State-based dynamic slicing technique for UML integrating activity model

Rasmita Dash, Mamata Manjari Behera and Rajashree Dash

Department of Computer Sc & Engineering, Siksha O Anusandhan University, Bhubaneswar, Orissa, INDIA.

ABSTRACT

Unified Modeling Language has been widely used in software development for modeling the problem domain to solution domain. The major problems lie in comprehension and testing which can be found in whole process. Program slicing is an important approach to analyze, understand, test and maintain the program. It is a technique for analyzing program by focusing on statements which have dependence relation with slicing criterion. Program slicing is of two types (i) Static slicing (ii) Dynamic slicing. Dynamic slicing refers to a collection of program execution and may significantly reduce the size of the program slice because runtime information, collected during execution, is used to compute the program slice. In this paper we introduce an approach for constructing dynamic slice of unified modeling language (UML) using sequence diagram, state chart diagram, class diagram along with the activity diagram. First we construct an intermediate representation known as model dependency graph. MDG combines information extracted from various sequence diagrams along with those from class, state-machine, sequence diagram and activity diagrams. Then dynamic slice is computed by integrating the activity models into the MDG. Finally for a given slicing criterion test cases has been generated.

© 2011 Elixir All rights reserved.
Class Diagrams
A class diagram consists of the parts classes, associations and generalizations, and can exist in several different levels. Below is an identification of three different useful levels, starting with the least detailed.
• Conceptual class diagrams (conceptual model), represent concepts of the problem domain
• High level class diagrams (type model), describe static views of a solution to a problem, through a precise model of the information that is relevant for the software system
• Detailed class diagrams (class model), include data types, operations and possibly advanced relations between classes

Sequence Diagram
UML sequence diagrams are used to model the flow of control between objects. It can be hard to understand the overall flow in a complex system without modeling it.

Sequence diagrams model the interactions through messages between objects; it is common to focus the model on scenarios specified by use-cases.

It is also often useful input to the detailed class diagram to try to model the specified use cases with sequence diagrams, necessary forgotten operations and relations are usually found.

The diagrams consists of interacting objects and actors, with messages in-between them.

State Chart Diagram
UML State charts are most often used for low level design, like modeling the internal behavior of a complicated class. But they are also useful on a higher level on modeling different states of a whole system; this can be compared to the usage of class diagrams on several levels.

Activity Diagram
Activity diagrams can be in many places in the design process; sometimes even before use case diagrams for understanding the workflow of a process. But they can also be used for defining how use cases interact or even for detailed design. Basic elements in activity diagrams are activities, branches.

Proposed model:
To compute the dynamic slice of the UML model, we structure our work into following steps:
1. Development of various UML diagram: Given a problem domain, first we have to develop UML model including class diagram along with various sequence diagram and activity diagram.
2. Construction of an intermediate representation: We propose an intermediate representation for software architecture by integrating various UML diagrams viz., class, sequence and state-machine diagrams into a single system model. Such a representation would capture all relevant information spread across diverse model views into a single structure and can facilitate effective and efficient slicing.
3. Implementation of dynamic slicing algorithm: Based on the constructed intermediate representation, we propose an algorithm for dynamic slicing of UML architectural models using state information.
4. Our slicing algorithm is based on traversing the edges of our intermediate representation for any given scenario execution in the slicing criterion. Through model dependency graph (MDG) traversal, our slicing algorithm would identify the relevant model elements from an architecture based on the dependencies among them to compute dynamic architectural model slice.

Figure 1: Block Diagram of Proposed Model
Model dependency graph (MDG):
In this section, we present an overview of an intermediate representation which we have named Model Dependency Graph (MDG). We first discuss the key elements of MDG and then outline the representation of a generic system using MDG. MDG represents both structural aspects modeled in various class diagrams as well as behavioral aspects modeled in sequence, state-machine and activity diagrams of an architecture. An MDG provides an integrated view of all such UML diagrams.

MDG Nodes
An MDG consists of different types of nodes that correspond to the elements of either class, or sequence diagrams. The different nodes of the MDG are as follows:
1. A class access (CA) node is defined for every class in the UML architectural model.
2. A method access (MA) node is defined for every operation of a class.
3. An attribute (AT) node is defined for every class attribute.
4. A parameter (PR) node is defined for every operation parameter specified in an operation signature.
5. A return (RT) node is defined for every return parameter specified in an operation signature. A predicate class (PC) node is defined for every combined fragment used in a sequence diagram.
6. An interaction (IT) node is defined for every interaction occurrence used in a sequence diagram.

Figure 2: MDG node representation
MDG edges
The nodes of an MDG can be interconnected by two types of dependence edges namely
• class-induced dependence edges
• interaction-induced dependence

We can classify MDG edges into following based on the type and nature of information available from class, sequence, state machine and activity diagram, respectively.
• Edges of the MDG can be based on the elements in the class diagram. Different types of edges in this category include member dependence, method dependence, data dependence and relationship dependence edges. We call all such dependencies class-induced dependence edges.
• Edges of the MDG can be based on the elements in the sequence diagram. We call all such edges message dependence edges.
• Edges of the MDG can be based on the elements in the state-machine diagram. The data dependence edges in the MDG arising because of specific guard conditions that change an object’s state fall under this category. We call all such dependencies state-induced dependence edges.

<table>
<thead>
<tr>
<th>MDG EDGES</th>
</tr>
</thead>
<tbody>
<tr>
<td>Member dependency edge</td>
</tr>
<tr>
<td>Method dependency edge</td>
</tr>
<tr>
<td>Data dependency edge</td>
</tr>
</tbody>
</table>

**Figure 3: MDG edges representation**

**Dynamic Slicing Technique**

A dynamic program slice is that part of a program that “affects” the computation of a variable of interest during program execution on a specific program input. A slice is constituted by the statements that affect the value of the program with respect to the given variable occurrence. The various statements are statements using variables, expressions and assignments and control flow statements.

It is an executable portion of the original program whose behavior is, under the same input, indistinguishable from that of the original program on a given variable ‘V’ at point ‘P’ in the program[1]. Weiser defines a program slice with respect to slicing criterion that consists of program point ‘s’ and a subset of program variables ‘v’ is now called executable backward static slicing criterion that fulfills the requirements of branch testing and also tests all of the independent paths that could be used to construct any arbitrary path. Basis Path testing is a hybrid between path and branch testing.

**Conclusion:**

We have proposed a technique for dynamic slicing using UML state information. Here MDG integrates the structural and behavioral aspects of the architectural design. Test cases will be generated based on the slicing criteria. Rather than generating test cases for all the diagram, only for the slices test cases will be generated. The slices help to achieve this simplification.

**References:**


