International Journal of Advanced Computer Research (ISSN (print): 2249-7277 ISSN (online): 2277-7970)
Volume-3 Number-3 Issue-11 September-2013

Executable UML plug-in for Eclipse

A. M. Magar[1], N. J. Uke[2]

Abstract

MDA is a standard from OMG and is used by industry as an approach to application design and implementation. MDA is a way to manage and organize enterprise architectures supported by automated tools and services for defining the models. MDA facilitates transformations between different model types. In MDA PIM models are directly transformed and executed. These PIM and PSM models and meta-models are created using UML and MOF. UML in its current state is not directly executable. It lacks action semantics to describe the steps executed by the system in response to events. Existing executable UML lacks a standardized Action Semantics. Executable UML is an attempt to make UML directly translatable and executable to 3rd or 4th generation programming languages using eclipse plug-in. Using EMF and GEF it is possible to create feature reach graphical editor for UML. These graphical UML models are stored in XMI format as models. Our proposed Eclipse xUML plug-in is part of MDA which provides rich UML class diagram editor along with palette which contains all UML notations. Plug-in combines Java perspective and class diagram perspective. Plug-in allows developer to edit class diagrams as models and it also adds action semantics to the class diagram using Java. xUML plugin makes use of existing features of eclipse to execute the models as Java project.

Keywords

Executable UML, Eclipse Plug-in, AST

1. Introduction

MDA supports model driven engineering of software system. MDA is providing a set of guidelines for structuring specifications which are expressed as models. Using MDA system functionality may be first defined as PIM through appropriate DSL. PIM may be then translated to PSM, for actual implementation using general purpose languages like java. The translations between PIM to PSM may be performed using automated tools compliant to the QVT standard[9].

Aim of the MDA is to separate design from architecture. Even though QVT is a specific standard for model transformation, xUML provides translative approach for executing models. UML was extended by semantics for actions. Earlier UML was not executable; in newer version the action semantics provides at a high level of abstraction a complete set of actions. For example, actions are defined for manipulating collections of objects directly, thus avoiding the need for explicit programming of loops and iterators. Executable UML relies on these new actions to be complete.

In this paper we introduce Eclipse as a editor for UML class diagram and execute it as a model by using translative approach. Action semantics are added directly using AST to the java code. Graphical Class diagram Editor is created using GEF and EMF frameworks. Java can be used as a action language to define action semantics. Class diagrams created using this approach can be easily stored in to XMI format.

2. Existing Methodology

There are already some related studies on application of Executable UML. TextUML toolkit is a eclipse plug-in which allows to create UML models at the same speed we write code. Some approaches include and exists for executable UML models is based on using UML state machine to describe behavior of an operation [3]. The direct execution of such state machines would be still inefficient. Behavior in this case is given in terms of action languages. These actions are written in languages such as

- Action Language(OAL)
- Shlaer-Mellor Action Language (SMALL),
- Action Specification Language(ASL)
- That Action Language (TALL)
- Starr's Concise Relational Action Language(SCRALL)
- Platform-independent Action Language (PAL)

A.M.Magar, Department of Information Technology, Sinhgad College of Engineering, Pune-411041.

N. J. Uke, Department of Information Technology, Sinhgad College of Engineering, Pune-411041.

- PathMATE Action Language (PAL)

In addition to this JAction and OCL4X are used for defining actions to make UML models executable [5][6].

3. Our Methodology

3.1. Architecture of Eclipse Plug-in

Eclipse provides development tools for different languages like Java, C and C++. There are many different plug-ins supported by Eclipse. Eclipse plug-in architecture as shown in figure contains class diagram editor and Java editor in common perspective. These editors are dependent on EMF and GEF frameworks from eclipse modelling.

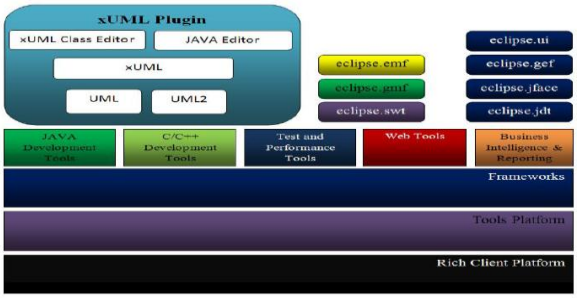

Figure 1: Plug-in architecture

xUML Plug-in uses the tools UML2 toolkit which provides different facilities and options for making models. Our methodology uses Eclipse Modeling Framework(EMF) for storing models in XMI format and Graphical Editing Framework(GEF) for feature rich graphical notations. xUML plug-in combines UML perspective and Java perspective which allows creating models graphically as well as generating code behind in Java perspective. xUML plug-in allows developer to work at very high level abstraction. Our methodology will enhance Eclipse as an MDA tool for which xUML plugin is needed for executing models.

3.2 UML Class Diagram Editor

EMF was started as a MOF of the OMG implementation. EMF is an enhancement of the MOF2.0. EMF is open source implementation that enhances the MOF 2.0 ECORE model and restructures its design in a way that is easy for the user[1]. The EMF is part of the Model Driven Architecture (MDA). It is the implementation of a part of the MDA in the Eclipse family tools. EMF can be used to describe a model. Java code can be generated by the addition of higher level Java code. Once the model has been completed, by means of EMF modeling or Java interface definition, we can generate the corresponding code to implement it.

The GEF allows us to easily develop graphical representations for existing XMI models. It is possible to develop feature rich graphical editors using GEF[1]. All graphical visualization is done via the Draw2D framework based on SWT. The editing possibilities of the GEF allow us to build graphical editors for nearly every XMI model. With these editors, it is possible to do simple modifications to your model, e.g. changing element properties or complex operations like changing the structure of your model at the same time. GEF assumes we have a model we would like to display and edit graphically. To do this, GEF provides viewers (of the type EditPartViewer) that can be used anywhere in the Eclipse workbench. Like JFace and GEF viewers are adapters on an SWT Control. GEF viewers are based on a MVC architecture. The controllers are to bridge the view and model[1].

EditParts are the central elements for the GEF applications[1]. They act as controllers that specify how model elements are mapped to visual figures and how these figures behave in different situations. Usually we will have to create an EditPart class for every model element class so we will have likely the same class hierarchy for the EditParts as we have for our model. EditParts are defined using the interface which can be referenced as org.eclipse.gef.EditPart [1]. Model is not manipulated directly when the user interacts with EditParts. Instead, a Command is created that will encapsulate the change. Commands are usually used to validate the user's interaction, and to provide undo and redo support. A GEF application is an editor for drawing diagrams. A diagram can be modeled as some UML notations. A shape might have properties for location, color, etc., and group structure of multiple shapes.

3.3 XMI Model

XML Metadata Interchange (XMI) is a standard that enables us to express our objects using Extensible Markup Language (XML) which is the universal format for representing data on the World Wide Web[2]. XMI is more than a set of serialization rules though. XMI is closely related to modeling standards from OMG, enabling us to employ modeling effectively in your XML efforts. In our project XMI 2.0 specifies how to create XML schemas from models. Following is the sample XMI model created using UML class diagram editor.

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<uml:Model xmi:version="2.1" xmlns:xmi="http://schema.omg.org/spec/XMI/2.1" xmlns:uml="http://www.eclipse.org/uml2/3.0.0/UML" xmi:id="_bbgqINRZEeKQp_kHw-tZdg" name="temp">
  <packagedElement xmi:type="uml:Package" xmi:id="_bbgqIdRZEeKQp_kHw-tZdg" name="pack"/>
  <packagedElement xmi:type="uml:Package" xmi:id="_bbgqItRZEeKQp_kHw-tZdg" name="package2">
    <packagedElement xmi:type="uml:Class" xmi:id="_bbgqI9RZEeKQp_kHw-tZdg" name="Class">
      <ownedOperation xmi:id="_bbgqJNRZEeKQp_kHw-tZdg" name="operation" visibility="public"/>
    </packagedElement>
  </packagedElement>
</uml:Model>
```

Figure 2: Sample XMI model

3.4 Abstract syntax tree

The AST can be used as detailed tree representation of the Java source code. The AST can be used to define API to modify, create, read and delete source code[10]. The Eclipse *JDT* provide APIs to access and manipulate Java source code. JDT allows to access the existing projects in the workspace, create new projects and modify and read existing projects. JDT also allows launching Java programs. JDT allows us to access Java source code via two different means. First is Java Model and second the Abstract Syntax Tree (AST) which is a Document Object Model similar to the XML DOM. Each Java project is internally represented as Java model in Eclipse. The Eclipse Java model is a light-weight representation of the Java project.

Our methodology is based on adding action semantics to the class diagram. Translation from class diagram model will be done simultaneously using AST Parser. As UML class diagram editor and Java editor are in same perspective it allows us to execute the project directly as Java Project. This will raise abstraction level as action semantics are written in Java itself. Methods can be added easily along with action semantics directly to the Java source code. Models created using UML class diagram can be executed by simultaneously translating code in Java Editor. Figure 3 shows sample code which adds a method with action semantics to the Java perspective.

```
try{
IWorkspace workspace = ResourcesPlugin.getWorkspace();
IWorkspaceRoot root = workspace.getRoot();
// Get all projects in the workspace
IProject project = root.getProject("anand");
 IJavaProject javaProject = JavaCore.create(project);
IType iType;
org.eclipse.jdt.core.ICompilationUnit iCompilationUnit;
 boolean processMeths=false;

iType = javaProject.findType("pack.Class");
IMethod[] meths=iType.getMethods();
        if(meths.length==0)
        {
        processMeths=true;

        }else if(meths.length>0)
        {
        for(int i=0;i<meths.length;i++)
        {

if((meths[i].getElementName().trim().equals(name.trim())))
        {
        processMeths=false;
        break;
        }
else
        processMeths=true;
        }
        }
if(processMeths==true)
{
iCompilationUnit = iType.getCompilationUnit();
Document document =
new Doc ument(iCompilationUnit.getSource());
ASTParser parser = ASTParser.newParser(AST.JLS3);
parser.setSource(document.get().toCharArray());
CompilationUnit cu = (CompilationUnit) parser.createAST(null);
AST ast = cu.getAST();
StringBuffer program=new StringBuffer(document.get());
int offset=0;
for(int i=0;i<program.length();i++)
{
if(program.charAt(i)=='{')
{
offset=i+2;
break;
}
}
program.insert(offset,"public void "+name+"()\n{}\n");
 document = new Document(program.toString());
ASTRewrite rewriter = ASTRewrite.create(ast);

TextEdit edits =
rewriter.rewriteAST(document,
iCompilationUnit.getJavaProject().getOptions(true));
```

```
edits.apply(document);
iCompilationUnit.getBuffer().setContents(document.
get());
iCompilationUnit.save(null, true);
}
```

Figure 3: AST for java source code

This sample AST adds a method to java source in eclipse. Action semantics can be added easily just by modifying AST. As mentioned earlier we can thus prepare class diagram as well as execute it using translative approach.

4. Conclusion

MDA is gaining more focus in many organizations. MDA has the benefits of modelling at various levels of abstraction. With MDA we first build object model, which differentiates from the traditional approach of server side development. We create Models after modelling completion, after development of software and systems is enabled. MDA can be achieved by using UML, DSL, or other modelling solutions. Eclipse Graphical Modelling Framework (GMF) used to develop graphical editors based on Eclipse Modelling Framework (EMF) and Graphical Editing Framework (GEF). It helps us for defining the domain model, their relationships and properties among them. xUML models help in creating PIM to PSM mapping. Our project helps creating such models for java project by generating the code and executing it. Even with the help of reverse engineering the models can be obtained from the code available.

References

[1] Eric Clayberg, Dan Rubel "Eclipse: Building Commercial Quality Plug-ins (2nd Edition)", Addison Wesley Professional, 2006, ISBN-10: 0-321-42672-X.

[2] Timothy J. Grose, Gary C. Doney, Stephen A. "Mastering XMI: Java Programming with XMI, XML, and UML", John Wiley & Sons; ISBN: 0471384291.

[3] Burden, H. ; Heldal, R. ; Siljamaki, T. "Executable and Translatable UML -- How Difficult Can it Be?", Software Engineering Conference (APSEC), 2011 18th Asia Pacific, Digital Object Identifier: 10.1109/APSEC.2011.37.

[4] Dos Santos, O.M. ; Woodcock, Jim ; Paige, R. "Using Model Transformation to Generate Graphical Counter-Examples for the Formal Analysis of xUML Models", Engineering of Complex Computer Systems (ICECCS), 2011, Digital Object Identifier: 10.1109/ICECCS.2011.1.

[5] Diggins, C. ; Hamou-Lhadj, A. JAction: "A High-Level Surface Syntax for UML Action Semantics Computational Intelligence for Modeling Control & Automation", 2008, Digital Object Identifier: 10.1109/CIMCA.2008.142.

[6] Ke Jiang ; Lei Zhang ; Miyake, S. "An Executable UML with OCL-based Action Semantics Language" Software Engineering Conference, 2007. APSEC 2007. 14th Asia-Pacific Digital Object Identifier: 10.1109/ASPEC.2007.21.

[7] Schattkowsky, T. ; Muller, W. "Transformation of UML state machines for direct execution Visual Languages and Human-Centric Computing", 2005, Digital Object Identifier: 10.1109/VLHCC.2005.64.

[8] Mellor, S.J., Balcer, M.J.: "Executable UML – A Foundation for Model-Driven Architecture" Addison-Wesley, 2002.

[9] OMG, "MDA," Accessed January 2013. [Online]. Available: http://www.omg.org/mda/.

[10] Fischer, G. ; Lusiardi, J. ; von Gudenberg, J.W. "Abstract Syntax Trees - and their Role in Model Driven Software Development", Software Engineering Advances, 2007. ICSEA 2007 Digital Object dentifier: 10.1109/ICSEA.2007.12.

[11] Enrico Biermann ; KarstenEhrig ; Claudia Ermel ; Jonas Hurrelmann; "Generation of Simulation Views for Domain-Specific Modeling Languages based on the Eclipse Modeling Framework" , 2009 IEEE/ACM, International conference on Automates Software Engineering. No: 978-0-7695-4061-0/10.

[12] A.M. Magar.; M.J. Chouhan; "Executable UML (xUML) and MDA", International Conference GIT-2010 "Green-IT & Open Source" Conference Proceedings. No: 978-93-80043-89-0/13.

[13] Feiler P.H.; de Niz D; Raistrick C; Lewis B.A.; "From PIMs to PSMs", Engineering Complex Computer Systems, 2007, 12th IEEE International Conference.

A. M. Magar
Place: Pune, DOB:7/3/1975, ME(IT) appeared, life time member of ISTE, working as assistant professor in sinhgad academy of engineering.

N. J. Uke
Place: Pune, ME(CSE), associate professor in sinhgad college of engineering.