version: represents the UML version supported by OMG.

Extension Mechanism: uses the UML extension which maps to UML Profile, or creates a parallel extension to UML which maps to Metamodel. [19]

Diagram Support: represents which UML diagrams to support; **behavioral diagrams** (Use case diagram, Activity diagram, State Machine diagram, and Interaction diagram) and **static diagrams** (Class diagram, Object diagram, Package diagram, Component diagram, Deployment diagram, and Profile diagram) [11].

Tool Support: indicates whether the created profile has been applied with a tool, is an open source tool or one of the commercial tools.

Code Generation: indicates whether the tool created supports code generation from the design.

Language Support: indicates the language keywords supported by the created profile or if it is language independent (language independent support neglects some details and takes the common context of different language supported by specific profile).

TABLE I. ASPECT-ORIENTED EXTENSIONS COMPARISON

Author & Year	UML Version.	Language Support.	Extension Mechanism	Diagram Support.	Tool Support.	Code generation.
J. Evermann, (2007) [23]	2.0	AspectJ	Light weight (UML Profile)	Static Diagrams (Class Diagram)	Magic Draw (Commercial tool)	Supported.
M. Chibani, (2013) [30]	2.4	AspectJ	Light weight (UML Profile)	Static Diagrams (Class Diagram)	Magic Draw (Commercial tool)	Supported.
Y. Han, (2006) [24]	2.0	AspectJ	Light weight (UML Metamodel)	Static Diagrams	Create new tool	Supported
Z. Sharafi, (2010) [25]	2.3	Language Independent	Light weight (UML Profile)	Static Diagrams	CASE Tool	Language Independent
Z. Qaisar, (2013) [26]	2.0	Language Independent	Heavy weight (UML Metamodel)	Static Diagrams + Behavioral Diagrams	CASE Tool	Language Independent
A. Ali, (2014) [27]	2.0	Language Independent	Light weight (UML Metamodel)	Static Diagrams (Class Diagram) + Behavioral Diagrams (Interaction Diagram)	Created new tool	Language Independent

Based on the comparisons of the latest studies on aspect-oriented extensions, most of the previous work doesn't support the latest OMG UML 2.5 standards. In addition, not all types of diagrams are supported. Most of the previous work also uses commercial tools or create their own tool.

This work focuses on supporting the latest UML 2.5 standards with the Eclipse open source tool.

IV. PROPOSED MODEL

To create a model, language syntax keywords to be represented need to be listed. Then, the relation between different elements is defined. Finally, the type of extension to be used is matched in UML Profile extension elements to be represented in a Modeling Tool as a profile. Mapping elements of AspectJ profile need full awareness of UML elements and the Metaclasses [16].

In the proposed model, a UML extension is created through a UML profile using EMF to support AspectJ language syntax on Eclipse tool as open source one. The main elements of aspect-oriented programming (aspect, pointcut, advice, and joinpoint) are mapped together as shown in Fig. 5. A detailed representation of AspectJ elements and their relation to elementary subtypes are represented in Fig. 7 as AspectJ profile.

A. Language Syntax Representation

PointCut: A pointcut can be considered to be a filter or predicate to a set of events (called joinpoints) that is accessible to an aspect during program execution. Pointcuts may be categorized based on the kind of joinpoint, scope or a context [32]. Detailed pointcut representation in the profile is shown in Fig 6. Pointcuts in this model are represented as a child of the UML *Property* Metaclass.

Structure of pointcut

<pointcut> ::= <access_type> <pointcut_name> ({ <parameters> }) : { designator [&& | ||] };

<access_type> ::= public | private [abstract]
<pointcut_name> ::= { <identifier> }
<parameters> ::= { <identifier> <type> }
<designator> ::= [!]Call | execution | target | args | cflow | cflowbelow | staticinitialization |within | if | adviceexecution |preinitialization
<identifier> ::= letter { letter | digit }
<type> ::= defined valid Java type [31]

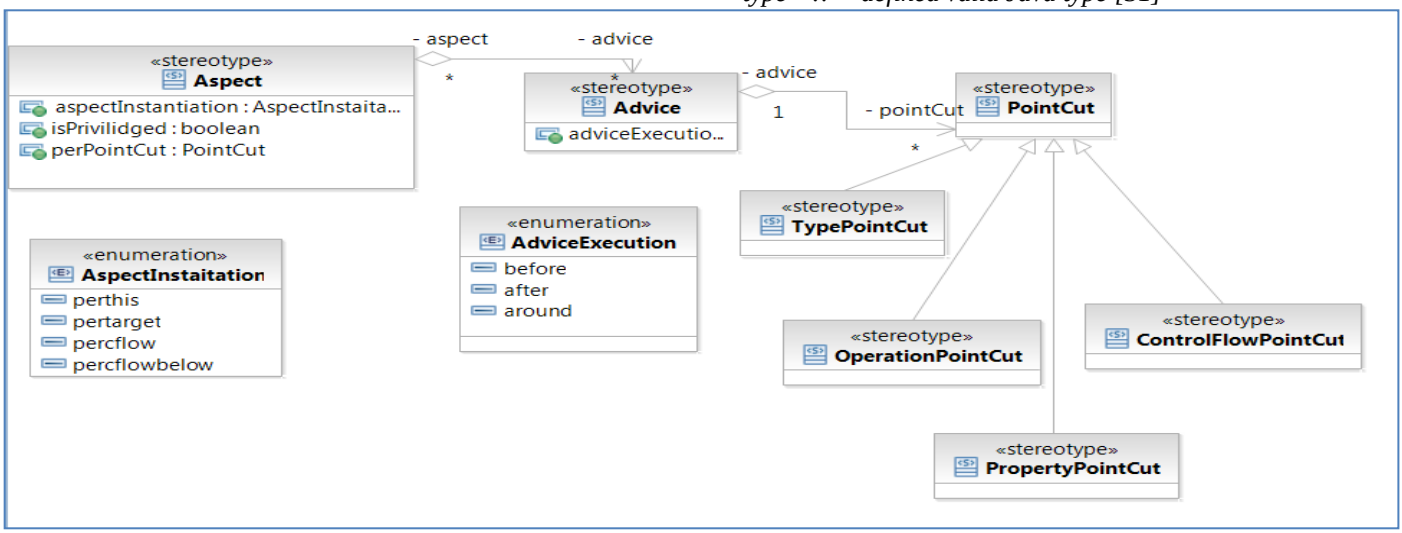

Fig. 5. Basic AspectJ syntax elements

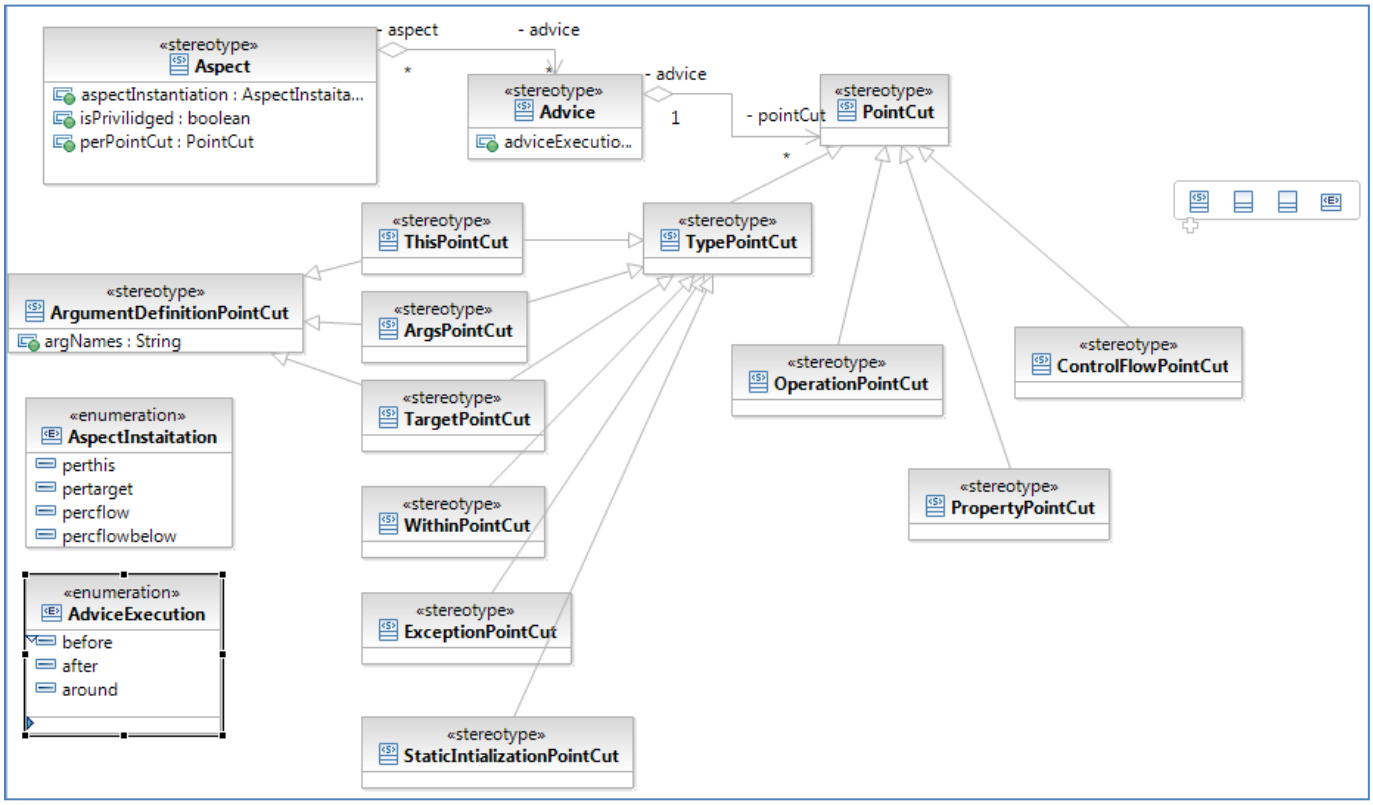

Fig. 6. Pointcut elements

Advice: An advice specifies what to do at a joinpoint matched by a specific pointcut. Each piece of advice is associated with a pointcut [32]. An advice in this model is represented as a child of the UML *Operation* MetaClass.

Structure of Advice

Advice ::= [ReturnType] TypeOfAdvice "("[Formals]")" [AfterQualifier] [throws TypeList] ":" Pointcut "{" [AdviceBody] "}"

TypeOfAdvice ::= before | after | around
ReturnType ::= TypeOrPrimitive ;(applies only to around advice)
AfterQualifier ::= ThrowsQualifier |
ReturningQualifier;(applies only to after--defined later)
Formals ::= ;(as a Java parameter list)
Pointcut ::= ;
AdviceBody ::= ;(as a Java method body with some difference in parameter passing as parameter values provided by pointcut) [31]

Aspect: An aspect represents a crosscutting concern in a modular way that supports encapsulation and abstraction [32]. An aspect in this model is represented as a child of the UML *Class* Metaclass.

Structure of Aspect

aspect ::= <access> [privilege] [static] aspect <identifier> <class identifier><instantiation>

<access> ::= public | private [abstract]
<identifier> ::= letter { letter | digit }
<class identifier> ::= [dominates] [extends]
<instantiation> ::= [issingleton | perthis | pertarget | percflow
| perflowbelow]
//pointcuts
//advice
//methods/attributes
}[31]

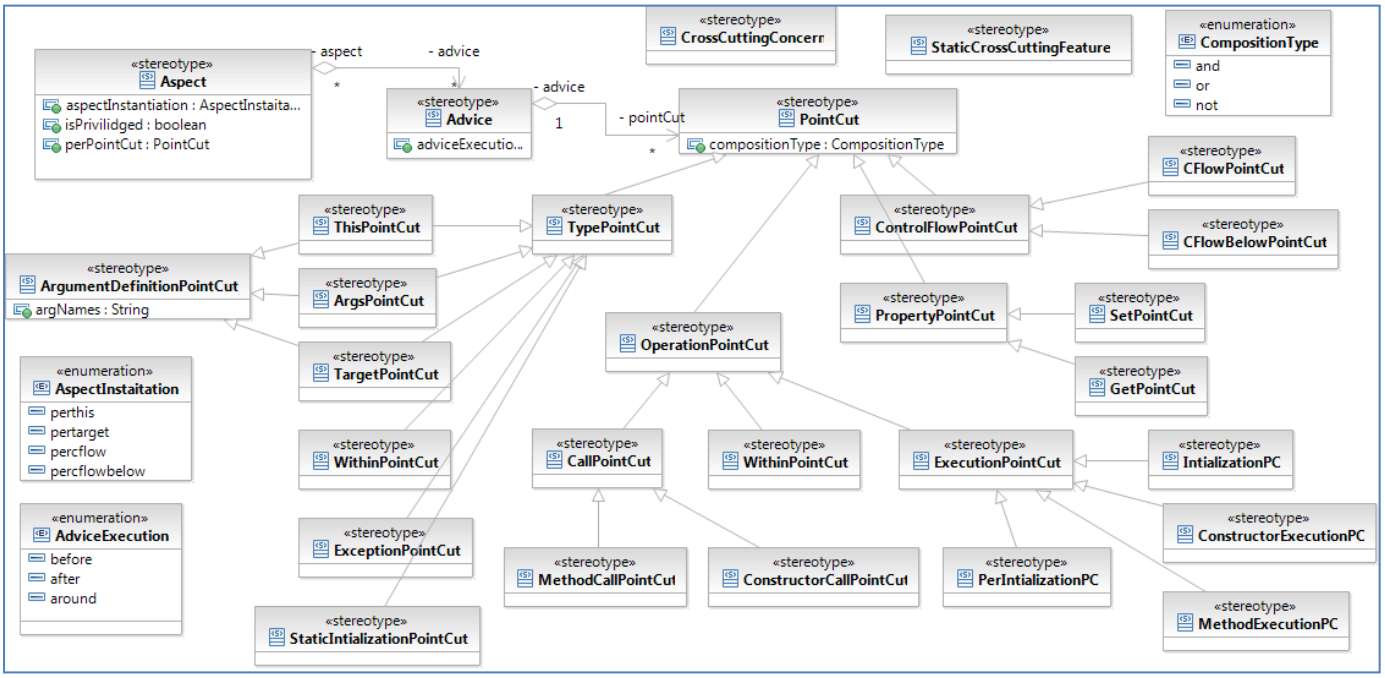

Fig. 7. AspectJ profile elements

B. AspectJ EMF UML Profile

Creating a profile using EMF requires the following steps [17][20][21] :

1) Create a UML profile: which is an extension mechanism supported by UML through standard extension elements: Stereotype, Object Constraint Language (OCL), and Tagged Value as shown in Fig. 8.

2) Validate the created profile: Validation is based on the version of UML that is applied. The proposed profile is created with UML 2.5 (latest version of UML by OMG) which has updated the OCL language validation for UML extension.

3) Generate an XMI profile (Fig 9): This is one of the features of using EMF. Once the model is created, it can be used in XML format. XMI is the XML representation to interchange the model between different tools supporting the OMG standards. As shown in Fig 9, the resulting AspectJ profile in XMI format holds the information of XMI version and the OMG specifications. It holds information of EMF Ecore, the UML Version as well as the detailed specification of the profile elements. The resulting XMI file may be used in Eclipse to create the modeling using the generative model of Eclipse. Moreover, the generated XMI file may be used with any tool supporting the XMI standard format.

Eclipse run-time environment is, then, used to run the profile.

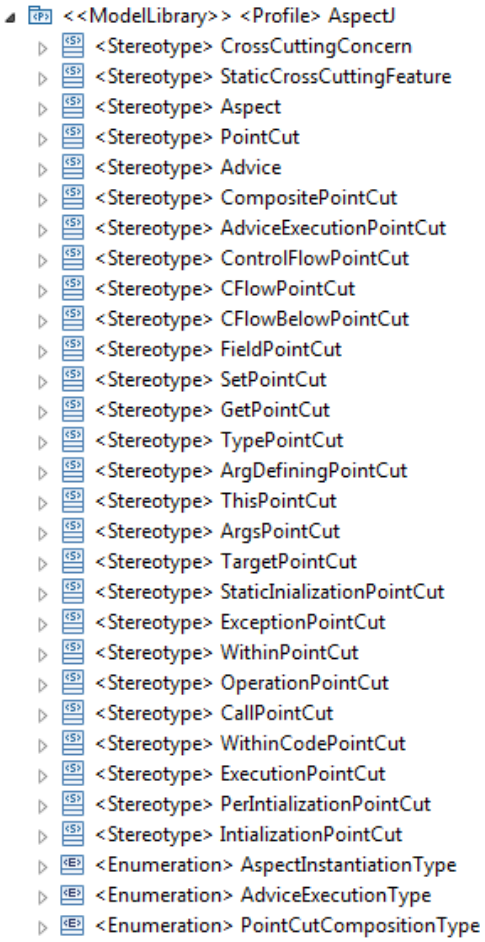

Fig. 8. AspectJ profile elements using EMF

```
<?xml version="1.0" encoding="UTF-8"?>
<xmi:XMI
 xmi:version="20131001"
 xmlns:xmi="http://www.omg.org/spec/XMI/20131001"
 xmlns:ecore="http://www.eclipse.org/emf/2002/Ecore"
 xmlns:standard="http://www.eclipse.org/uml2/5.0.0/UML/Profile/Standard"
 xmlns:uml="http://www.eclipse.org/uml2/5.0.0/UML">
  <uml:Profile xmi:id="_0" name="AspectJ">
    <eAnnotations xmi:id="_http2F2Fwww.eclipse.org2Fuml22F2.0.02FUML" source=
    "http://www.eclipse.org/uml2/2.0.0/UML">
      <contents xmi:type="ecore:EPackage" xmi:id=
      "_http2F2Fwww.eclipse.org2Fuml22F2.0.02FUML-AspectJ" name="AspectJ" nsURI=
      "http:///schemas/AspectJ/_LHY7QPPfEeS8v-ux5RacWQ/0" nsPrefix="AspectJ">
    </eAnnotations>
    <packagedElement xmi:type="uml:Stereotype" xmi:id="CrossCuttingConcern" name=
    "CrossCuttingConcern">
    <packagedElement xmi:type="uml:Stereotype" xmi:id="StaticCrossCuttingFeature" name=
    "StaticCrossCuttingFeature">
    <packagedElement xmi:type="uml:Stereotype" xmi:id="Aspect" name="Aspect">
    <packagedElement xmi:type="uml:Stereotype" xmi:id="PointCut" name="PointCut">
    <packagedElement xmi:type="uml:Stereotype" xmi:id="Advice" name="Advice">
      <ownedRule xmi:id="_MIH_cL-fEeagP8MnXSS-rg" name="adviceconstrain">
        <specification xmi:type="uml:StringExpression" xmi:id="_OEQ8AL-fEeagP8MnXSS-rg" name=
        "Advice" symbol="context Advice inv:allInstances.featuredClassifier.oclsIsKindOf(Aspect)"/>
      </ownedRule>
      <generalization xmi:id="Advice-_generalization.0">
        <general xmi:type="uml:Class" href="pathmap://UML_METAMODELS/UML.metamodel.uml#Operation"/>
      </generalization>
      <ownedAttribute xmi:id="Advice-adviceType" name="adviceType" type="AdviceExecutionType"/>
      <ownedAttribute xmi:id="Advice-pointCut" name="pointCut" type="PointCut" association=
```

Fig. 9. AspectJ profile in XMI format

C. Case Study

A Simple Telecom Simulation of a telephony system in which customers make and accept both local and long distance calls is presented here as an example of applying the proposed model [18]. The basic objects of the telecom model are shown in Fig. 10. The Customer class holds methods for managing calls. The *Connection* class models the physical details of establishing a connection between customers. The *Call* class is created for both caller and receiver. If the caller and receiver have the same area code then the call is established with a *Local* connection. Otherwise a LongDistance connection is required. Three Aspects are used in this example. The *Timing* aspect keeps track of total connection time for each Customer by starting and stopping a timer associated with each connection. The *TimerLog* aspect can be included in a build to get the timer to monitor when it started and stopped. The *Billing* aspect adds billing functionality to the telecom application on top of timing.

The created profile successfully mapped the code for the model representation of aspect, pointcuts, and advices as in Fig.11. This case study shows the ability of the created model to support the language representation of AspectJ.

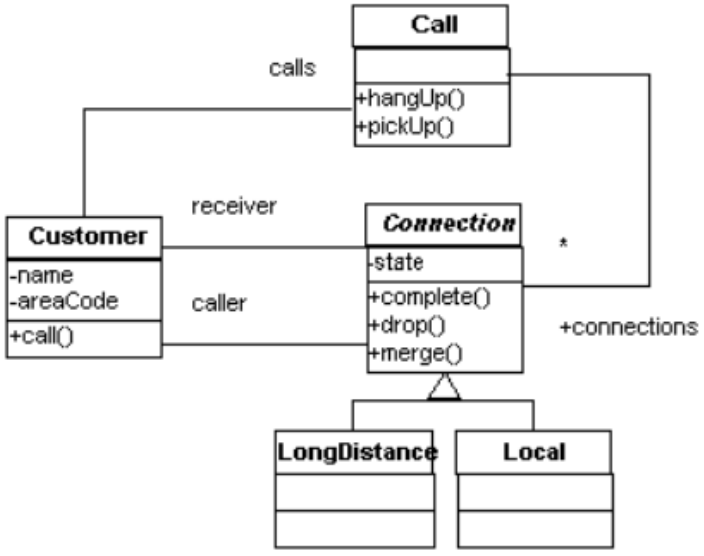

Fig. 10. Telecom Example Basic Objects[18]

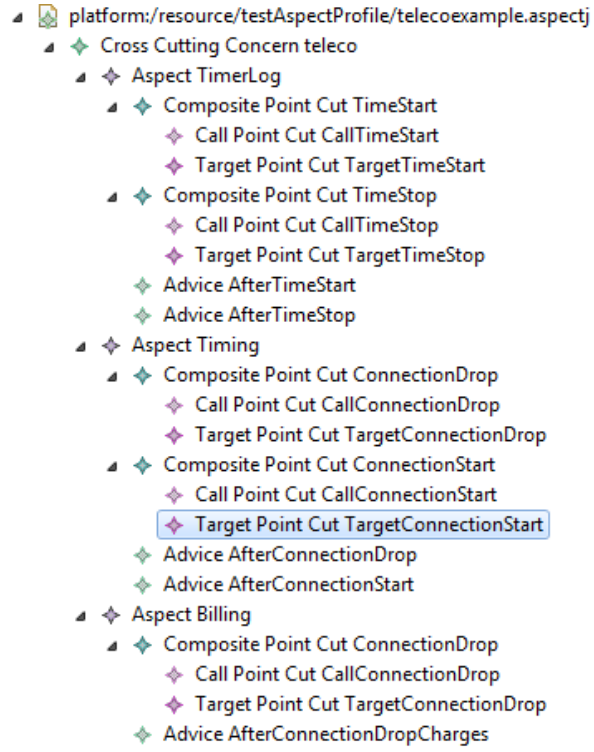

Fig. 11. Telecom Example representation using created AspectJ Profile

V. Conclusion and Future Work

A successful modeling representation of AspectJ language using Eclipse open source tool is created. Model-driven architecture concepts are applied following UML 2.5 standards; the latest version of UML up to the time of writing this paper. Most of the previous work doesn't support the latest OMG UML 2.5 standards. Moreover, most of the previous work uses commercial tools or create their own tool to extend UML to support aspect-oriented concepts. MDA concepts may be applied to both language and the domains customization in UML. This work uses MDA concepts to support language customization as a UML profile. Future research in this area may deal with supporting code generation from design as well as generating the class diagram from the code. More research can be done in handling the unification of specific domains such as healthcare, finance, telecom in a standard UML model using MDA concepts.

References

[1] Fillman, Elrad, Clark, Aksit (Eds.), Aspect-Oriented Software Development , Addison Wesley Professional, 2004.

[2] Sommerville,SoftwareEngineering,9th Edition, Pearson,chapter21, Aspect oriented engineering, 2011.

[3] Y Raghu Reddy, An aspect oriented approach to early software development, 7th ICUML, 2004.

[4] Mark Basch, Arturo Sanchez , Incorporating Aspects into the UML, Aspect Oriented Modeling Workshop at AOSD, 2003.

[5] Grigoreta S. Cojocar and Adriana M. Guran, A Comparison of Aspect Oriented Languages, Proceedings of the National Symposium ZAC2014, pages 11-18, 2014.

[6] The Home of AspectC++ (http://www.aspectc.org/) Retrieved 22-9-2014

[7] AspectJ (http://eclipse.org/aspectj/) Retrieved 22-9-2014

[8] Enas Ashraf, Getting Started with Model Driven Development and Domain Specific Modeling, Software Engineering Competence Center, 2013.

[9] OMG(http://www.omg.org/gettingstarted/gettingstartedindex.htm), Retrieved 22-9-2014

[10] Martin Fowler,UML Distilled: A Brief Guide to the Standard Object Modeling Language,3rd ed,Addison-Wesley Professional,2004.

[11] Object Management Group. UML 2.4 Infrastructure, OMG document ptc/10-11-16, 2011.

[12] A. Vodovnik, K. Žagar, MODEL DRIVEN ARCHITECTURE, CONTROL SYSTEMS AND ECLIPSE , ICALEPCS , 10th, 2005

[13] Johan Den Haan, MDA ,Model Driven Architecture, basic concepts(http://www.theenterprisearchitect.eu/blog/2008/01/16/mda-model-driven-architecture-basic-concepts/). Retrieved 22-9-2014

[14] Dave Steinberg,Frank Budinsky ,Marcelo Paternostro ,Ed Merks ,EMF: Eclipse Modeling Framework, 2 ed,Addison-Wesley Professional ,2008.

[15] Richard Paige,The Meta-Object Facility (MOF), University of York, UK, July 2006.

[16] Object Management Group. UML 2.5 Infrastructure, OMG document ptc/ formal-15-03-01, 2015.

[17] Ed Merks , James Sugrue,Essential EMF, DZone,2008

[18] The AspectJ Programming Guide, http://www.eclipse.org/aspectj/doc/next/progguide/printable.html#a-simple-telecom-simulation, Palo Alto Research Center, 2003.

[19] James Bruck , Kenn Hussey,Customizing UML: Which Technique is Right forYou?, http://www.eclipse.org /modeling/mdt/uml2/docs/articles/Customizing_UML2_Which_Technique_is_Right_For_You/article.html .2008

[20] Model Development Tools (MDT), UML2, http://www.eclipse.org/modeling/mdt/?project=uml2 , retrieved (07-02-2016).

[21] MDT/UML2 ,http://wiki.eclipse.org/MDT/UML2., retrieved (07-02-2016).

[22] D. W. Embley, B. Thalheim, Handbook of Conceptual Modeling: Theory, Practice, and Research Challenges. Springer, 2011.

[23] J. Evermann, "A Meta-Level Specification and Profile for AspectJ in UML," Journal of Object Technology, Volume 6, no. 7, pages 27-49, 2007.

[24] Y. Han, G. Kniesel and A. Cremers, "Towards Visual AspectJ by a Meta Model and Modeling," 6th International Workshop on Aspect-Oriented Modeling, Vancouver, 2006

[25] Z. Sharafi, P. Mirshams, A. Hamou-Lhadj, and C. Constantinides. Extending the UML Metamodel to Provide Support for Crosscutting Concerns. In Proceedings of the 34th ACIS International Conference on Software Engineering Research, Management and Applications (SERA'10), Montreal, Canada, pages 149–157.IEEE, 2010

[26] Z.Qaisar, N.Anwar, S.Rehman. Using UML Behavioral Model to Support Aspect Oriented Model. Journal of Software Engineering and Applications, pages 98-112, 2013

[27] A.Ali, Z.Malik, N.Riaz ,M.Jaffer,K.Usmani. The UML Meta Modeling extension mechanism by using Aspect Oriented Modeling (AOM). In Proceedings of the International Advance Computing Conference (IACC), pages 1373– 1378.IEEE, 2014

[28] A.Magableh,Z.Shukur, N.MohdAli. Heavy weight and lightweight UML Modeling Extension in aspect orientation in the early stages of software development. In Proceedings of the Journal of Applied Science, pages 2195 –2201.Asian Network for Scientific Information, 2012

[29] A.Magableh, Z.Shukur, N.MohdAli. Systematic Review on Aspect Oriented UML modeling: A Complete Aspectual UML modeling Framework. In Proceedings of the Journal of Applied Science, Asian Network for Scientific Information, 2013

[30] M.Chibani, B.Belattar, A.Bourouis .Towards a UML Meta Model Extension for Aspect Oriented Modeling. ICSEA, 2013

[31] J.D.Gradecki, NLesiecki, Mastering AspectJ Aspect-Oriented Programming in Java, Wiley, 2003.

[32] A.Colyer, A.Clement, G.Harley, M.Webster, "Eclipse AspectJ: Aspect-Oriented Programming with AspectJ and the Eclipse AspectJ Development Tools", Addison Wesley Professional, 2004.