Use of AST for Translating Executable UML Models to Java Code in Eclipse and Testing Strategy for UML Models

Anand Mohanrao Magar, Nilesh J. Uke

***Abstract*—Executable UML, coupled with Model-Driven Architecture is the foundation for a new software development methodology in which domain (business) experts would be involved in the crafting of high-level models and technologists would be concerned with translating the models to 3rd or 4th generation code. Editors like Eclipse provides modeling framework and code generation facility for building tools and other applications based on a structured data model. Eclipse Modeling Framework (EMF) is a framework to create meta-models with code-generation capabilities. Eclipse Modeling Framework is not an Model Driven Architecture framework but is the building block on top of which other tools may build. The Graphical Editing Framework (GEF) provides technology to create rich graphical editors and views for the Eclipse Workbench UI bundled with draw2d.**

Our work specifies translatable approach for executing UML models in Eclipse editor. The Abstract Syntax Tree (AST) maps plain Java source code in a tree form. This tree is more convenient and reliable to analyze and modify programmatically than text-based source. Eclipse plug-in Editor have Java Perspective and UML perspective merged together. Our plug-in contains UML editor which makes use of EMF and GEF based implementation. Class diagram which can be easily translated to Java code. These models are further modified to have action semantics and by modifying AST final Java source code which is directly executable in Eclipse Editor is generated. For testing xUML models, best strategy is to use model based testing.

***Index Terms*—Abstract Syntax Tree, xUML, Model Based Testing**

I. INTRODUCTION

Eclipse is an extensible platform for building IDEs. It provides a core of services for controlling a set of tools working together to support programming tasks. Tool builders contribute to the Eclipse platform by wrapping their tools in pluggable components, called Eclipse plug-ins, which conform to Eclipse's plug-in contract.

The basic mechanism of extensibility in Eclipse is that new plug-ins can add new processing elements to existing plug-ins[4]. Eclipse provides a set of core plug-ins to bootstrap this process.

EMF is the Eclipse Modeling Framework used by IBM's open source Eclipse project. EMF represents the next generation of meta-modeling and object repository technology that started with the Meta Object Facility standard of OMG, and was progressed within the Java Community Process to produce the Java Metadata Interface standard.

MOF and EMF are very similar conceptually. The key concept in both is the notion of classes with typed attributes and operations with parameters and exceptions, supporting reuse through multiple inheritances.

The Eclipse JDT provides application programming interface to access and manipulate Java source code. JDT allows us to access the existing projects in the workspace and creates new projects and modify and read existing projects. JDT also allows us to launch Java programs and gives access to Java source code via two different means. The Java Model and the AST are Document Object Model similar to the XML DOM[5].

Each Java project in Java Perspective is internally represented in Eclipse as a Java model. Eclipse Java model is a light-weight representation of the Java project. Each Java file is represented as a subclass of the ASTNode class. Each specific AST node provides specific information about the object it represents.

Our Eclipse plug-in contains UML class editor using EMF and GEF framework which is capable for generating executable models (xUML) and by integrating Java perspective we modified AST so that source code gets generated from models[2].

The two transformations which are important in MDA are transformations from PIMs to PSMs and transformations from PSMs to code [10]. Plug-in allows us to work on directly models which are transformed to code by EMF transformations and AST modifications. Similar work is done by Braune, A, Hennig, S, in their paper they discussed about how to program algorithmic aspects using xUML[11].

FUML (Foundational Subset for Executable UML Models) could be used to instantiate and sequentially simulate software process models on a single computer. However, FUML is insufficient to execute software process models to drive realistic projects with large and geographically spread teams[6].

dos Santos, O.M. ; Woodcock, Jim in their paper discussed about model translation issues[12] . In our work we used AST as discussed above for transforming model in code. This work is just for enhancing Eclipse capabilities. Programmers are able to add action semantics to model and then models are simultaneously transformed in to the Java project.

Manuscript Received December, 2013.

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

Prof. Nilesh J. Uke: Department of Information Technology, Sinhgad College of Engineering, Pune-411041, India.

II. ECLIPSE EDITOR UML PERSPECTIVE

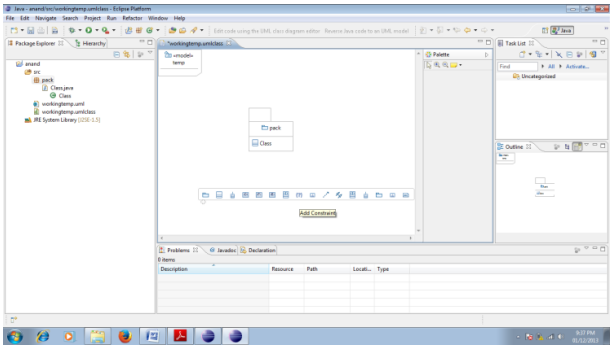

Figure 1 : Eclipse Editor UML perspective

Figure 1 shows our eclipse plug-in with UML perspective. Editor is capable to create precise class diagram models. Editor is capable for generating java source code using EMF. While editing class diagram editor is capable for adding action semantics using separate window. Changes made in the class diagram immediately reflected in java perspective. Action semantics which is added using separate window is modified directly by modifying AST in eclipse.

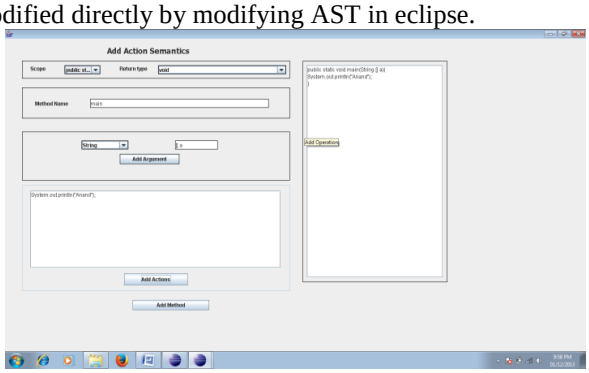

Figure 2 shows window for adding action semantics.

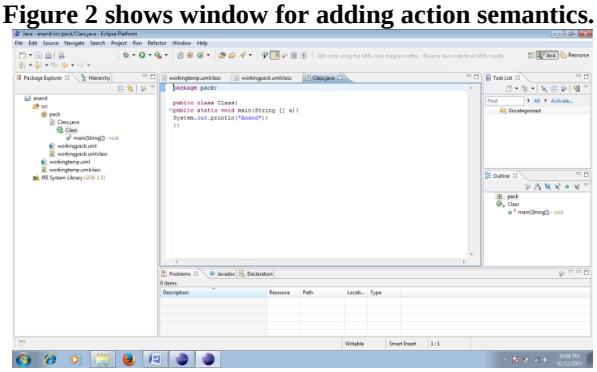

Figure 3 Modified AST with action semantics added to the code

Figure 3 shows modified complete source code after adding action semantics to the java source code. This code can be directly executed in eclipse environment.

III. AST FOR ECLIPSE

Our methodology is based on adding action semantics to the class diagram and translation from class diagram model is done simultaneously using AST Parser[7]. As both editors that is UML class diagram editor and Java editor are in same perspective it allows us to execute the project directly. 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.

Following figure shows sample source code for modifying AST programmatically.

```
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.t
rim())))
   {
     processMeths=false;
     break;
   }
   else
   processMeths=true;
   }
   }
   if(processMeths==true)
   {
iCompilationUnit = iType.getCompilationUnit();
Document document = new
Document(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,prog);
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);
}
} catch (Exception e) {
// TODO Auto-generated catch block
e.printStackTrace();
}
```

Figure 4 Sample AST code used in Eclipse Plug-in

IV. TESTING UML MODELS

Only few approaches exist that are used to test UML models. One of the approach for testing UML design models is transforming them into an executable form and executing tests which are generated under the consideration of defined test adequacy criteria. Another approach for testing UML class diagrams and OCL models is using snapshots. Whereas research work is going positively on model-based testing, defining and assessing test cases for UML models similar to unit testing of code is an open issue.

Testing executable UML model requires model based testing. Domain experts are supposed to prepare precise models so that it will result into executable system. Figure 5 shows evolution of model based testing.

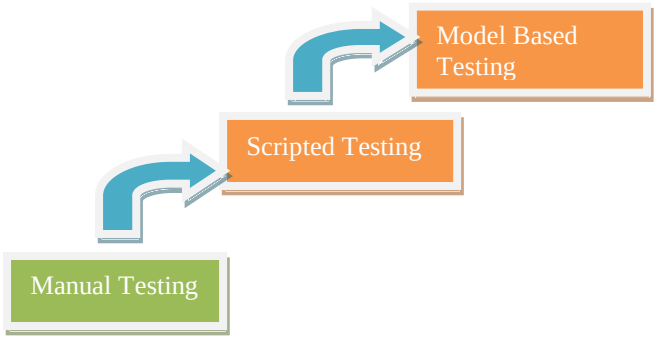

Figure 5 Evolution of Model Based Testing

Model-based testing(MBT) is software testing strategy in which test cases are generated in whole or in part from a model that describes some functional aspects of the system under test (SUT). There are two types of MBT i.e. online MBT and offline MBT[9].

Following figure shows phases involved in testing UML models. It contains Modeling & Test requirement selection, Test generation, Test concretization and Test execution phases. There are various known ways to deploy MBT, which include online testing, offline generation of executable tests, and offline generation of manually deployable tests.

Online testing means that a MBT tool connects directly to an system under testing and tests it dynamically.

Offline generation of executable tests means that a MBT tool generates test cases as computer-readable assets that can run automatically. Offline generation of manually deployable tests means that a MBT tool generates test cases as human-readable assets that can later assist in manual testing.

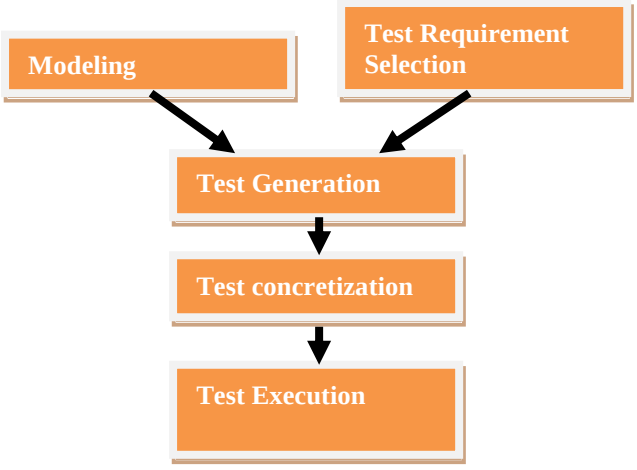

Figure 6 Phases in MBT

Purpose of modeling is to describe the system requirements for test generator. Many faults can be found in this phase only. In MBT we work at very high level abstraction. Purpose of Test requirement selection phase is to guide test generation.

V. CONCLUSION

Model Driven Architecture (MDA) is gaining more focus in many organizations. MDA stresses the benefits of modeling at various levels of abstraction and the integration and flow of information among models[1]. With MDA, first the object model is built, which differentiates it from the traditional approach of server side development. Models are built and after modeling completion, development of software and systems is enabled.

MDA can be achieved by using UML, DSL (Domain Specific Language), or other modeling solutions. DSL is one of the most efficient and proven ways of creating an MDA-based solution. There are many frameworks available for DSL like xText, EMF, GMF, Groovy, etc.

xUML models help in creating PIM to PSM mapping[3]. This project helps creating such models for java project by generating the code and executing it. With the help of Reverse Engineering, the model can be obtained from the code available. Eclipse can be used as a MDA tool and allows direct UML models to execute using translative approach .

REFERENCES

1. 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.
2. Mellor, S.J., Balcer, M.J.: “Executable UML – A Foundation for Model-Driven Architecture” Addison-Wesley, 2002.
3. A. M. Magar, N. J. Uke, “Executable UML plug-in for Eclipse”, International Conference on Emerging Trends, Technology and Research ICETTR-2013
4. Eric Clayberg, Dan Rubel “Eclipse: Building Commercial-Quality Plug-ins (2nd Edition)”, Addison Wesley Professional, 2006, ISBN-10: 0-321-42672-X
5. Timothy J. Grose, Gary C. Doney, Stephen A. “Mastering XMI: Java Programming with XMI, XML, and UML”, John Wiley & Sons; ISBN: 0471384291
6. Object Management Group, Semantics of a Foundational Subset for Executable UML Models, Object Management Group Std., Rev 1.1 [Online] Aug 2013. Available: http://www.omg.org/spec/FUML/1.1
7. Fischer, G. ; Lusiardi, J. ; von Gudenberg, J.W., “Abstract Syntax Trees - and their Role in Model Driven Software Development”, 2007. ICSEA 2007. International Conference on Software Engineering 2007
8. Burden, H. ; Heldal, R. ; Siljamaki, T. , ”Executable and Translatable UML -- How Difficult Can it Be?”, Software Egineering Conference (APSEC), 2011 18th Asia Pacific 2011
9. Fei Xie ; Levin, V. ; Browne, J.C., “Model checking for an executable subset of UML”, 2001. (ASE 2001). Proceedings. 16th Annual International Conference on Automated Software Engineering 2001
10. Wei Zuo ; Jinfu Feng ; Jiaqiang Zhang, “Model Transformation from xUML PIMs to AADL PSMs”, International Conference on Computing, Control and Industrial Engineering (CCIE), Control and Industrial Engineering Volume: 1 2010
11. Braune, A. ; Hennig, S. , “Using Executable UML to model algorithmic aspects of visualization systems “. 7th IEEE International Conference on Industrial Informatics, 2009
12. dos Santos, O.M. ; Woodcock, Jim ; Paige, R., “Using Model Transformation to Generate Graphical Counter-Examples for the Formal Analysis of xUML Models”, 16th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS), 2011

AUTHORS PROFILE

Mr. Anand Mohanrao Magar,
Received his BE degree in Computer Science and Engineering from Dr. B.A. M. U. University Aurangabad, in 1999. Currently he is pursuing ME IT form Pune Univercity. He is time member of ISTE, He is working as assistant professor in Sinhgad Academy of Engineering, Pune, India
anand7375@gmail.com

Mr. Nilesh J. Uke,
Received his BE degree in Computer Science & Engineering from Amravati University, Maharashtra, India, in 1995 and the ME in computer engineering in 2005 from Bharathi University Pune. He is Associate Professor with the department of Information Technology, Sinhgad College of Engineering, Pune University, Pune. Currently he is pursuing his PhD in Computer Vision from SRTM University. His research area includes Computer Vision, Human Computer Interface and Artificial Intelligence. He is member of IEEE, ACM, Life member of ISTE and Indian Science Congress.
njuke.scoe@sinhgad.edu