소프트웨어공학개론 *Team Project #1._*

ROBOT VACUUM CLEANER SASD

T7
201011373최지환
201011376한지승
200611449강동원
200611514임진용

Presentation

- CFG란?
- Overview (Statement of Purpose)
- Visualization Algorithms
  1. Reading the model (Method)
  2. Persistance
- Editor
  1. Zest Layout Algorithms
  2. View
- Outline View
  1. McCabe metric
- Source & References

Overview

A control flow graph (CFG) is a representation, using graph notation, of all paths that might be traversed through a program during its execution. It refers to the order in which the individual statements, instructions, or function calls of an imperative or functional program are executed or evaluated. If the graph is built from code execution the hole program is first compiled than is executed with special parameters so that a log file is generated to obtain the necessary information to build a graph. This makes it only possible if the code can be complied and run, which is not always the case during testing when parts of the implementation might miss. For this reason the evaluation of the code looked like a nice try to see if it is possible to build a correct graph from static analysis only and not compiling or running any of the code which needs to be analyzed. The methods to do it are explained in details to get a better understanding of how the evaluation of code can lead to a CFG. Normally when building control flow graphs we focus on a specific method and very seldom on hole classes. Classes can get big, but methods as well, thats why for this plug-in special functions like node collapsing and expanding or node hovering where made available to help viewing the graph. The aim was to build CFG-s form methods in the quickies and best way possible without extra clicking and compiling, an all-in-one-click operation.

Visualization Algorithms

We can divide the functionalities into two main parts. The first one is the collecting of information from the Java source code and the second part is the processing of the model form a tree into a graph.

Execution order of the functionalities:

– gaining the Java source

– parsing into a AST

– traversing the AST and building the tree model

– persisting the model

– normalizing

– viewing

Reading the Model (Method) - 1

Method	Information
Visit (ExpressionStatement node)	Insertion of a node into the model if a expression statement is encountered. This case covers ternary declarations as well. But they are inserted as if statements in the model.
Visit (VariableDeclarationFragment node)	Insertion of a node into the model if a variable declaration, such as field declaration, local variable declaration or others are encountered.
Visit (IfStatement node)	Insertion of a node into the model if a ifstatement is encountered. This Node contains at least two children as for the graph empty if statements need to be considered as well.

Reading the Model (Method) - 2

Method	Information
Visit (TryStatement node)	Insertion of a node into the model if a trystatement is encountered. Contains as many children as catch statements plus the finally statements and the try block.
Visit (ForStatement node)	Insertion of a node into the model if a forstatement is encountered. It has two children.
Visit (WhileStatement node)	Insertion of a node into the model if a whilestatement is encountered. There is no difference regarding the possible paths between the for and while statements so that we treat them the same.

Reading the Model (Method) - 3

Method	Information
Visit (DoStatement node)	Insertion of a node into the model if a trystatement is encountered. Contains as many children as catch statements plus the finally statements and the try block.
Visit (SwitchStatement node)	Insertion of a node into the model if a forstatement is encountered. It has two children.
Visit (BreakStatement node)	Insertion of a node into the model if a whilestatement is encountered. There is no difference regarding the possible paths between the for and while statements so that we treat them the same.
Visit (ContinueStatement node)	Insertion of a node into the model if a breakstatement is encountered. Contains no child nodes.

Reading the Model (Method) – Ex.

ASTNodeMainVisitor

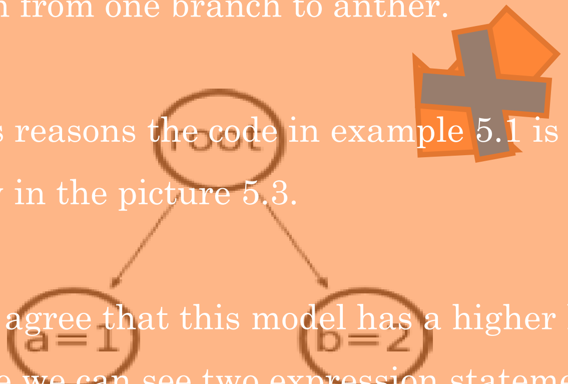
This class implements org.eclipse.jdt.core.dom.ASTVisitor and builds the simple tree model. The model has some differences from the original Java source sequence due to optimization for the building of the graph.

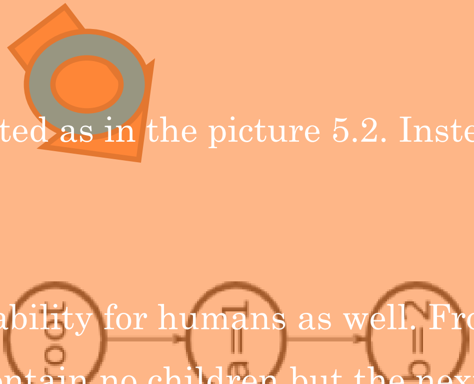
The children of the root element do not hang on it but point to each other as the next nodes to come.

Reading the Model (Method) – Cont.

Example:
void expressionTestMethod(int b,int a) {
a = 1;
b = 2; }

From the code above we can build a tree representation as is the next picture. This model type would fit for the viewer but it would make it very hard and require extra computations to link children from one branch to anther.





For this reasons the code in example 5.1 is not represented as in the picture 5.2. Instead it is build as show in the picture 5.3.

We can agree that this model has a higher level of readability for humans as well. From this example we can see two expression statements which contain no children but the next node. For other nodes it is necessary to use a convention setting the first node of every parent to be the next node from the root and not a child node. Remembering that children of a node can be just nodes within the declared brackets of a statement. An exception is the if-statement but due to the AST parsing these shortcuts are transparent to us.

Persistence

Ones the model has been built, it gets serialized and saved as a file with the .ff3 extension in a subfolder of the project called cfg. The name is given by convention from the <classname>_<methodname>.ff3. To perform this action a serializing class called NodeSerializer was created. If an older version already exists or same class names with same method names were already examined then a confirmation is shown asking if it should be overwritten or not. Changes to the model are not planed so that the reallocation of the graphical nodes or node folding information are discarded as the editor is closed.
The tree model as it is saved is the plain representation of the Java code with the order changes shown above. Therefore a reorganization of the node must be done before it is been shown.

↳ **NodeNormalizer**

Editor

Eclipse offers a base implementation of all workbench editors called EditorPart which can be extended to offer two basic editor types

– textual

– graphical

Editor - Zest Layout

Zest is a visualization toolkit for Eclipse. It has a predefined set of classes, interfaces and operations to help building graphics on a GEF editor. It also has drag and drop support for its elements and uses animations in the initial time. The layout algorithms which the library offers are the one listed below

- Spring Layout Algorithm
- Fade Layout Algorithm
- Tree Layout Algorithm
- Radial Layout Algorithm
- Grid Layout Algorithm

Editor - View

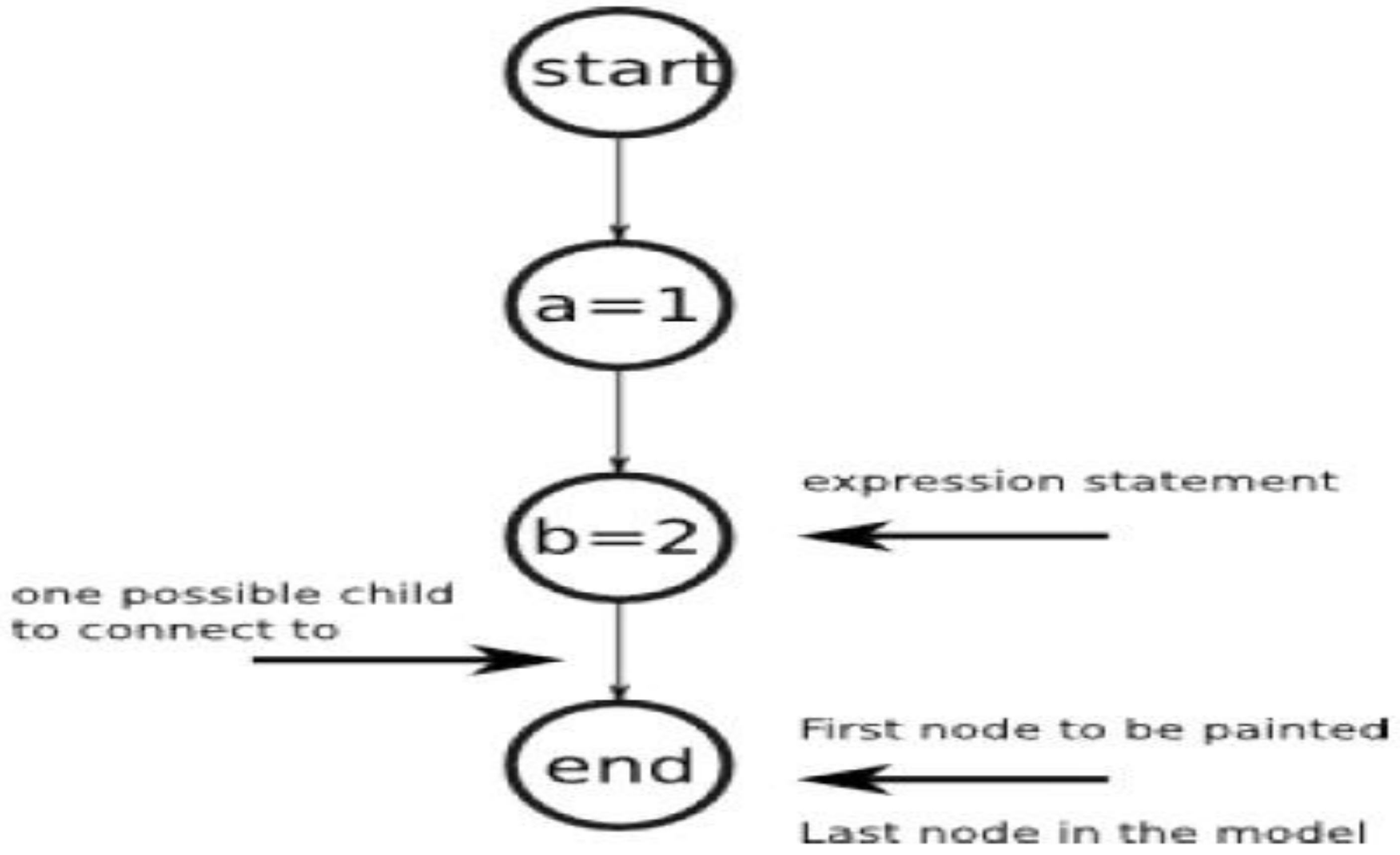
The viewer/editor class is called FlowChartEditor and as we already know, it extends EditorPart and implements IAdaptable.

There are to ways to provide it with the model data.

– To call the editor with the data

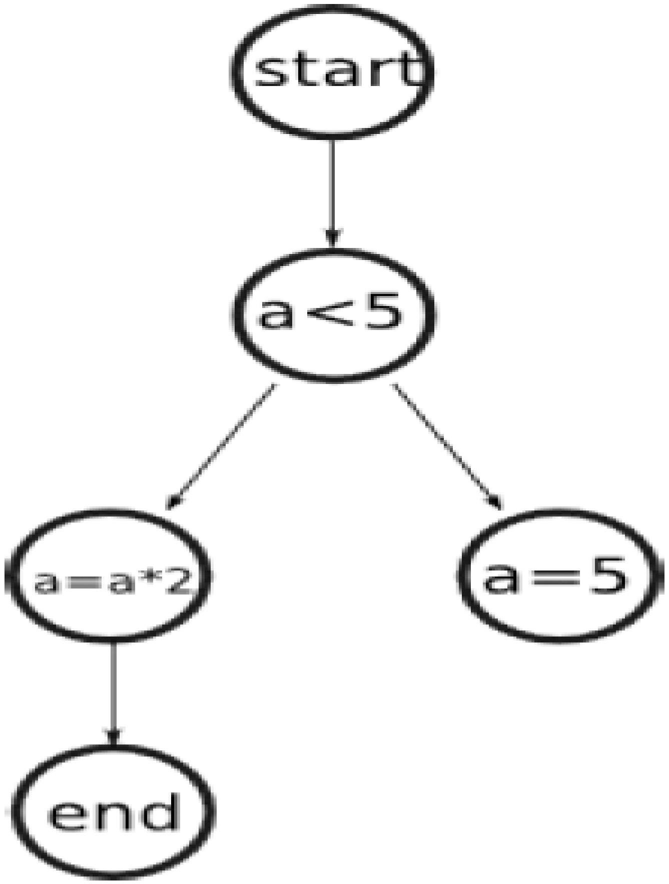
– To execute the file which opens the editor

GraphicBuilder - Standard



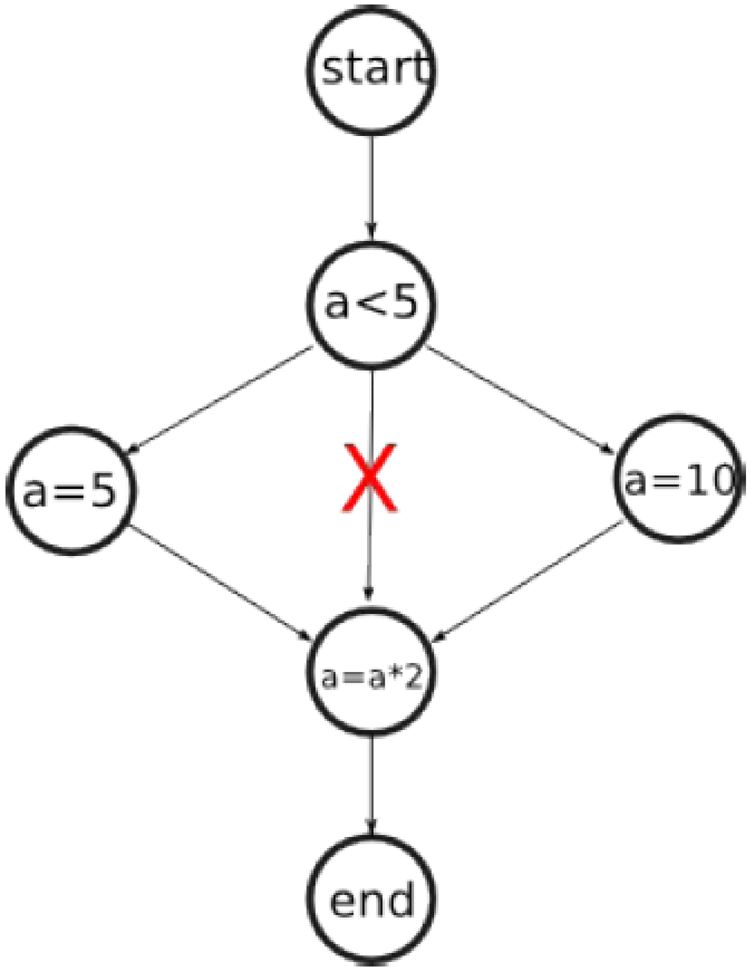
[Figure 6.1 graph model for the expression statement]

GraphicBuilder - If



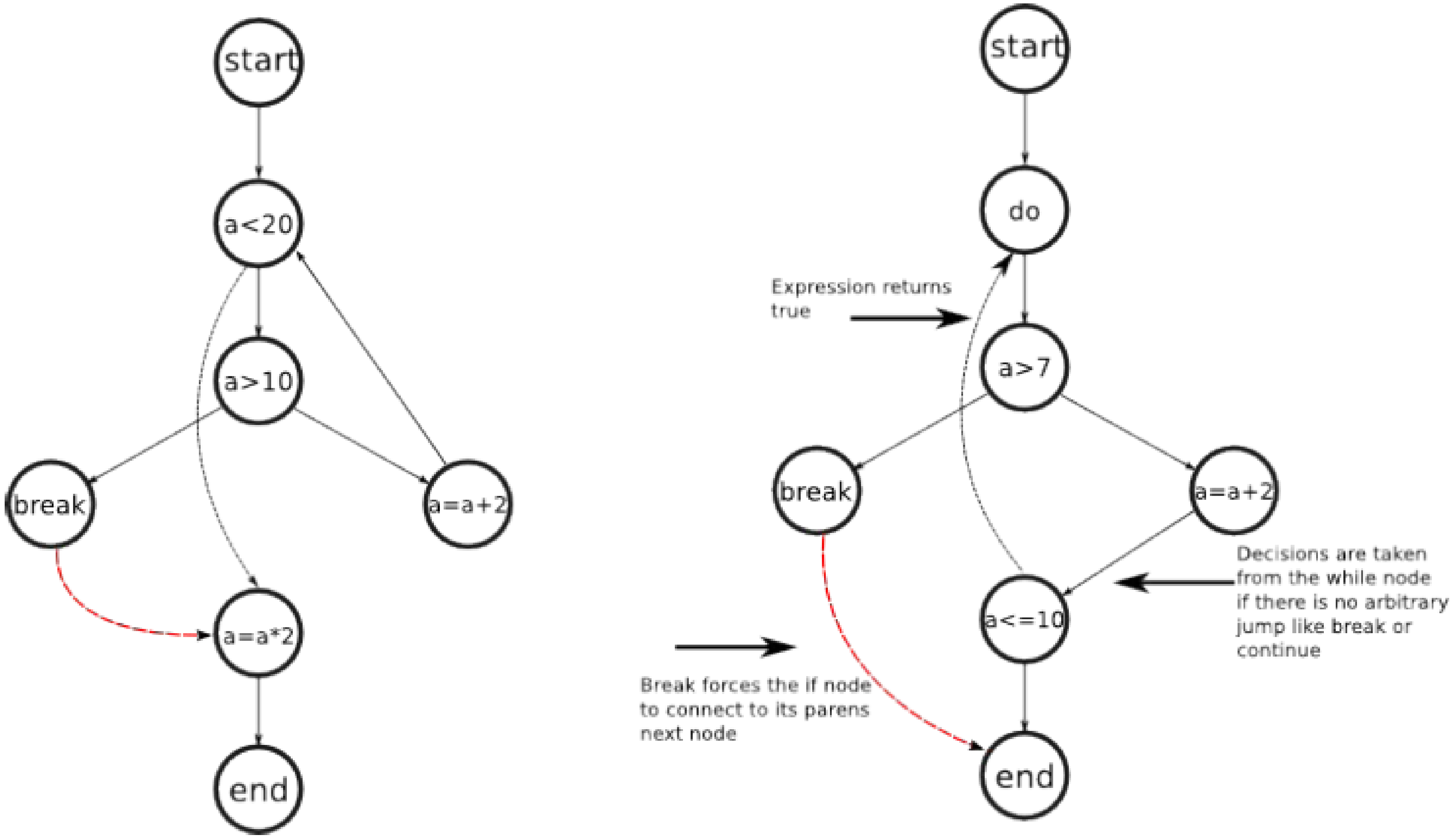
[Figure 6.3 If statement INode representation]

GraphicBuilder - If else



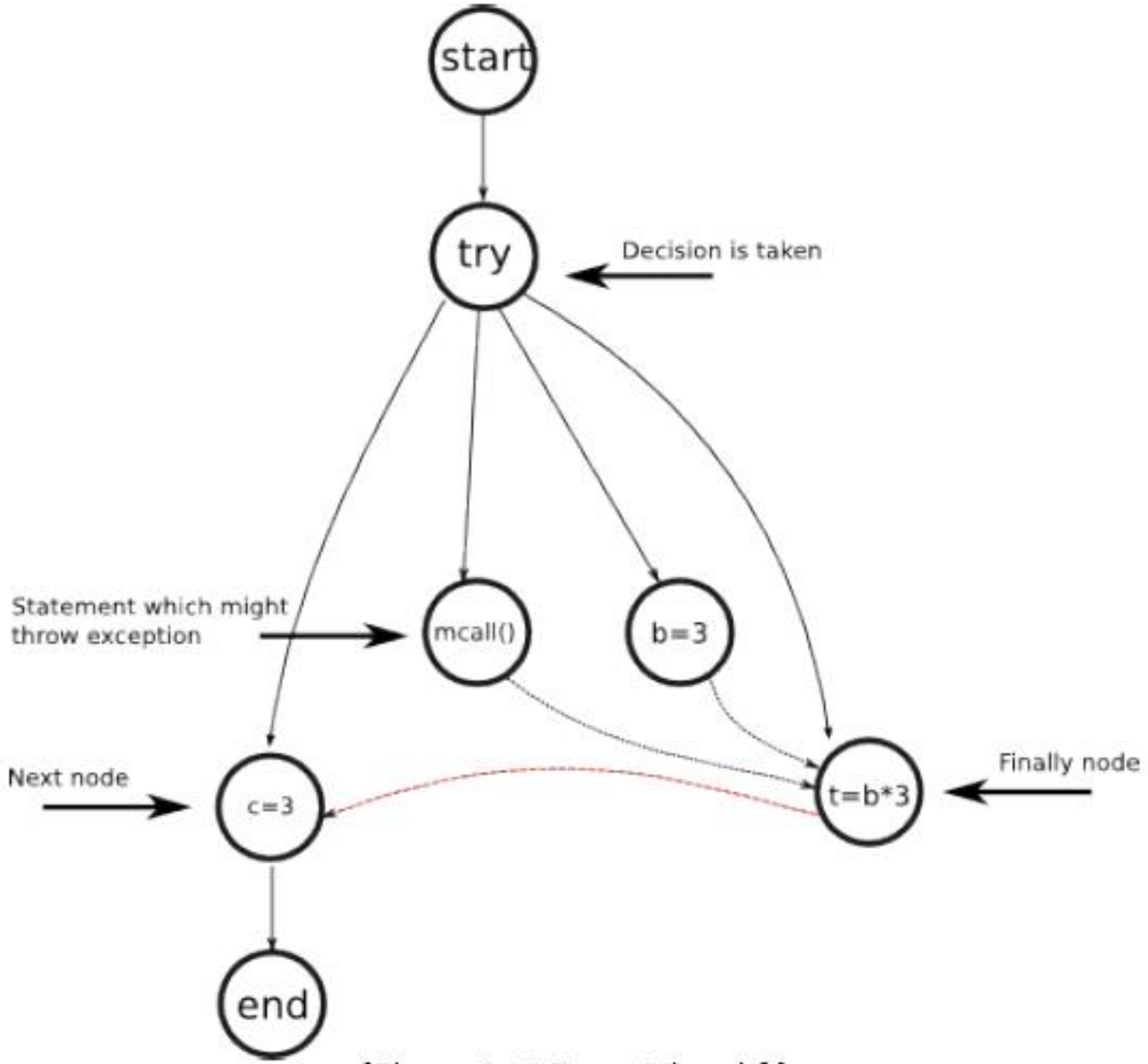
[Figure 6.5 If else statement]

GraphicBuilder - For while, Do while


start
a<20
a>10
break
a=a+2
a=a*2
end
start
do
Expression returns true
a>7
break
a=a+2
a<=10
Decisions are taken from the while node if there is no arbitrary jump like break or continue
Break forces the if node to connect to its parens next node
end

[Figure 6.10 For-while and do-while statements. Graph model]

GraphicBuilder - Try Catch



[Figure 6.14 Try-catch model]

Outline View

The outline view was implemented to show static information about the graph like the number of the nodes and the connections. It is shown automatically with the appearance of the editor so that no extra activation is needed.

MacCabe metric

The measurement of cyclomatic complexity by McCabe was designed to indicate a program's ***testability and understandability (maintainability).***

Source and References

CFG Generator [http://eclipsefcg.sourceforge.net/]

Eclipse Application [http://www.eclipse.org/]

Eclipse SDK API [http://help.eclipse.org/]

Eclipse GEF [http://www.eclipse.org/gef/]

GEF Tutorial [http://www.ibm.com/developerworks/library/os-eclipse-gef11/]

Eclipse Development [http://www.redbooks.ibm.com/redbooks/pdfs/sg246302.pdf]

CodeCover [http://www.codecover.org/]

c1visualizer [https://c1visualizer.dev.java.net/]

Control Flow Graph Factory [http://www.drgarbage.com/]

[linkCTools] [http://java-source.net/open-source/code-coverage]

[OREclipse] O'Reilly Eclipse, Holzner, 2004, Ch 1.3

[AWMeMo02] Metrics and Models in Software Quality Engineering, Second Edition

Kan, 2002, Ch 11.3