International Journal of Computing Academic Research (IJCAR)
ISSN 2305-9184 Volume 2, Number 5(October 2013), pp. 167-181

http://www.meacse.org/ijcar

A Framework for Model-Based Code Generation from a Flowchart

Basma Moukhtar Hussein[1] and Akram Salah[2]

[1&2] Computer Science Dept. , Faculty of Computers and Information, Cairo University, Giza, Egypt

Abstract

Productivity is a key concept in today's software industry, that's why a lot of researches became interested in automatic Code Generation. But these researches focused only on the application structure and high-level design details. In this research, a framework for automatic code generation from a flowchart is introduced, based on metamodels for the flowchart and for the programming language of the generated code.

Keywords: Model, Platform Independent Model (PIM), Platform Specific Model (PSM), MetaModels, Meta Object Facility (MOF), Model Driven Architecture (MDA), Atlas Transformation Language (ATL).

Introduction

Model-based Code generation is a time-saving technique that helps software engineers to be more productive by reducing redundant hand-coding. In this world of increasingly code-intensive frameworks, the value of replacing laborious hand-coding with code generation is acute and, thus, its popularity is increasing.

This idea of the automatic code generation emerged from the concept of Model Driven Architecture (MDA). The MDA is a new way of writing specifications, based on a platform-independent model. A complete MDA specification consists of a definitive platform-independent based UML model, plus one or more platform-specific models and interface definition sets, each describing how the base model is implemented on a different middleware platform. The MDA focuses primarily on the functionality and behavior of a distributed application or system, not the technology in which it will be implemented. This raises the level of abstraction at which designers and developers interact with the software systems they are building [1].

There are some existing code generation tools that rely on the concept of MDA to do code generation such as Acceleo [2], Andromda [3] and others. But all these tools focus only on the generation of the high level components of the application such as packages and classes skeletons, Data Access Layer and presentation layer based on the application design. But very few tools focus on the code in the business layer which represents the business logic of the application, such as [4, 5] that generates a code from a structured flowchart.

This research develops a framework for code generation from a flowchart which represents the business logic of a specific function in the application. Metamodels for both the flowchart and for the programming language of the generated code (java) are introduced. And then the developed

framework is verified and tested and finally it's is applied to transform the flowchart metamodel to the java code.

Background

Models

A model of a system is a description or specification of that system and its environment on abstract form for some certain purpose. A model is often presented as a combination of drawings and text. The text may be in a modeling language or in a natural language [6]. A model can be one of two types with respect to the level of abstraction:

a) A Platform Independent Model (PIM)

The PIM provides formal specifications of the structure and function of the system that abstracts away technical details [6].

b) A Platform Specific Model (PSM)

PSM is expressed in terms of the specification model of the target specific platform. PSM have to use the platform concepts of exception mechanisms, parameter types (including platform-specific rules about objects references, value types, semantics of call by value, etc.), and component model [6].

Metamodels

Metamodel is a collection of "concepts" (things, terms, etc.) within a certain domain. A model is an abstraction of phenomena in the real world; a metamodel is yet another abstraction, highlighting properties of the model itself. A model conforms to its metamodel in the way that a computer program conforms to the grammar of the programming language in which it is written; a model that respects the semantics defined by a metamodel is said to conform to this metamodel.

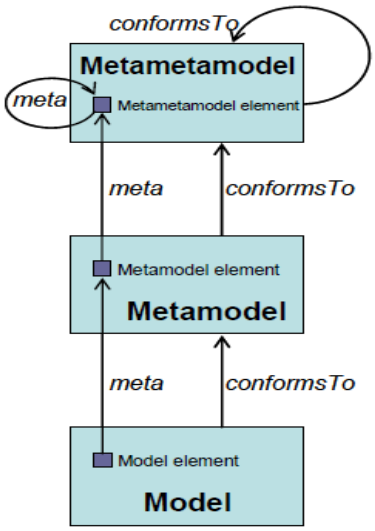


Fig. 1 : Models conforms to MetaModels

Meta Object Facility (MOF)

The Meta-Object Facility (MOF) technology provides a model repository that can be used to specify and manipulate models, thus encouraging consistency in manipulating models in all phases of the use of MDA [6]. MOF defines an abstract language and a framework for specifying, constructing, and managing technology neutral metamodels (e.g. UML, MOF itself).

Model Driven Architecture

The MDA is "an approach to IT system specification that separates the specification of system functionality from the specification of the implementation of that functionality on a specific technology platform."[6]

The MDA approach allows the same model to be realized in multiple different platforms through auxiliary mapping " transforming the elements of a model conforming to a particular metamodel into elements of another model that conforms to another (possibly the same) metamodel" [6].

Model to Model Transformation

Model transformation is the process of converting one model conforming to a metamodel to another model conforming to another (or same) metamodel using a transformation model. This can be illustrated in Fig. 2.

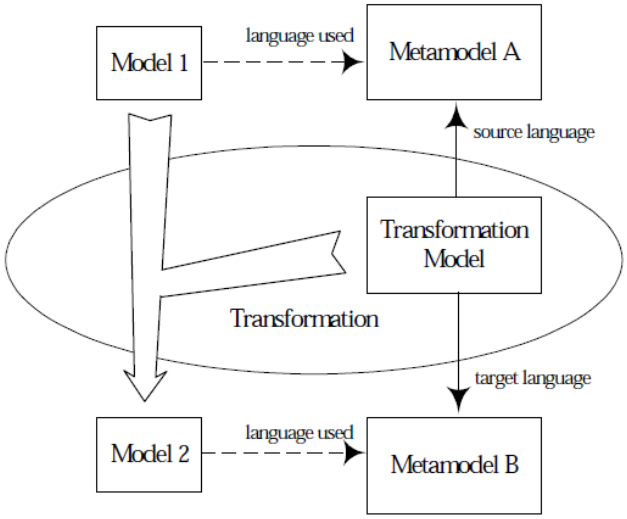


Fig. 2: Model to Model Transformation [6]

Atlas Transformation Language (ATL)

To make the transformation from flowchart to code, we used the Atlas transformation Language (ATL). ATL is a model transformation language specified as both a metamodel and a textual concrete syntax. In the field of Model-Driven Engineering (MDE), ATL provides developers with a means to specify the way to produce a number of target models from a set of source models.

An ATL transformation program is composed of rules that define how source model elements are matched and navigated to create and initialize the elements of the target models. Besides basic model transformations, ATL defines an additional model querying facility that enables to specify requests onto models. ATL also allows code factorization through the definition of ATL libraries.

ATL is applied in a transformational pattern shown in Fig. 3. In this pattern a source model "Ma" is transformed into a target model "Mb". The transformation is driven by a transformation definition (or a transformation program) mma2mmb.atl written in the ATL language. The transformation definition is a model. The source and target models and the transformation definition conform to their metamodels "MMa"," MMb", and "ATL" respectively. The metamodels conform to the "MOF" metametamodel [7].

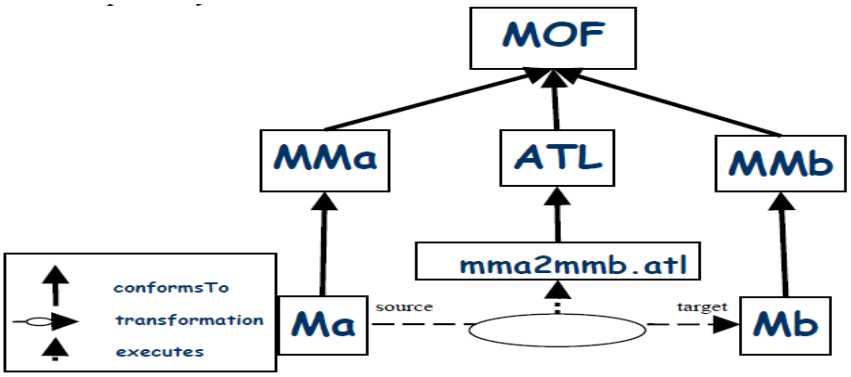


Fig. 3 : ATL Transformation Pattern [7]

The Proposed Framework

The proposed MetaModels

As this research uses the ATL in the code generation, and ATL uses metamodels in the model-to-model transformation, metamodels are needed for both the input and output models (flowchart and java code).

As there is no standard metamodel for the flowchart, we introduced a metamodel to be used in the transformation (this will be covered in the following subsection). As for java, there is a standard metamodel developed by the OMG. But this metamodel is found to be too complicated for this research, so a simplified metamodel for java is introduced in a following subsection.

a) The Proposed Flowchart Metamodel

The proposed flowchart consists of two main components Nodes and Transitions. The nodes can be of different node types (start, stop, input, output, declare, calculation, decision and convergence). The convergence node type is not transformed to any code, it's only used in the flowchart representation for if statements and loops. Also, each node has a set of incoming transitions and outgoing transitions.

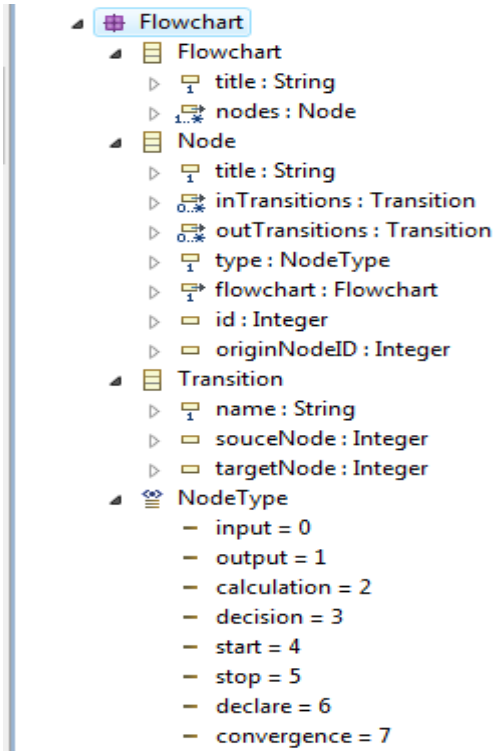


Fig. 4: Flowchart Metamodel

b) The Proposed Java Metamodel

The proposed java metamodel represents the main component of the basic java code, which is the java statement. The java statement can be of type If, For, Assignment, PrintOutput, Input or VariableDeclaration.

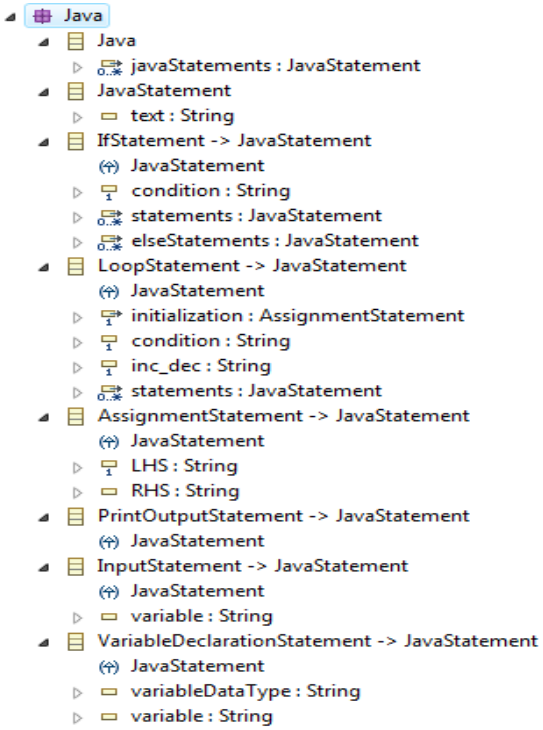


Fig. 5: Simplified Java Metamodel

Flowchart to Java Code Transformation

In the proposed approach, a flowchart is transformed to java code using ATL transformation rules. The flowchart and the code are considered as models. Each of them conforms to a metamodel. We need to transform a sample flowchart model that conforms to the flowchart metamodel to its corresponding code model that conforms to the code metamodel (both metamodels are introduced in the previous section).

This is done with the following sequence:

1. The flowchart as a graph (drawing) is transformed to xml model that conforms to the proposed flowchart metamodel.
2. The xml model is then transformed to the java model in xml.
3. The xml java model is transformed to java code.

It is assumed in this research that the transformation of the flowchart from a drawing to xml model (conforming to the proposed metamodel) is done. So, the transformation is done on two stages. First we transform the flowchart metamodel to the java metamodel. Second, we transform the java metamodel to java code. This can be represented in Fig. 6:

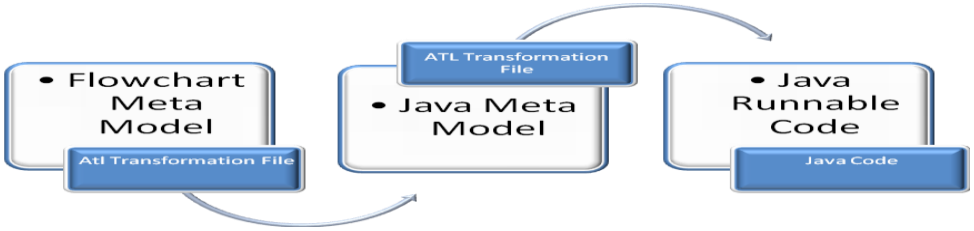


Fig. 6: Overview of the proposed framework

In Details:

First Stage: Flowchart metamodel to Java metamodel

In this stage, the flowchart metamodel is transformed to the java metamodel, by applying the ATL rules specified in the "flowchartMM2javaMM.atl" file.

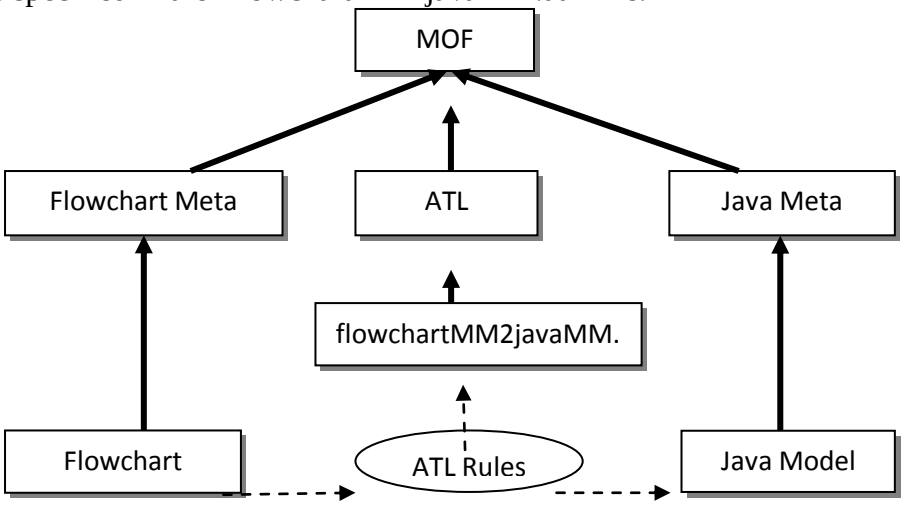


Fig. 7: Transformation of Flowchart metamodel to Java metamodel

Second Stage: Java metamodel to Java code

In this stage, the java metamodel is transformed to the java code, by applying the ATL query specified in the " JavaMM2JavaCode.atl" file.

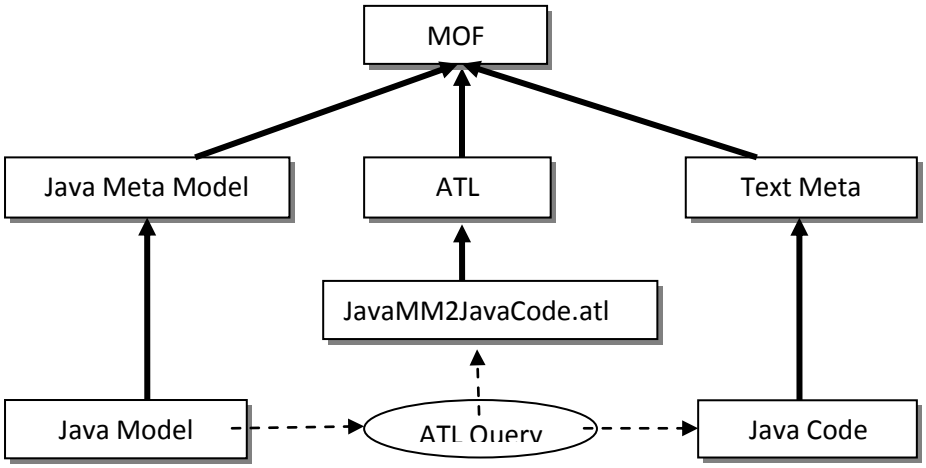
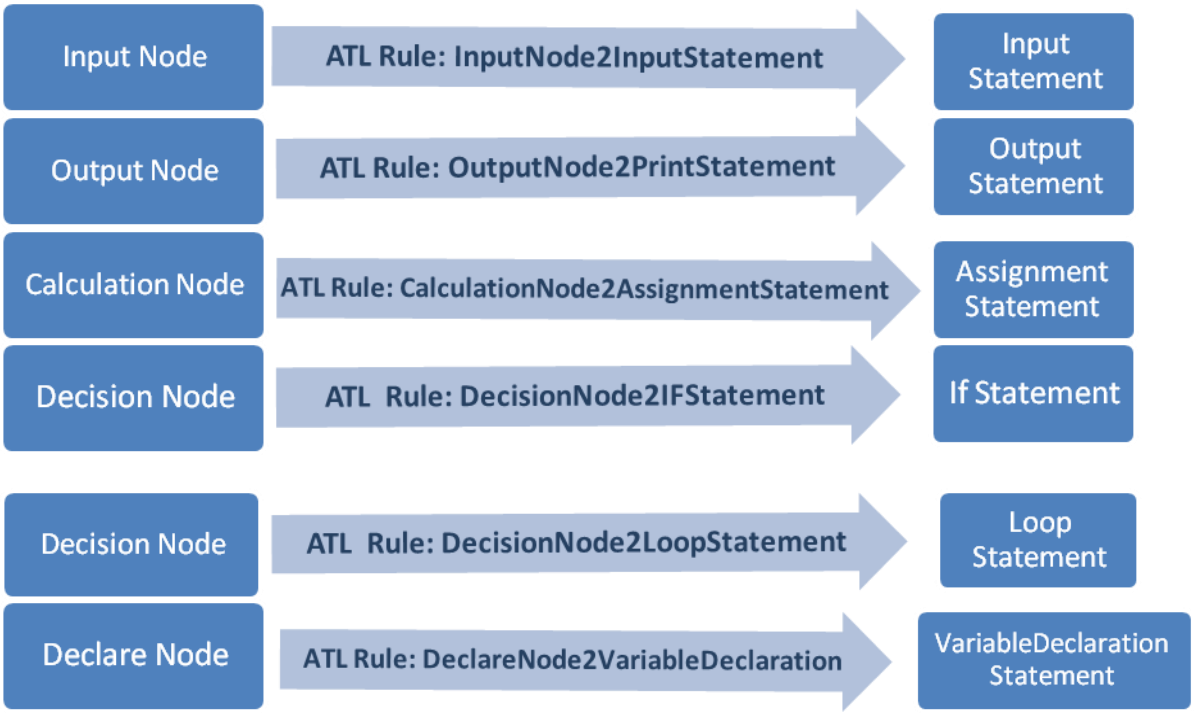


Fig. 8: Transformation of Java metamodel to java code

More Implementation Details

First Stage: Flowchart metamodel to Java metamodel

Each node type in the flowchart metamodel is transformed to its corresponding java statement, using the atl rules written in the atl transformation file, as follows:



For example, if we need to transform a flowchart calculation node to a java AssignmentStatement, the following ATL rule is used:

```
rule CalculationNode2AssignmentStatement
{
    from
      rule CalculationNode2AssignmentStatement
{
    from
      node : Flowchart!Node (
             node.type=#calculation
      )
    to
      stmt : Java!AssignmentStatement(
             LHS<-node.title.substring(1,node.title.indexOf('=')) ,
             RHS<-node.title.substring(node.title.indexOf('=')+2,node.title.size())

             )
}
```

This ATL rule match the attributes of each calculation node in the source model to their corresponding attributes of the Java AssignmentStatement in the target model. For example, if we have a calculation node with title "x=5", this rule will consider "x" as the left-hand side (LHS) of the assignment statement (all the characters before the "=" operator) and "5" as the right-hand side (RHS) of the assignment statement (all the characters after the "=" operator).

Second Stage: Java metamodel to Java code

In this stage, each generated java statement "that conforms to the proposed java metamodel" is transformed to a run-able java code, using the atl query in the atl file " JavaMM2JavaCode.atl". The query is as follows:

```
query Java2Code_query = self.convert(Java!JavaStatement.allInstances())
                    .writeTo('/FlowChart2Java_24.0_testing/out-
codes/tc8_out_testLoopNestingIF.java');

uses MM2Code_Helpers;
```

The "convert" helper is as follows:

```
helper def:convert(s: Sequence(Java!JavaStatement) ) : String =
      thisModule.removeSubStatements(s)->
             iterate(st; res : String = ''| res + st.toString() + '\n');
```

This query takes all the instances of the java statements and converts each to its corresponding java run-able code, using the atl helpers in the atl file

"MM2Code_Helpers.atl".

For example, to convert a java "AssignmentStatement" with "LHS" and "RHS" attributes to a Java Assignment run-able Statement, the following helper is called:

```
helper context Java!AssignmentStatement def: toString() : String =
      self.LHS + ' = ' + self.RHS + ';';
```

This will convert a Java "AssignmentStatement" with LHS="x" and RHS=5 to a run-able java statement written as follows:

```
X=5;
```

Related Work

Many researches exist that deal with the model-based code generation in general [8, 9, 10], but very few researches make code generation from a flowchart [4, 5].

"Making model-based code generation work" [8], focuses on what is a "true" model-based code generator and how metamodels can guide the code generator.

"Template and model-based code generation for MDA Tools" [9], introduces a code generator that uses intermediate data and makes use of code templates for the final transformation into pieces of text. But this generator generates the general structure of the system (classes and methods with empty body).

"Automatic Code Generation from Design Patterns" [10], a tool based on design pattern can automatically generate a design pattern of abstract level.

The papers in [4] and [5] are more related to this research, as they generate code from a flowchart.

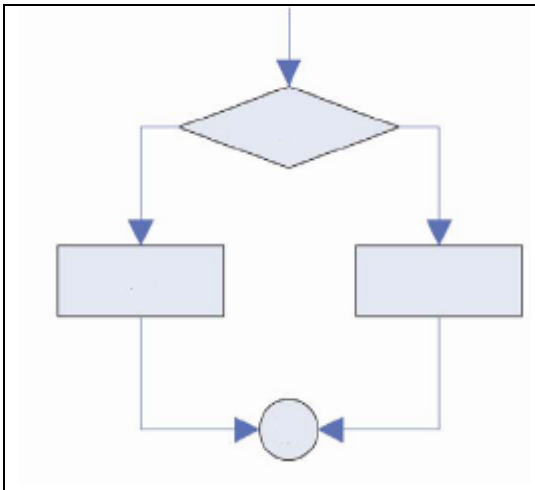
For "Visual Programming using Flowchart" [4], the main concept is to utilize the advantages of flowchart. The programmer just describes his idea and algorithm easily by drawing the flowcharts, and then the application compiles and links all flowcharts to create EXE file.

"Research and Application of Code Automatic Generation Algorithm Based on Structured Flowchart" [5], generates c-code from a structured flowchart using recursive method described in their paper.

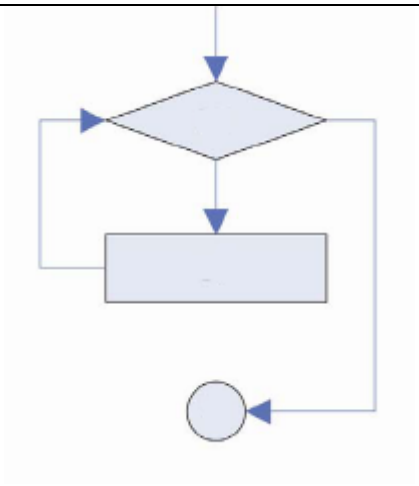
Neither [4] nor [5] use specific metamodels for the input and output of the code generation process. Also, they don't mention a clear framework for their work. Another difference is that both researches use a graphical tool to draw the flowchart, while the proposed framework deals with the flowchart as a model represented in xml even if there's no drawing for the input flowchart. If a drawing exists, it will need to be transformed so that it conforms to the proposed flowchart metamodel.

Verification of the Framework

In [5], a recursive algorithm for the transformation is used, so the exhaustive method is used in the verification. This can't be applied for the proposed framework as transformation rules are used for model-to-model transformation. But they rely on that the flowchart consists of some basic structures such as if and loop:



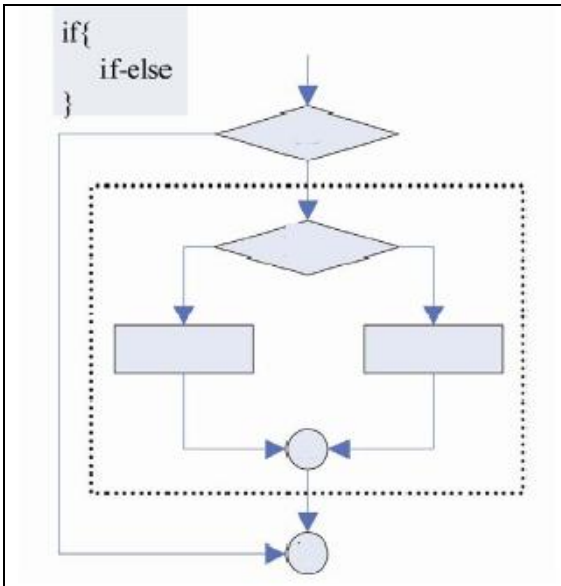
(a) Selection Structure (IF)



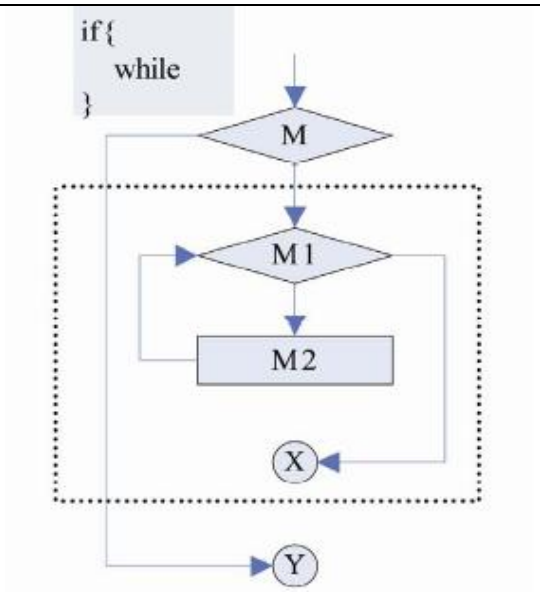
(b) Loop structure (while/for)

Fig. 9: Flowchart basic structures

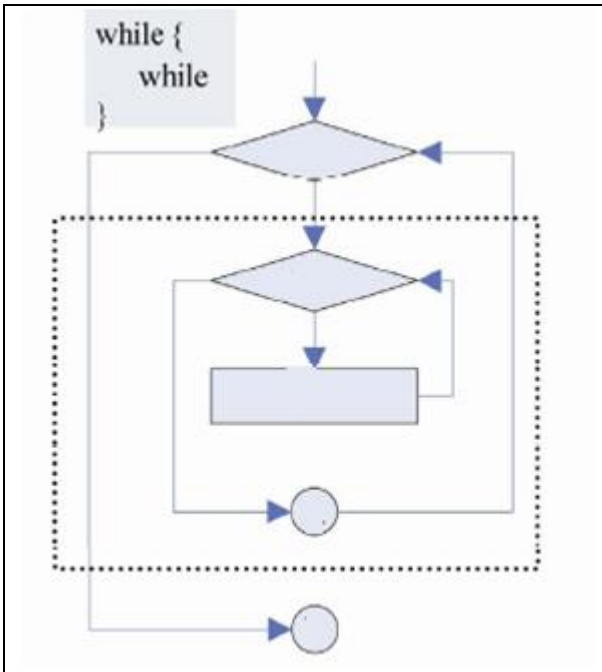
Any flowchart is a combination of these two basic structures, they can be in a sequence (selection followed by a loop or vice versa), or can be nested as illustrated in the Fig. 10:



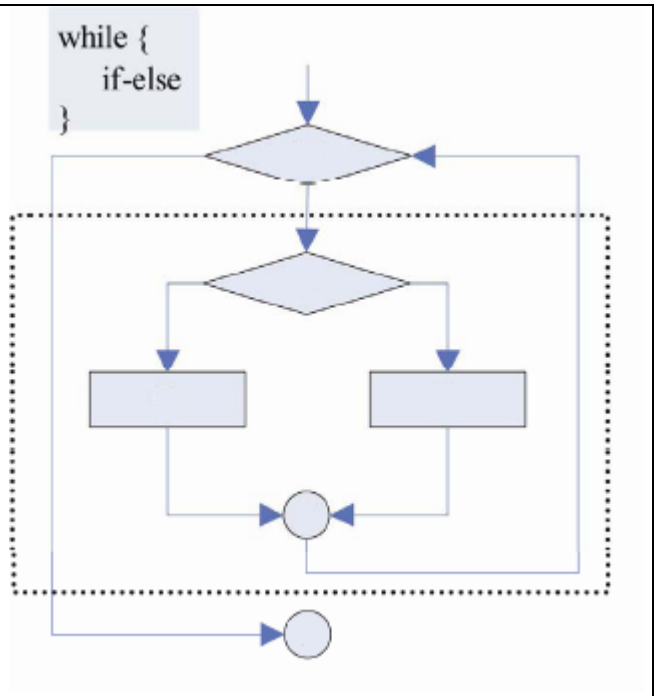
(a) IF nesting IF



(b) IF nesting Loop



(c) Loop nesting Loop



(d) Loop nesting IF

Fig. 10: Flowchart nesting structures

As long as any flowchart can only be in a form of the above forms, these structures are used in the proposed approach to verify that any flowchart can be transformed to code using the proposed framework.

The framework test cases are designed so that they test all the above structures as shown in Table 1:

Table 1 : The Proposed Framework Test Cases

#	Description of the input flowchart	Expected Output	Result
1	Flowchart with any sequence of simple statements (No Selection or Loops)	Run-able java code with the same structure	Succeeded
2	Flowchart with selection only	Run-able java code with the same structure	Succeeded
3	Flowchart with loop only	Run-able java code with the same structure	Succeeded
4	Flowchart with selection followed by loop or vice versa (as a sequence not nesting)	Run-able java code with the same structure	Succeeded
5	Flowchart with selection nesting selection	Run-able java code with the same structure	Succeeded
6	Flowchart with selection nesting loop	Run-able java code with the same structure	Succeeded

7	Flowchart with loop nesting loop	Run-able java code with the same structure	Succeeded
8	Flowchart with loop nesting selection	Run-able java code with the same structure	Succeeded

A simple flowchart to a simple java code

This is an example that shows how a flowchart is transformed to its corresponding java code. The example contains a loop statement nesting if statement (test case 8).

The flowchart as a graph

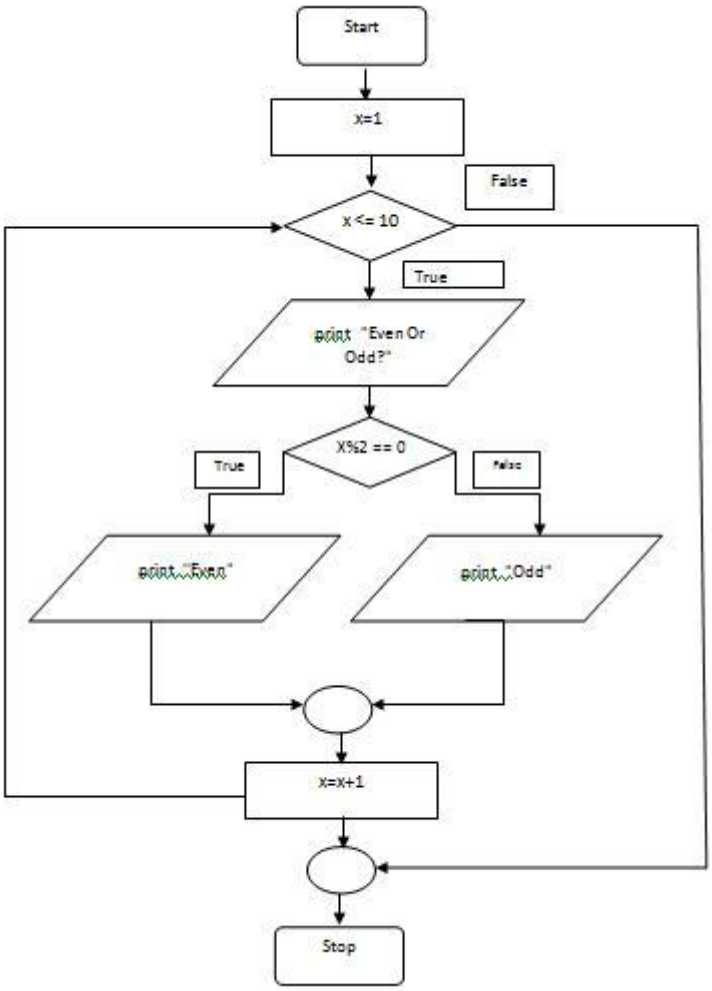


Fig. 11: Flowchart Example

The flowchart represented as xml (input file)

```
<?xml version="1.0" encoding="ASCII"?>
<xmi:XMI xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI" xmlns="Flowchart">
  <Flowchart title="testLoopNestingIF" >

    <nodes  title="int x" id="0"  type="declare">
        <outTransitions targetNode="1"/>
    </nodes>

    <nodes  title="x=1" id="1"  type="calculation">
        <outTransitions targetNode="2"/>
    </nodes>

    <nodes  title="x &lt;= 10" id="2"  type="decision">
        <outTransitions name = "true" targetNode="3"/>
        <outTransitions name = "false" targetNode="9"/>
    </nodes>

    <nodes  title="print &quot;Even Or Odd?&quot;" id="3"  type="output">
        <outTransitions targetNode="4"/>
    </nodes>

    <nodes  title="x%2 == 0" id="4"  type="decision">
        <outTransitions name = "true" targetNode="5"/>
        <outTransitions name = "false" targetNode="6"/>
    </nodes>

    <nodes  title="print &quot;Even&quot;" id="5"  type="output">
        <outTransitions targetNode="7"/>
    </nodes>

    <nodes  title="print &quot;Odd&quot;" id="6"  type="output">
        <outTransitions targetNode="7"/>
    </nodes>

    <nodes id="7" originNodeID="4" type="convergence">
    </nodes>

    <nodes  title="x=x+1" id="8"  type="calculation">
        <outTransitions targetNode="2"/>
    </nodes>

    <nodes id="9" originNodeID="2" type="convergence">
    </nodes>

  </Flowchart>
</xmi:XMI>
```

Fig. 12: Flowchart as xml

The java code represented as xml (output file)

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Java xmi:version="2.0"
    xmlns:xmi="http://www.omg.org/XMI" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns="Java">
  <javaStatements xsi:type="VariableDeclarationStatement" variableDataType="int" variable="x"/>
  <javaStatements xsi:type="AssignmentStatement" LHS="x" RHS="1"/>
  <javaStatements xsi:type="LoopStatement" condition="x &lt;= 10">
    <statements xsi:type="PrintOutputStatement" text="&quot;Even Or Odd?&quot;"/>
    <statements xsi:type="IfStatement" condition="x%2 == 0">
      <statements xsi:type="PrintOutputStatement" text="&quot;Even&quot;"/>
      <elseStatements xsi:type="PrintOutputStatement" text="&quot;Odd&quot;"/>
    </statements>
    <statements text="convergence}"/>
    <statements xsi:type="AssignmentStatement" LHS="x" RHS="x+1"/>
  </javaStatements>
  <javaStatements text="convergence}"/>
</Java>
```

Fig. 13: Java Code as xml

The run-able Java code (final output file)

```
int x;
x = 1;
while(x <= 10)
{
    System.out.println("Even Or Odd?");

    if(x%2 == 0)
    {
        System.out.println("Even");
    }
    else
    {
        System.out.println("Odd");
    }

    x = x+1;
}
```

Fig. 14: The final output (Java code)

Conclusion and Future Work

A new framework for code generation from a flowchart is produced and verified for correctness. This is supposed to speed up the development process especially for novice programmers. But this framework needs to be tested on more complex cases. Also, it needs to be integrated with a graphical tool that generates a flowchart that conforms to the proposed metamodel. Finally, if this framework is integrated with any of the ready existing tools that generate the general structure of the system, a nearly complete application can be generated using the code generation tool.

References

[1] Jon Siegel and the OMG Staff, "Developing in OMG's Model-Driven Architecture", Strategy Group, Object Management Group White Paper, November, 2001, Revision 2.6
[2] Acceleo Home Page: http://www.acceleo.org/pages/home/en
[3] Peter Wittmann, "MDA using AndroMDA", http://www.wittmannclan.com

[4] Kanis Charntaweekhun and Somkiat Wangsiripitak, "Visual Programming using Flowchart", Computer Engineering Department, Faculty of Engineering King Mongkut's Institute of Technology Ladkrabang, Bangkok, Thailand.
[5] Xiang-Hu Wu, Ming-Cheng Qu, Zhi-Qiang Liu, Jian-Zhong Li, "Research and Application of Code Automatic Generation Algorithm Based on Structured Flowchart", International Journal of Reviews in Computing, July, 2011.
[6] Joaquin Miller and Jishnu Mukerji, "MDA Guide Version 1.0.1", Copyright © 2003 OMG.
[7] Frédéric Jouault and Ivan Kurtev, "Transforming Models with ATL", ATLAS Group (INRIA & LINA, University of Nantes).
[8] Dr. Juha-Pekka Tolvanen, the CEO of MetaCase, "Making model-based code generation work".
[9] Leif Geiger, Christian Schneider and Carsten Reckord, "Template and model-based code generation for MDA Tools".
[10] Frank Budinsky, Marilyn Finnie, Patsy Yu, Toronto Software Laboratory, John Vlissides, "Automatic Code Generation from Design Patterns", T.J. Watson Research Center.