An Approach for UML based Scenario Oriented Slicing

R. Senthil Kumar, M. Gokul Nath, U. Raaghul
VIT University/SCS, Vellore, Tamil Nadu, India
Email: rsenthilkumar@vit.ac.in, gklnth@yahoo.co.in, mailbagrahul@yahoo.co.in

Abstract—Understanding a large software product is known as a very challenging task. One way that facilitate program understanding is program slicing. Slicing is an important decomposition technique for program understanding[3]. However, traditional slicing methods tend to produce too large slices for human inspection as modern programs are often huge and static program analyses are hard to be precise enough. One possible solution to such problem could be combining other program decomposition technique together with program slicing. A scenario oriented program slicing method is to slice programs under specified execution scenarios. The scenarios provide a clear and easy understanding functional decomposition for the system, while the new slicing method can help figure out how a computation is implemented in a given scenario by effectively reducing the amount of code that a user needs to inspect.

I. INTRODUCTION

As all known slicing is an important decomposition technique usefull for program understanding. Although, traditional static slicing methods tend to produce too large slices for human inspection. Such decomposition are far from really improving the efficiency of program understanding. To overcome this problem, Many alternative program slicing technique has been proposed, including dynamic slicing[1], conditioned slicing[6], constrained slicing, hybrid slicing[2], call-mark slicing[8] and so on. However, these approaches have serious efficiency problems in applying to large scale industrial programs. Another feasible solution to effectively and efficiently reduce slice size could be combining other light weight program decomposition approach with slicing. As scenarios correspond to requirements and modern software development processes are often scenario driven, the scenarios just provide an excellent decomposition technique for program understanding.

This paper studies the feasibility of incorporating UML scenario models into program slicing. Scenario used to refine the pointer/type information, and a scenario can be used to discard many unconcerned dependences by focusing on constrained execution paths.

In this approach, scenarios are specified through UML sequence diagram. A graphical scenario model will then be abstracted into more formal model says XML using visual paradigm. Scenario relevant codes are identified according to the model. By properly setting the concerned scenario, scenario oriented slicing can effectively reduce the slice size and provide a better support for program understanding.

The rest of this paper is organized as follows. Section 2 presents related works on program slicing. Section 3 introduces a XML from sequence diagram using a specified scenario. Section 4 merging developed java objects. Section 5 discuss how to identify scenario relevant codes from the main class. Finally we draw a conclusion of the paper.

II. PROGRAM SLICING

Program slicing is a technique for simplifying programs by focusing on selected aspects of semantics. The process of slicing deletes those parts of the program which can be determined to have no effect upon the semantics of interest. Program slicing finds program parts relevant to the computation of the variables in set V at program point P, where <P,V> is also called as Slicing criterion. Slicing has many application areas like testing and debugging, re-engineering, program comprehension and software measurement. Essentially, any area of software engineering and development in which it is helpful to extract subprograms based upon arbitrary semantic criteria are potential applications of slicing.

At first, the slicing was only static, i.e. applied on the source code with no other information than the
source code. Static slices are computed by finding consecutive sets of indirectly relevant statements, according to data and control dependencies. Then comes the dynamic slicing and many slicing technique like conditioned slicing, constraint slicing, hybrid slicing, call-mark slicing and Quasi-static slicing. Conditioned slicing and constrained slicing slice program under a constrained set of program inputs. Quasi-static slicing slices the program with some of its inputs fixed. Call-mark slicing records the executed procedure calls to refine static slicing. However, these techniques either highly depend on an executable system and representative test cases, or have serious efficiency problem, so we going for scenario oriented program slicing.

III. IDENTIFYING SCENARIO RELEVANT SLICES

Scenario are concerned with situations that might reasonably occur in the operations of the system. A scenario typically presented by a sequence diagram (or a collaboration diagram) in UML. The core element of a scenario is the message flow. The sequence diagram is easy for human inspection but not formal enough for automatic processing. To perform slicing, firstly we have to identify the codes relevant to a given scenario. This identification has three major steps. Firstly, For convenience, we abstract message flows from sequence diagram and converting it into XML, using JAXB. Then the XML is converted into Java objects, obtained java objects are merged together to create function call and main java object have to be created. Then based on the function call relevant slices are obtained.

A. XML code generation

The scenario is represented as sequence diagram, the generated sequence diagram is taken as input, which is then converted into valid XML code, which contains the functionality of the sequence diagram.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<item
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="item.xsd">
  <name>Clean room.</name>
  <priority>3</priority>
  <task>Pick up clothes.</task>
</item>
```

Fig:1 Sample XML Code

B. Main java file generation

JAXB (Java Architecture for XML Binding) combines the benefits of DOM (Document Object Model) parser and simple API (Application Programming Interface) for XML parser. JAXB provides a layer of abstraction between XML documents and Java applications. It converts data in XML elements into equivalent valid Java objects and set constraints with the help of XML schemas or DTDs (Document Type Definition).

XML schema says XSD (XML Schema Definition) has to be generated. Using unmarshalling process valid XML file is converted into java object that are instances of the classes. XSD has been created before converting XML into java objects, the XSD schema which is responsible for converting XML into java objects. Separate java objects have been generated for each class, as these classes won’t contain any function calls, these java objects cannot be taken as input to the slicing engine, since this is a functional decomposition technique, these objects won’t give relevant slices. So, the generated java objects are merged together, objects are created for each class. Using these generated object function calls are generated. Finally, merged main java object is created.

```xml
<item
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="item.xsd">
  <name>Clean room.</name>
  <priority>3</priority>
  <task>Pick up clothes.</task>
</item>
```

Fig:2 Overall Architecture

C. Relevant program slices

The merged main java object have to be given as input to the slicing engine. A slicing criterion have to be specified, based on the slicing criterion relevant slices are obtained. The slice will give the entire flow of that function call, name of any function call is given, the declaration of particular function and the declaration of any other function call which has called inside that particular declaration has also been extracted and displayed.
For example, let ‘A’ be a function call, slicing criterion is specified, slicing will be done based on the slicing criteria. Suppose let ‘B’ be another function called within A’s declaration, that particular function declaration will also be sliced. The final sliced code will be obtained like:

```java
public A()
{
    C();
    System.out.println("Function A");
}
public C()
{
}
```

Fig.3 Sample sliced code

IV. CONCLUSION

This paper proposes a scenario oriented program slicing method. It takes the user specified scenario and finds all the program parts relevant to a special computation under the given execution scenario. The whole slicing method takes three major steps. Firstly, the scenario input, an annotated sequence diagram, is parsed into XML. Then the relevant codes are identified. We conduct an experiment. The results show that scenario oriented slicing can effectively reduce the size of final slices when proper scenarios are specified. This focus reduction will provide a better support for scenario driven code inspection.

ACKNOWLEDGMENTS

We would like to thank the VIT University for providing their great support and guidance in completing this paper successfully.

REFERENCES


