International Journal of Computer Information Systems and Industrial Management Applications.
ISSN 2150-7988 Volume 6 (2014) pp. 294 - 304
© MIR Labs, www.mirlabs.net/ijcisim/index.html

Translation of Behavioral Models to Java Code and Enhance With State Charts

Sunitha E.V[1], Philip Samuel[2]

[1] Department of Computer Science,
Cochin University of Science and Technology, India – 682022
sunithaev@gmail.com

[2] Information Technology, SOE,
Cochin University of Science and Technology, India – 682022.
philips@cusat.ac.in

***Abstract*: It is a wonderful idea to directly execute the system designs. In this paper we are introducing a method to convert the behavioral models to the implementation code. UML is used for modeling and Java is used as the target language. This paper describes how a system design depicted using activity, sequence and statemachine diagrams can be converted to its implementation code. Activity diagram helps to make the outline of the source program, and the sequence and statemachine diagrams contribute to the expansion of the source code. We are using an MDA approach where the system design is done in Platform Independent Model (PIM), then converted to Platform Specific Model (PSM) and finally to implementation code. One tool is implemented based on our method and it is evaluated against some other existing tools.**

Keywords: code generation, statemachine, activity diagram, sequence diagram**.**

I. Introduction

In the current business scenario, organizations want software systems that work properly, flawlessly, and user friendly. They are not at all bothered about which software development lifecycle is being used, which are the design models that are being used, which are documents being generated, or even the programming language being used for the software development. Organizations are concerned only about the cost, the time needed for the system delivery and the ability of the system to incorporate changes. Executable UML will be the best choice for the software developers to develop softwares with low cost and minimum time.

The executable UML provides a high level of abstraction for both specific programming language and the software structure. Executable UML models can be directly executed without generating any implementation code. It has three fundamental projections; data, control, and algorithm.

The first projection is data/object which has to be classified and structured. Normally UML Class diagrams are used for this purpose [1]. The second projection control means, the objects may have different behavior over time. This lifecycle is modeled using the state machine diagrams. The state machines have a set of procedures that causes the state changes of the objects. These procedures are sequence of actions including simple data manipulation, decision making, loop etc. These actions are modeled using action languages. Out of these three projections, we focus on data and control.

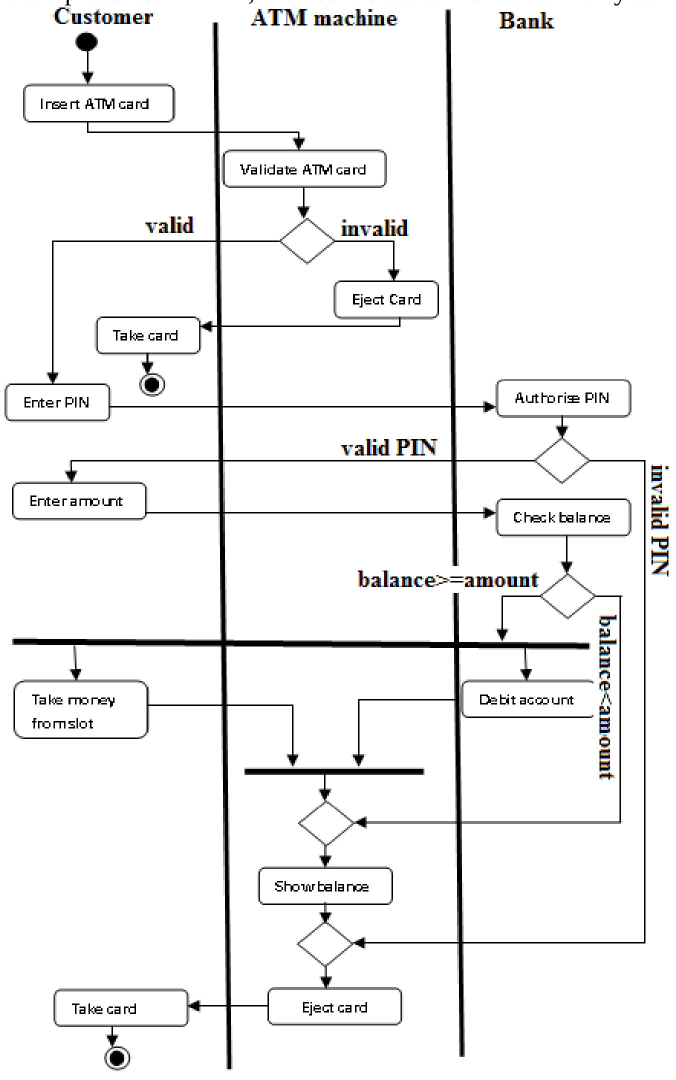
Nowadays executable UML is an attractive research area which discusses the possibilities to directly execute the UML models. Research is going on in this area to invent the methods to verify the UML models, to add more implementation specific details into the models, generation of implementation code from UML models, and directly execute the models without generating code.

In Software development, the coding step is very time consuming and error prone. In this step, we need to translate the system designs to programs. It needs less intelligence and more care to avoid errors. If we automate this step, it can save time and reduce errors.

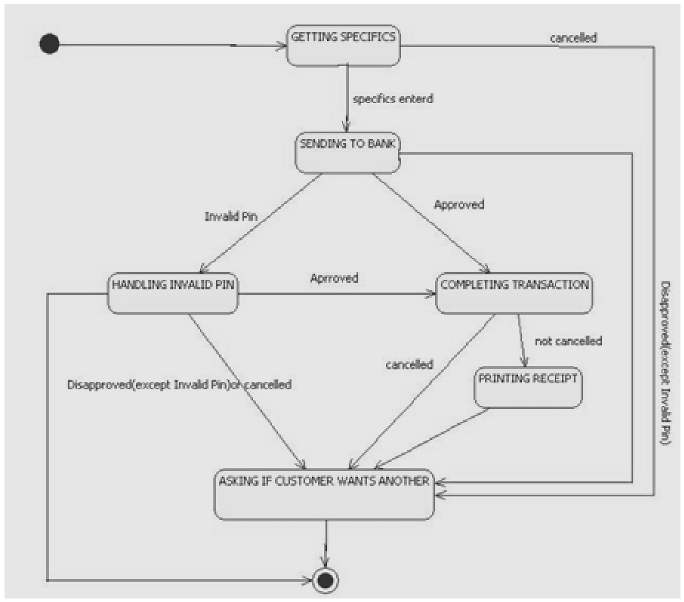
The system designs can be done in UML. UML provides structural and behavioral diagrams to design a system [2,3]. Structure diagrams show the static structure of the system and its parts on different abstraction and implementation levels and how they are related to each other. The elements in a structure diagram represent the meaningful concepts of a system, and may include abstract, real world and implementation concepts. Behavior diagrams show the dynamic behavior of the objects in a system, which can be described as a series of changes to the system over time.

UML forms a de facto standard for software system design. It is used for high level system design and additional model languages like Object Constraint Language (OCL) help us to introduce more details in the UML design [4]. Different tools are available for UML modeling.

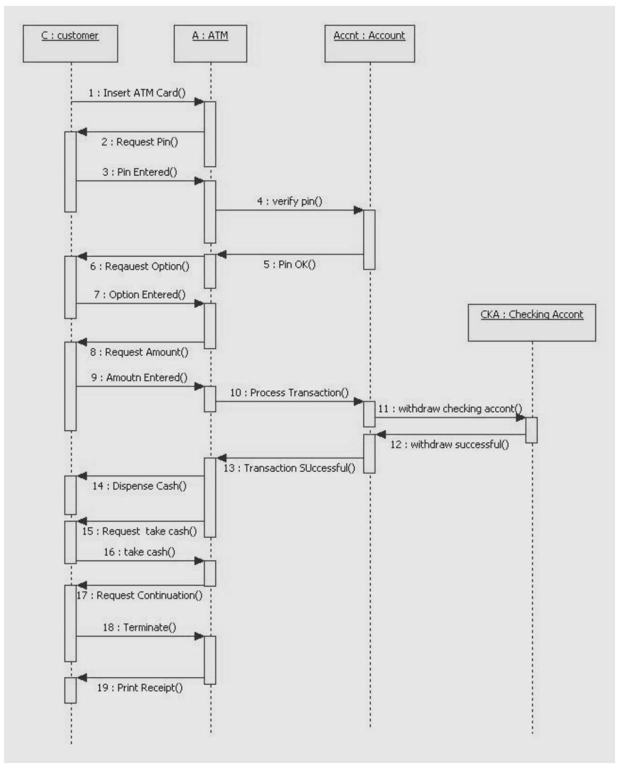
In this paper, we are discussing a method to generate Java code from the system design. As the first step of our research we are concentrating on the behavioral models which include state machines, sequence diagrams and activity diagrams. Since we are using an MDA approach [5], the system designs are represented in PIM, then converted to PSM and finally to



(a) Activity Diagram for ATM transaction



(b) Statemachine for PIN verification



(c) Sequence diagram for cash withdrawal

Figure.1 Behavioral Model for ATM machine

implementation code in target language; here it is java code. During code generation each activity diagram forms the outline of the coding, that is, classes are generated for each activity diagram and each method in the class will be expanded using the sequence diagram. State machine is used to fill up the remaining methods to show the full functionality of the system.

The code generation process is not that simple [6]. To the best of our knowledge, no existing CASE tool generates hundred percent complete source code from UML models, because using UML we can't represent all implementation details. The reason is that the syntax and semantics of UML are imprecise and informal. For example, in a sequence diagram we need not mention the object name, but during implementation the programmer gives suitable name for the anonymous object. When we automate code generation we have to find a way to fill these fields to get a complete code [7, 8]. One method is to allow the user to enter their own code in the generated code, second method allows user to include finer details using action languages like OCL.

In our method, XML is used as the intermediate language which helps us to export and import the models between different CASE tools. OCL is used to enhance the behavioral models. It helps us to include finer details such as actual parameters to the methods.

This paper is organized as follows. Section II gives idea about behavioral modeling with an example. Section III explains the transformation process. Section IV evaluates the method presented in this paper. Section V presents the works already done in this area and Section VI concludes the paper.

II. Behavioral Modeling

Behavior model capture different interactions and states within a model as it executes over time. It shows the control flow, data flow and state machine of a system. Behavior modeling includes diagrams like, activity diagram, state machine diagram, Interaction diagrams, and action language.

Activity diagram gives the sequence of activities, workflow and decision paths. It is useful to model processes involved in the business activities. So, UML activity diagram, especially UML 2.x activity diagram, is popular for Business Process modeling. It includes features like, activity, action, control flow, data flow, conditionals, concurrency (fork & join), nested behavior calls, partitions, and data stores.

State machines are used for describing the discrete behavior of an object in an event driven environment. Statemachine includes the features like, transition effects, state entry, do, exit, composite states, nested state machines, concurrency, transition guards, conditionals (choice, junction). The continuous behavior will not be modeled in a statemachine.

Interaction diagrams give the possible interactions between the objects in the problem domain. It can be viewed as a graph in which the nodes represent the objects and the links represent the communication links between them. There are two types of interaction diagrams in UML; sequence diagram and interaction diagram. Sequence diagram focuses on the time of communication and the interaction diagrams focuses on the structure of communication links.

Action Languages allow the designer to add code snippets in the model. They are used to refine the UML diagrams. Action languages can be used to specify the guard conditions and constraints in the model. It has syntax similar to the programming languages.

In our method we are considering behavior models for the code generation process. It includes state machine, sequence diagram and activity diagram. Activity diagrams show the entire business process flow. Each activity in the model is explained using the sequence diagrams. The state of each object in the activity diagram is explained using the state machine.

In activity diagram, we consider the basic features like activity nodes, decision making node, fork/join, initial node, and final node, for code generation. Swimlanes are also included in our method.

Figure.1 shows one sample situation. Figure 1 (a) shows activity diagram which represents the ATM transaction. It shows the PIN verification and cash withdrawal. Figure 1 (b) and (c) gives its corresponding state machine and sequence diagrams.

In our method, the activity diagram is used as the main tool for modeling the behavior of a business system. The sequence diagram and the statecharts are used for the refinement of the code generated from the activity diagram.

The activities shown in the main activity diagram can be more detailed in a secondary level of activity diagram or by using a sequence diagram. So, each activity, which has many sub activities, will be expanded using another activity diagram, also called as a child diagram. Instead of the child activity diagram, a sequence diagram can also be used depending upon the applicability and suitability.

The state chart diagram is used for the completeness of the code generated from the activity diagram and sequence diagram. The classes generated from these diagrams may not fully depict the system's behavior or functionality, or there may be chances of duplications, or absence of some features. These inconsistencies can be rectified using the statemachine diagram.

These behavioral models are then converted to XML files. Each diagram produces one XML document. The files are connected properly so that the activities, sequence of actions, and states of the objects can be traced properly.

III. Transformation Process

This section describes the steps in the transformation process. The code generation process has the following steps.

1. The system design is modeled using UML activity diagram, sequence diagram, and statechart diagram with the help of a modeler.
2. The model diagrams will be processed to generate consistent XML files
3. The XML files will be integrated to generate XMI document which is portable among CASE tools
4. Verify the consistency of the activity diagram and sequence diagram by comparing it with the state chart diagram.
5. Generating target code from the XMI document by transforming the activity & sequence diagrams.

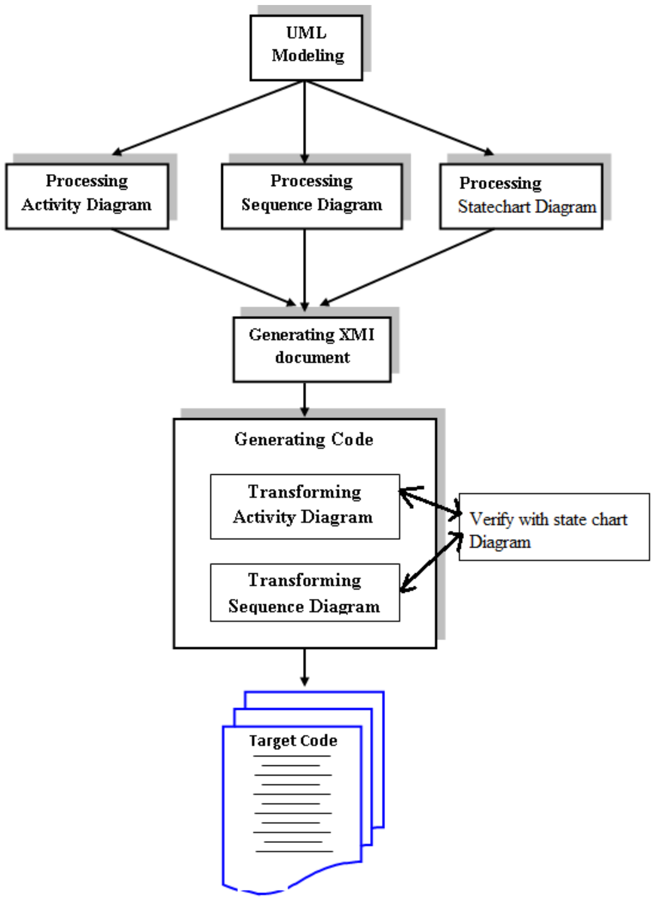

UML Modeling
Processing Activity Diagram
Processing Sequence Diagram
Processing Statechart Diagram
Generating XMI document
Generating Code
Transforming Activity Diagram
Transforming Sequence Diagram
Verify with state chart Diagram
Target Code

Figure. 2 Code generation Process

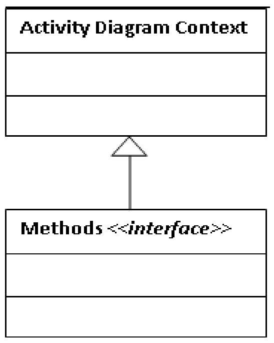


Figure.3 Activity Diagram meta model

These steps are shown in Figure 2. The processing of the system designs includes the following steps

a) Convert behavioral models to XML format
b) Parse the XML files to object representation
c) Do modifications in the object representation, if necessary
d) Convert modified object representations to XML files

During XMI generation, the XML files representing the system designs in the form of UML activity diagram and sequence diagram will be combined and converted to the XMI format. The statemachine diagram will be processed independent of activity and sequence diagrams. This is an optional step. This XMI representation gives the interoperability between CASE tools though the portability of system designs in the form of XMI documents.

The final step is the code generation step which identifies the behavioral models from the XMI document and transforms them to the corresponding Java code [9]. For this code generation, we specify conversion rules in XSLT (Extensible Style sheet Language Transformation) [10].

Each activity diagram will be converted to a class. Each node in the activity diagram makes a function call in the main () function. These functions' definitions will be made with the help of the sequence diagram attached with each activity in the activity diagram. Each message in the sequence diagram forms a method definition. All the internal messages will make further function calls and the sum of all that makes the definition of the outer message. The description of the code generated is described in the following example.

The conversion starts from the activity diagram. The context of each activity diagram is considered as a class. The code includes one context class and one interface. The activity diagram meta model is shown in Figure 3. Activity diagrams are identified from the XML files and two classes (one base class and an interface class) will be generated from each activity diagram. Each activity in the activity diagram is considered as a function call. The method declarations will be done in the interface class. The flow of activities shows the sequence of the method calls.

To implement the interfaces, the corresponding sequence diagrams will be traced out. XML files are tagged properly to make this tracing possible. The request in sequence diagram prompts a method invocation of the respective objects. For example, in Figure (c), the sequence of messages will be implemented as

```
A.InsertATMCard();
A.PinEntered();
A.OptionEntered();
```

Where 'A' is the object for which the methods are invoked. Nested requests are also processed in the same manner. These implementations are listed under the proper interfaces. The sequence diagram meta model is shown in Figure 4.

The name of the sequence diagram is used to build the method declaration in the interface class, which has to be matched with the methods in the interface class developed from the activity diagram. Or else, we can have the same interface class for activity as well as sequence diagrams. The real use of sequence diagram is to implement the interfaces declared by the activity diagram.

The state machines are taken to improve the coding done by activity diagram. It is converted to a hierarchy of classes starting with the class represents the context. Then it implements interface class. The interface is inherited by the state classes. There will be one class for each state. The state machine meta model is shown in Figure 5.

IV. Implementation

Based on the above mentioned code generation process we have implemented a prototype called UmlCode. UmlCode is a tool that supports software development process. The software system can be modeled using UML. The implementation code will be obtained by a button click.

It increases the work of system architects, but reduces the coding effort. The code generator UmlCode is implemented in Java. The UmlCode architecture is shown in Fig. 3.

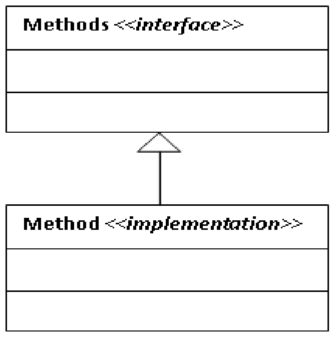


Figure.4 Sequence Diagram meta model

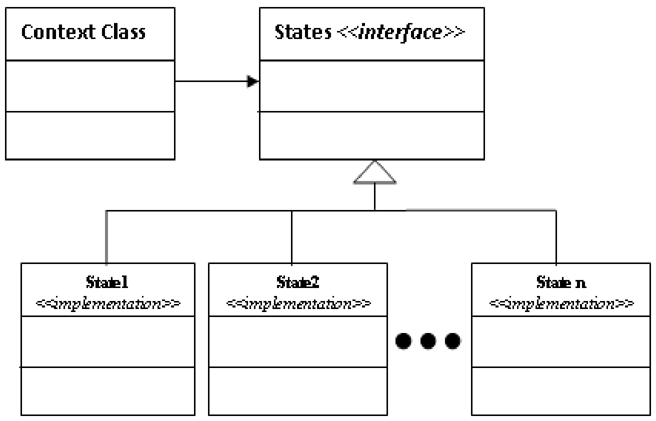

Context Class
States <<interface>>
State1 <<implementation>>
State2 <<implementation>>
State n <<implementation>>

Figure.5 State machine meta model

It has mainly four parts.

i. UML Modeler
ii. Model Processor
iii. XMI generator
iv. Code Generator

UML Modeler provides a canvas to model the system designs using the UML activity diagram and sequence diagram. Currently it supports basic elements like action node, initial and final nodes, fork, join and decision node for activity diagram, and object life line, function calls, return etc for sequence diagram. The modeler helps us to save the model as a JPEG image. The modeler creates the XML representation of the model. Here, we represent the graphical data in text format. This will be useful when we need to retrieve information from the model. For the XML representation we define a new document type definition (DTD) [11], which describes how UML 2.0 activity diagram should be expressed in XML.

The Model Processor processes, the sequence diagram, activity diagram and the OCL commands. It checks the consistency of activity and sequence diagrams. Also checks the OCL commands for syntax correctness. During system design each activity can be expanded using sequence diagram.

The finer details included in the sequence diagram will be helpful during the code generation of activity diagrams. So, the synchronization between activities and sequence diagrams should be done properly. This is accomplished by the Model Processor.

The XMI generator converts the XML files of activity and sequence diagrams to a single XMI file. This XML Metadata Interchange format improves the interoperability between CASE tools. Each sequence diagram is referenced properly at each activity. This referencing is done using the diagram id, which is an attribute in the XMI tag.

Code generator is the main part of UmlCode. It converts the models represented in XMI to java code. The transformation rules for converting XMI to java is written in XSLT. Different XSLT processors are available to process the XSLT rules. We use built-in processor in java. The class diagram of the code generator, UmlCode, is shown in the Figure 6.

UmlCode gives seven major features. It helps us to draw the UML models, save the model as JPEG and XMI, display the object tree representation of XMI, import XMI file to the Modeler, generate java code, and edit the model.

The implementation includes twelve main classes. UmlModeler is the main class which uses different classes, like MyModel, CustomCell, SAXTreeBuilder, to include the model drawing features, editing features, tree display features etc.

A. *Code generation*

UmlCode provides an editor which generates the XML file corresponding to each diagram (activity and sequence diagrams). The XML document will be saved in a standard directory in the name of the corresponding activity and sequence diagrams. The parsed XML document can be viewed as a tree structure. It helps us to check the system generated XML document and can check for errors. Fig.7 gives the screen shot of the UmlCode editor. It provides basic edit options like copy, paste, zooming and grouping options.

UmlCode provides option to expand one action. One action, or more precisely it can be called as an activity, can be expanded with a seqeuence diagram. This will help us to include more detailed information in the diagram. In UmlCode, we generate one class for each activity diagram. Each action is considered as a member method. In Fig.7, we create a class named ShipOrder, which has one member method getOrder(). Sometimes the user will enter the name as Get Order. The Model Processor will modify this name, so that it will remove the white space and make the first letter lowercase, and add simple brackets at the end if it is not there already. The model processor will do similar modifications to the class names too.

The XML representation of the activity diagram includes data to identify child sequence diagrams, i.e, expanded diagram. We add an attribute called ‘parent’ to the element ActivityGraph. This attribute will be set to true if it is a child diagram. Similarly, when an action is expanded, an attribute named ‘expanded’ will be added to the action node. It will be set to true if the node is expanded. These data will be useful when we do code generation of the activity node.

UmlCode provides an option to generate code from the activity and sequence diagrams drawn on the editor. The code generator takes the XML document from the standard directory and it will save the code in the same directory with the name ‘NameOfActivityGraph.java’. We have developed a rule set in XSLT to convert the activity diagram represented in XML to the java code. In the prototype we give skeleton of the actions. It needs to be elaborated with the help of sequence diagram. It considers the fork & join and create threads accordingly.

The code generation rules are implemented in XSLT. It specifies the transformation rules for the XML document. Rules are written in XSL and XPath [12]. In the prototype we are using built-in java XSLT processor. It is using TransformerFactory and Transformer for the conversion from XML to text document. This takes an XML document as input and returns a text document. In our project, we use XSLT processor to generate java code from the XML file. This java file will be stored in the standard output directory.

The following code segment shows how to use the built in java XSLT processor. The TransformerFactory() takes the XML file as input.

```
StreamSource xslSource = new StreamSource(xslFile);
TransformerFactory factory =
TransformerFactory.newInstance();
Transformer transformer =
factory.newTransformer(xslSource);
transformer.transform( new StreamSource(xmlFile), new
StreamResult(outFile));
```

XSLT is a template based language. It specifies when a node is occurred in the document which template should be done. We have implemented a XML to Java converter using XSLT.

It takes each node in the XMI document which has name ‘ActivityGraph’. First letter of its name should be capital letter.

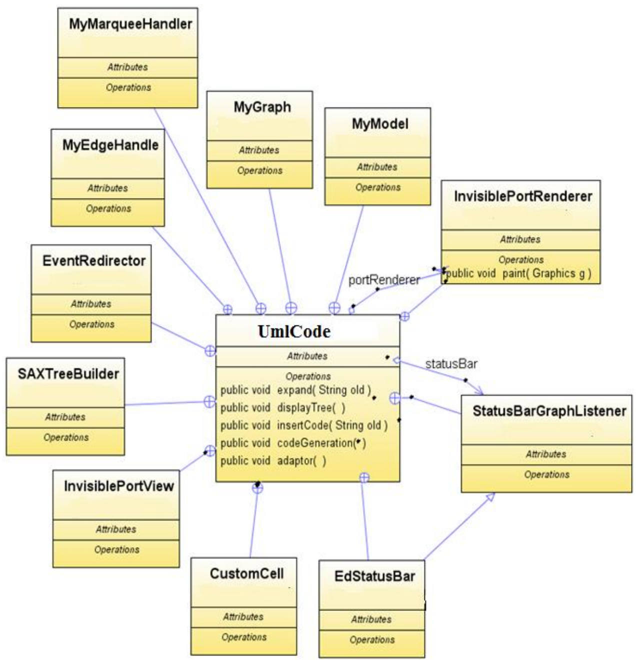


Figure 6. The class diagram of UmlCode

The fork and join in UML is implemented as Thread in java. We count the number of fork and generate that many threads. These threads will be invoked subsequently. The activity diagram may include sequence diagram as child diagram.

Each action node is implemented as a method in the implementation code. The nodes can be distinguished using the xmi:type attribute. We choose the nodes with type= uml:CallBehaviorAction. Initial and final nodes have type uml:InitialNode and uml:FinalNode respectively. The method body of each action is got from the child sequence diagram which is again a sequence of function calls.

The whole control sequence will be depicted in the main() method. Whenever a fork is encountered, it creates and starts threads. The number of threads created is equal to the number of control paths starting from the fork. These paths show parallel execution paths in the program. The nodes in these paths will be shown in the run() method of the MyThread class, which is a child class of Thread class.

When a decision node is encountered each decision path should be continued till it reaches a merge node. Next decision path will be considered only after closing the previous path, i.e, after meeting the merge node in the path.

In the application class the method body is implemented with the help of data extracted from the sequence diagrams. The UML:Operation template contains codes for extracting the name, return type and parameters of the method. The following code represents the general template for writing a method.

```
    <xsl:template match="UML:Operation">
    <xsl:call-template name="Visibility"/>
        <xsl:call-template name="Abstract"/>
        <xsl:call-template name="ReturnType"/>
        <xsl:value-of select="@name"/>(<xsl:call-template
    name="Parameters"/>){
            <xsl:call-template name="methodBody"/>
        }
</xsl:template>
```

The body of the method is created with the help of message sequences in the sequence diagram. Consider message sequence as shown in Fig. 8.

Here the occurrence of message switchOn() will invoke the next message getInitialCash() of class OperatorPanel which will send the initialCash in return. In this case, while creating the switchOn() method the code generator will search for the next message in that particular sequence diagram. Then it will find the message getInitialCash() and will add the code that invokes the getInitialCash() method with the object of the class OperatorPanel. Thus the swithOn() method will look like:

```
    swithOn(){   operatorPanel.getInitialCash();  }
```

and the getInitialCash() will look like:

```
    getInitialCash(){return initialCash;}
```

The next message can be of any one of the following type:

uninterpreted message, asynchronous message, return , send , call, create

All of these types of messages must be handled in different way. Another possibility for the next message is the presence of a combined fragment. It can be a loop, alternate or break fragment. The combined fragment details can also be translated to java code.

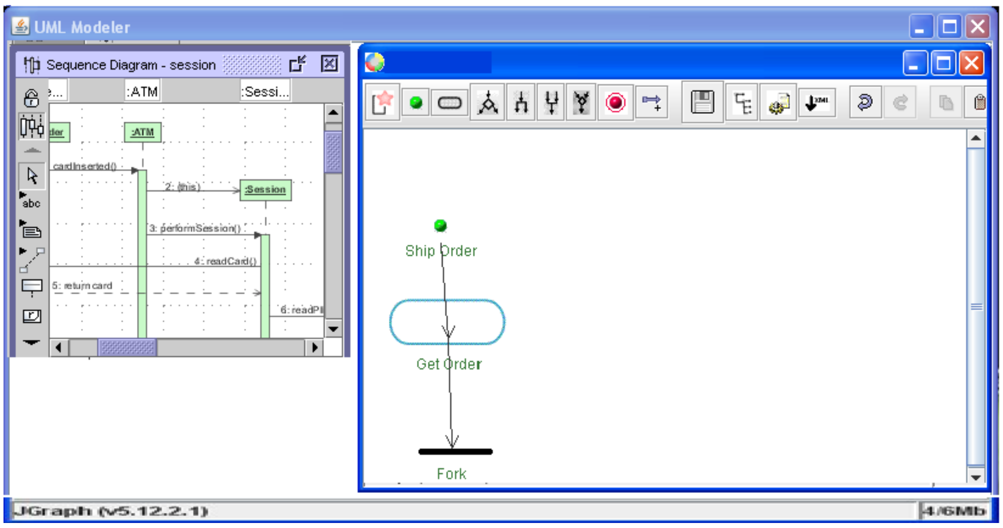


Figure. 7 UmlCode Editor Window

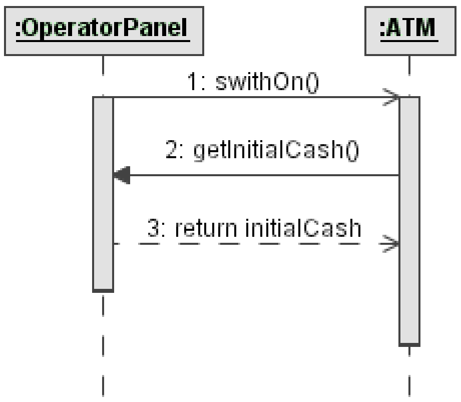


Figure. 8 A message sequence

V. Evaluation

The method described in this paper is implemented and tested against other similar tools like Rhapsody and OCode. Our implementation, UmlCode, has four major modules; UML modeler, Model Processor, XMI generator and code generator.

UML Modeler provides a canvas to model the system designs using the UML activity diagram, state machine, and sequence diagram. Currently it supports basic elements like action node, initial and final nodes, fork, join and decision node for activity diagram, and object life line, function calls, return etc for sequence diagram, and states in state machine. The modeler helps us to save the model as a JPEG image. The modeler creates the XML representation of the model. Here, we represent the graphical data in text format. This will be useful when we need to retrieve information from the model. For the XML representation we define a new Document Type Definition (DTD) [11], which describes how UML 2.0 activity diagram should be expressed in XML.

The Model Processor processes, the state machine, sequence diagram, activity diagram and the OCL commands. It checks the consistency of activity and sequence diagrams. Also checks the OCL commands for syntax correctness. During system design each activity can be expanded using sequence diagram and state machine. The finer details included in the sequence diagram and state machine will be helpful during the code generation of activity diagrams. So, the synchronization between activities, state machine and sequence diagrams should be done properly. This is accomplished by the Model Processor.

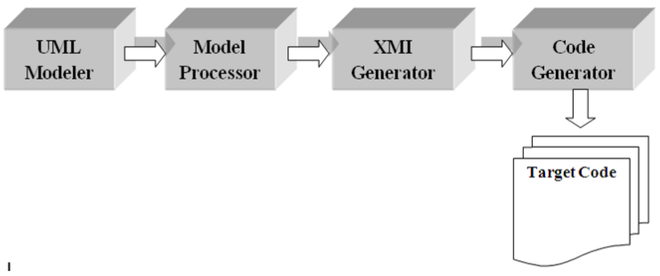


Figure.9 UmlCode Architecture

The XMI generator converts the XML files of the models to a single XMI file. This XML Metadata Interchange format improves the interoperability between CASE tools. Each sequence diagram is referenced properly at each activity. This referencing is done using the diagram id, which is an attribute in the XMI tag.

Code generator is the main part of UmlCode. It converts the models represented in XMI to java code. The transformation rules for converting XMI to java is written in XSLT. Different XSLT processors are available to process the XSLT rules. We use built-in processor in java.

A. *Business process type Vs percentage of code generated by UmlCode*

We have worked with different examples which include action nodes alone, action nodes with decision making, concurrency, decision making & concurrency, action node expansion, code elaboration etc. Predicting the number of lines of code in the completed source code will not be accurate. It depends on the logic we use (that differ from person to person), the complexity of the task to be done etc. We tried to map number of action nodes against the percentage of code generated. One action node can be implemented with single line of code or many lines of code, which depends on the logic we use and the task we need to do, and also the level of abstraction. One feasible method to map the percentage of lines of code generated is to map it against different types of the business process. We identified six different categories of programs; simple I/O, simple Decision making, concurrency, decision making & concurrency, Code elaboration, Node expansion. The percentage of code completion is plotted in Fig. 10.

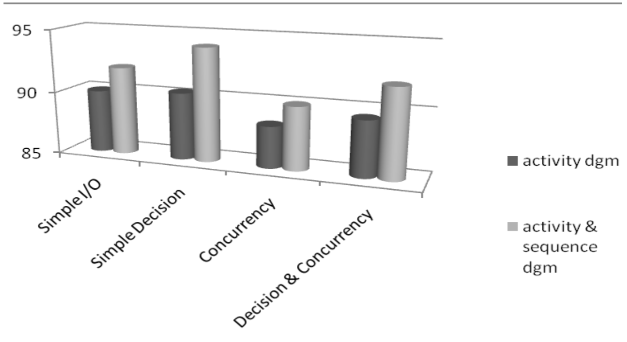


Figure 10 Type of business process Vs percentage of code generated by UmlCode

Simple I/O type business process has only action nodes, no control nodes such as, decision node, merge node, fork & join. UmlCode generates 90% complete source code for this type of processes. In simple decision making type process, we have action nodes and decision making nodes. For such processes, UmlCode generates 90.32% source code. Similarly for processes which include concurrency have 88.3% code coverage. General thing that we can deduce from Fig. 10 is that UmlCode provides more than 85% code coverage for all type of processes.

B. Complexity Vs Percentage of Generated Code

Two important parameters we need to compare are cyclometric complexity and code coverage. We have taken different business processes which have complexities 1, 2, 3, and 4. These process includes concurrency, decision making etc. A process which has complexity 1 has 90% code coverage. A process with complexity 3 has 90.5% code generation. An interesting factor we can deduce from Fig. 11 is that as the complexity increases, the code coverage increases. UmlCode gives more code coverage for more complex business processes. This implies that, our method can be used for automatic code generation of complex as well as simple business processes.

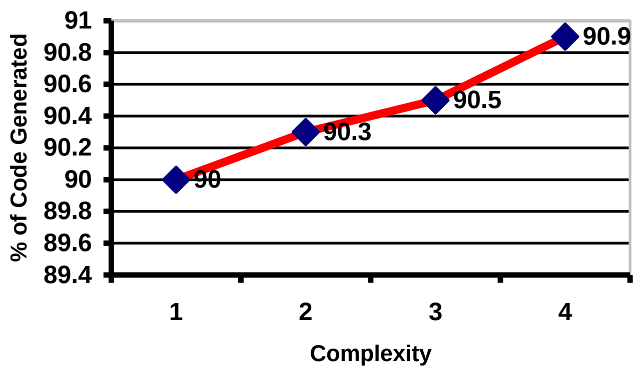


Figure.11 Complexity Vs Percentage of Code Generated by UmlCode

TABLE 1.
EFFICIENCY OF UMLCODE COMPARED WITH RHAPSODY & OCODE

	Ocode (ms)	Rhapsody (ms)	UmlCode (ms)	Efficiency over Ocode	Efficiency over Rhapsody
Total time for events without transitions	4.4	5.05	4		
Average time per event without transition	0.00248	0.00284	0.00225	9.27	20.77
total time for events having transitions	22.05	23.1	10.1		
average time per event having transition	0.00992	0.01039	0.0045	54.64	56.69
total time for all events	26.4	28.15	14.1		
average time per event	0.0066	0.00704	0.00353	46.52	49.86

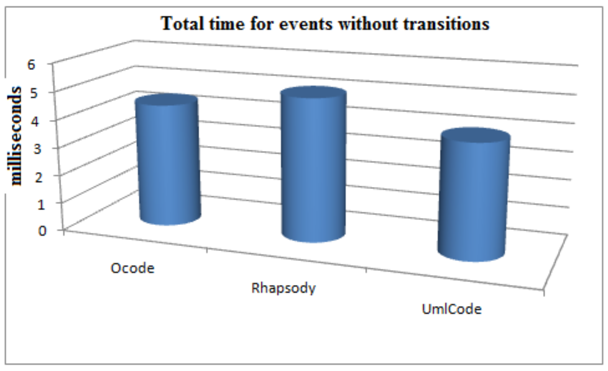


Figure 12 Total time for events without transition

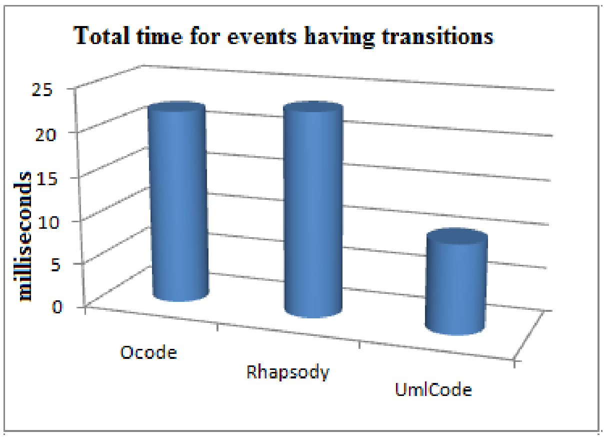


Figure.13 Total time for events having transition

100% complete java implementation code will be generated with the help of code elaboration. Code elaboration allows the user to include his/her own code in the UML model which will be used later during automatic code generation. Code elaboration is optional. Without code elaboration, UmlCode generates not less than 80% complete source code.

C. *Comparison with Rhapsody and OCode*

UmlCode is compared with similar tools like Rhapsody and OCode. The number of lines generated by each tool, the number of bytes generated and total number of classes generated is compared. Table 1 shows the comparison of UmlCode, Rhapsody and OCode. We considered the events with and without transitions. Total time taken for each type is calculated in milliseconds. Total number of requests for events without transition is 1778 and for events with transition is 2222. The efficiency of our tool (UmlCode) over other tools is shown in the table 1. Figure 12 and Figure 13 compare the total time taken for events without and with transition respectively.

VI. Related Works

Q Long[13] illustrate a algorithm to convert sequence diagram and class diagram to a target language, rCOS (Relational Calculus of Object Systems). It will first check the consistency of the class diagram and sequence diagram. it generates an error report if the diagrams are not consistent, otherwise the diagrams will be given for code generation.

Harrison [14] depicts a mapping method that converts the abstract system models to a high-level skeletal implementation code. UML is used as the design language and Java as the implementation (target) language. Harrison [14] considers class diagram for code generation. Classes marked with the stereotype <<entity>> will be mapped into an interface and a pair of implementing classes. One class will be abstract class and the other one will be instantiable. They also present the problems of generating object oriented language implementation code from high-level designs. They summarize the issues for Java implementation, like how to handle multiple inheritance etc.

A method to convert the class diagram represented in XMI format to Java code is presented in Bjoraa [15]. They have developed a prototype to output one Java file per class specified in the class diagram. The class diagram drawn in UML will be converted to XMI format. The XMI file will be parsed using XML parser and extracts the details, like class name, attributes and methods. Using this information the skeleton of the class definition will be produced in Java.

An approach to the model driven generation of programs in the Business Process Execution Language for Web Services (BPEL4WS) which transforms a platform independent model to platform specific model is described in Kochler [16]. Business process modeling is done using the activity diagrams. They define rules for integrating business process. This rule helps them to reduce complex activity diagrams to comparatively simple diagrams which do not contain loops. According to their approach the control flow models will be analyzed first. Sub processes in the model will be identified. These are the regions in the model which have a single entry node to the region and single exit node from the region. Check whether this region can be reduced to a single node. To find the reducibility they are providing some rules. They are providing a declarative method to convert these reduced models to BPEL4WS.

Schattkowsky [17] demonstrates how a fully featured UML 2.0 state machine can be represented using a small subset of the UML state machine features that enables efficient execution. They are trying to directly execute the state machines without converting it to implementation code. It is an alternative to native code generation approaches since it significantly increases portability. The paper describes the necessary model transformations in terms of graph transformations and discusses the underlying semantics and implications for execution.

Rudahl [18] presents a multi language code generator named as YAMDAT (Yet Another MDA Tool). As the name indicates, it's an MDA tool. It generates C++ and Java code from UML designs of the system. UML models will be represented in XML and this XML representation is the code model in the tool. They generate skeleton code for all methods and attributes in the UML class diagram. Moreover, unit test framework will be generated for the class. YAMDAT generates finite state machine class from each state diagram of the class.

Bajwa [19] presents a rule based production systems for automatic code generation in java. They take the requirement scenario in English language and will automatically generate UML diagrams for these scenarios. It has mainly five steps. In step 1 they accept the text input, i.e., the scenario description in English. In step 2 they do the text understanding using natural language processing. The knowledge extraction will be done in step 3. In this step, classes and objects, and their attributes will be identified. In step 4, UML class diagrams will be drawn based on the knowledge extracted in step 3. Finally the skeleton code generation is done in step 5.
Ruben Campos have proposed a method for xUML engine which will internalize the details behind translating UML models into a text-based program [20]. Their proposed xUML engine is comprised of the UML class, sequence, and activity diagrams in conjunction with the Java language. The sequence diagram is selected as the focal point of execution in that xUML Engine. It uses the class diagram as the entry point in implementing the class methods and the Activity diagrams are

used to implement the details of a class method. The xUML Engine is implemented in Java and the models are executed on top of the Java Virtual Machine [20].

VII. Conclusion

In this paper we have presented a new method for generating implementation code from the system design. Here we have use state machine, activity and sequence diagrams. Since activity diagram is suitable for business process modeling our method helps us to automate the business processes. Sequence diagram shows the logic of message passing between different objects in a scenario. Hence, the use of sequence diagram helps us to implement the sequence of actions in the business process. State machines help to synchronize the actions of each object.

We presented a meta model for activity, sequence, and statemachine based on our method. The implementation and testing of our method shows that it is much better than other rival tools.

Our method gives an efficient way to export and import system designs between the CASE tools with the help of XMI representation of the system designs. Moreover, the activity diagram and sequence diagrams are linked effectively to generate maximum implementation code. We are generating the implementation code in Java, which is a widely accepted and user friendly programming language.

The evaluation of our prototype shows that this method can produce 80% complete source code from the system design. The rest of the code can be incorporated in the final source code if we include some more design diagrams, like class diagram, use case diagram etc. in our system design.

References

[1] I. Jacobson, . Rumbaugh, and G. Booch. The Unified Modelling Language Reference Manual. Addison-Wesley, 1999.

[2] G. Booch, J. Rumbaugh, and I. Jacobson. The Unified Modelling Language User Guide. Addison-Wesley, 1999.

[3] J. Rumbaugh, I. Jacobson, and G. Booch. The Unified SoftwareDevelopment Process. Addison-Wesley, 1999.

[4] "Object Constraint Language" , OMG Available Specification, Version 2.0, May 2006

[5] "Model Driven Architecture", OMG specification

[6] Cyprian F. Ngolah and Yingxu Wang, "Exploring Java Code Generation Based on Formal Specifications in RTPA", *Canadian Conference on Electrical and Computer Engineering* 2004 - Volume IV, pp. 1533-36. 2004.

[7] Asma Charfi, et.al. "Does Code Generation Promot or Prevent Optimizations?", *13th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing (ISORC),* pp. 75 – 79. 2010.

[8] Tomas G Moreira, et. al. "Automatic code generation for embedded systems: from UML specifications to VHDL Code", *8th IEEE International Conference on Industrial Informatics (INDIN),* pp. 1085 – 1090, 2010.

[9] Madhusudhan Govindaraju, "XML Schemas Based Flexible Distributed Code generation Framework", *IEEE International Conference on Web Services (ICWS),* pp. 1212 – 1213, 2007.

[10] Mathupayas Thongmak, Pornsiri Muenchaisri. "Design of Rules for Transforming UML Sequence Diagrams into Java code", *Ninth Asia-Pacific Software Engineering Conference (APSEC'02),* IEEE, pp 485-494, 2002.

[11] Philip Samuel, Sunitha E V, "Automatic Code Generation using Model Driven Architecture", *Proceedings of 2009 IEEE International Advance Computing Conference (IACC 2009)* Patiala, India, pp. 2339 – 2344, March 2009.

[12] Cristian Georgescu, "Code Generation Templates Using XML and XSL", *C/C++ Users Journal - Mixed-language programming, ACM*, Volume 20 Issue 1, pp. 6-19, January 2002.

[13] Q.Long, Z.Liu et.al., "Consistent Code Generation from UML Models", *Proceedings of Australian Software Engineering Conference*, 2005.

[14] William Harrison, Charles Barton, Mukund Raghavachari, "Mapping UML designs to Java", Proceedings of the 15th *ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, pp. 178 - 187 , 2000.

[15] Eivind Bjoraa, Torgeir Myhre, Espen Westlye Straapa, "Generating Java Skeleton From XMI", *Open Distributed Systems*, Agder University College, 2000.

[16] J Kochler, R Hauser, S Sendall, M Wahler, "Declarative techniques for model-driven business process integration", *IBM Systems Journal*, Volume 44, No 1, pp. 47-65, 2005.

[17] Tim Schattkowsky, Wolfgang Muller, "Transformation of UML State Machines for Direct Execution", VLHCC, *Proceedings of the 2005 IEEE Symposium on Visual Languages and Human-Centric Computing,* pp. 117 - 124 , 2005.

[18] Kurt T Rudhal, Sally E Goldin, "Adaptive multi-language code generation using YAMDAT", *Proceedings of ECTI-CON 2008, Proceedings of the 5th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology*, , 2008.Volume 1, pp. 181 – 184. May 2008.

[19] Imran Sarwar Bajwa, M. Imran Siddique, M. Abbas Choudhary," Rule based Production Systems for Automatic Code Generation in Java", *Proceedings of the International Conference on Digital Information Management,* pp. 300 – 305. 2006

[20] Ruben Campos, "Model Based Programming: Executable UML with Sequence Diagrams", CS Thesis, 2007.

Author Biographies

Sunitha E V

She is Assistant Professor of IT department at Toc H Institute of science and Technology, Kerala, India. Her research intrest are in software engineering and object oriented modelling. She received a B-Tech in IT and an M-Tech in Software engineering from Cochin University of Science and Technology (CUSAT), Kerala, India. She is currently persuing PhD in Software Engineering at CUSAT.

Philip Samuel

He is Head and Associate Professor of IT Division, SOE at CUSAT, Kerala, India. His research interests are Software Engineering, Object Oriented Modeling and Design, Mobile Communication, and Ad hoc Networks. He received M Tech in computer science from CUSAT, and Ph D from IIT Kharagpur, India. He had published several research papers in this area.