CHAPTER III

TEST APPROACH

3.1 Overview

The test approach presented in this research focuses on test case generation from UML sequence diagrams. However, it also introduces a verification technique to be employed after the generated test cases are applied against the software under test. Therefore, the approach offers a complete cycle of test process. Figure 3.1 shows the overview of the approach.

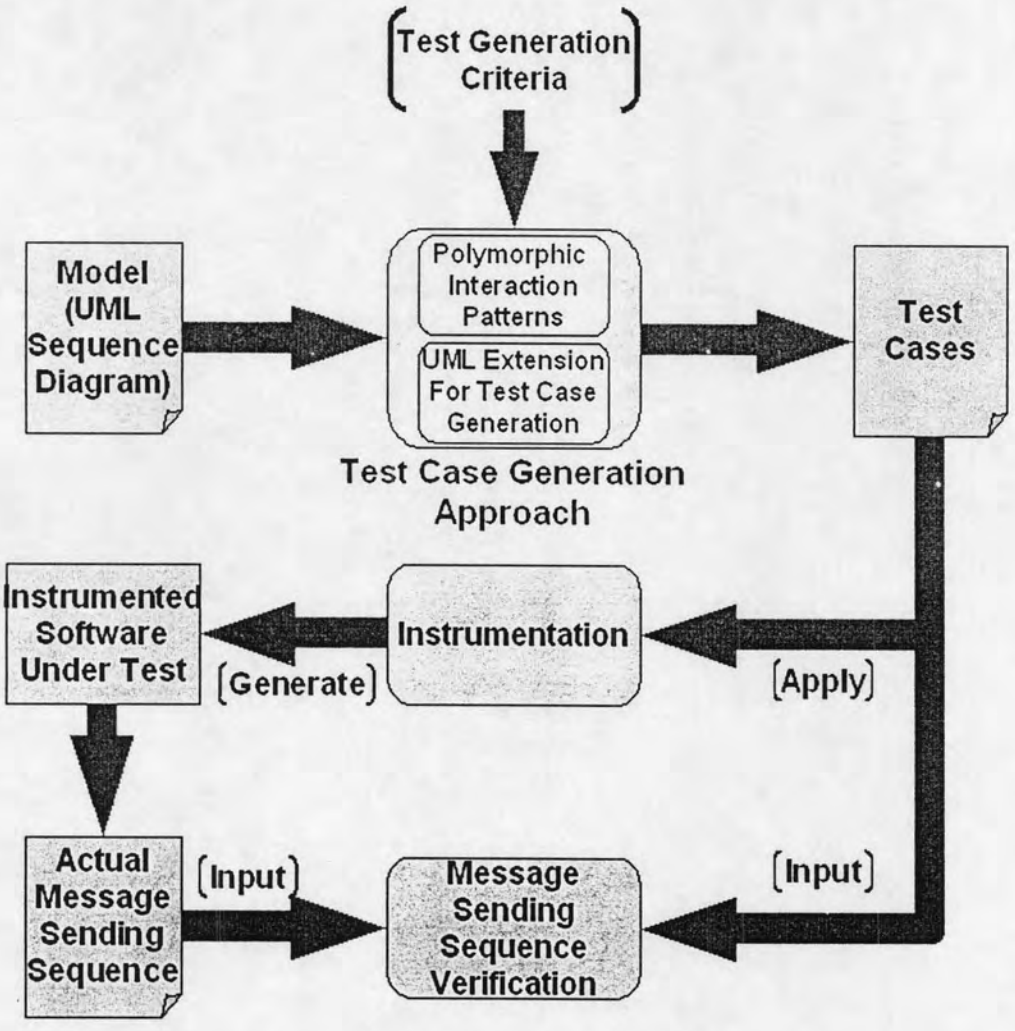

Figure 3.1 Overview of the approach

The test generation approaches takes models, UML sequence diagrams in particular, as its primary input. Human input is required for specifying a test adequacy criterion. Adequacy criteria identify necessary test coverage and also influence test case generation.

Next, test cases are generated based on a selected adequacy criterion. Since the approach focuses on testing polymorphic interactions, generated test cases are for executing polymorphic features of the interaction under test. Each generated test case comprises of a set of test input data and an expected message sending sequence.

The software under test is then expected to be executed against the generated test cases. The approach requires that the actual message sending sequence occurred during the test execution must be captured for verifying against the expected message sending sequence from the executed test case. The capturing is achieved through instrumentation. Instrumentation is not uncommon in software testing. The technique used for instrumentation in this research is discussed in detail in chapter 8 along with other implementation techniques.

The instrumented version of the software is used for test instead of the original version. The instrumented version performs just exactly the same as the original version, except that it also generates the actual message sending sequence or, in other words, the trace of method calls occurred during its execution. Finally the generated actual message sending sequence is verified against the expected message sending sequence to verify the result of test execution.

The subsequence sections discuss concepts in the approach.

3.2 Adequacy Criteria

Adequacy criteria are always required for software testing. Without adequacy criteria, it is impossible to tell whether a program is adequately and properly tested. Also adequacy criteria influence test case generation to a degree. To generate a set of test cases that are adequate for testing a particular program, an adequacy criterion must be defined. Test case generation approaches usually rely on one or more adequacy criterion.

The approach in this research is no exception. Since the approach is strictly related to polymorphism, test adequacy criteria, which focus on polymorphism, are required. Although there are adequacy criteria that focus on testing polymorphism feature such as ones defined by Alexander and Offutt [31], none of them is appropriate for the approach in this research. The approach is for testing polymorphic interactions; therefore, adequacy criteria specifically for testing polymorphic interactions are required.

In a polymorphic interaction, it is important that whenever a polymorphic call is encountered, one should be aware that the subclasses of the target of the call should be tested as well. However, testing every subclass for every polymorphic call could be very tedious. A simple interaction in a sequence diagram could possibly end up with hundreds of test cases. Human decision may be required to compromise the adequacy criterion, in order to meet practical test execution requirement. Therefore, different adequacy criteria are required for different levels of test coverage. To test a polymorphic interaction, one of the adequacy criteria presented in this research must be selected. For a particular polymorphic interaction, the selected adequacy criterion tells how many test scenarios are required and which classes to be tested for each scenario. Polymorphism related adequacy criteria proposed in this research are discussed in detail in chapter 4.

3.3 Polymorphic Interaction Pattern

After an adequacy criterion is applied to a certain polymorphic interaction to identify test scenarios, a test case is generated for each test scenario. While a test scenario states which class to be tested for each polymorphic entity in a polymorphic interaction, a test case states how to achieve such the scenario. To have a particular class tested in a polymorphic interaction, an object of the class is assigned to the corresponding polymorphic entity. Such assignment has to be done through a polymorphic assignment; as a result, the polymorphic assignment for each polymorphic entity must be located. However, from an interaction diagram it is usually not obvious where exactly each polymorphic assignment is.

This research proposes a method to locate a polymorphic assignment through the use of polymorphic interaction patterns. The patterns are derived from the observation about polymorphic assignments in polymorphic interactions. A polymorphic interaction has a characteristic, which classifies itself as one of the three polymorphic interaction patterns. For each pattern, how to locate a polymorphic assignment in a polymorphic interaction is defined. Moreover, each pattern also states how to control the polymorphic assignment to get an object of a particular class assigned to the polymorphic entity. This concept is very necessary for test case generation for a polymorphic interaction for a specific test scenario where a particular set of classes must be tested.

The polymorphic interaction patterns are the basis for test case generation in the research. In chapter 5, the patterns are explained in full detail along with basic examples of how test cases are generated from interaction diagrams of known patterns.

3.4 Test Result Verification

As stated earlier, there are two aspects of software testing: fault-directed testing and conformance-directed testing. The approach in this research in this research is conformance-directed testing. The major purpose of the approach is to ensure that a program is implemented accordingly to its UML sequence diagrams. Strictly speaking, a program must be implemented in a way that its object interaction during execution is equivalent to the interaction defined in its sequence diagrams. It is obvious that there is no specific fault model for the approach.

In conformance-directed testing, verification of test result must reflect conformance or the lack of conformance of implementation against its specification. The verification of test result for the approach cannot solely rely on comparison of the execution output values against the expected output value, since it does not exactly demonstrate conformance of the program under test against its design specification, in this case its UML sequence diagram. A UML sequence diagram specifies an object interaction in a form of a sequence of messages which objects in the interaction send to each other; consequently, conformance-directed testing based on a UML sequence diagram must ensure that the implementation of the diagram has the same sequence of

messages, which are usually implemented as operation calls between objects. In other words, test result verification must ensure that operation calls during execution occurred as defined in the sequence diagrams. The verification then needs comparison of the message sending sequence in a UML sequence diagram against the message sending sequence from the execution of the implementation of the diagram.

This can be achieved through capturing message sending sequence from both UML sequence diagrams and execution of the implementation. From UML sequence diagrams, an expected message sending sequence is extracted to represent the sequence of operation calls which the implementation must follow. From implementation, an actual message sending sequence is captured from the execution through instrumentation of the implementation. The instrumentation process alters the implementation to create an instrumented version of the implementation. In the instrumented version, operation calls occurred during execution are captured as an actual message sending sequence.

This research proposed a model for representing message sending sequence, which can represent both expected message sending sequence and actual message sending sequence. Also a procedure for verification of message sending sequence is presented. In the approach, the verification of message sending sequence is verification of test result. Although verification of message sending sequence may seem simple, it is not just a simple comparison of message sending sequences due to some issues. The model for representing message sending sequence and the verification procedure are discussed in detail later in chapter 6.

3.5 Test Case Generation

In general software testing, a test case essentially includes a set of test inputs and an expected test result. In this approach, an expected test result is in a form of an expected message sending sequence as stated in the previous subsection. Test case generation in the approach must then take into account an expected message sending sequence of each test scenario it generates a test case for.

However, how test inputs can be generated is not yet discussed so far. For a given polymorphic interaction, the polymorphic interaction patterns only tell how to

control each polymorphic assignment in the interaction. Test inputs are generated from that fact. However, identifying a polymorphic interaction as one of the polymorphic interaction patterns does not automatically reveal all information necessary for test case generation. Strictly speaking, additional information is required for test case generation. This information is not readily available on usual UML sequence diagrams.

In chapter 7, an extension to UML sequence diagrams is presented. The purpose of the extension is for providing the additional information necessary for test case generation in this approach. Also test case generation based on the extension are discussed in full detail in chapter 7.

From conceptual view, there are several steps in test case generation as shown in figure 3.2. Note that implementation of the approach may include more additional steps, which are implementation dependent.

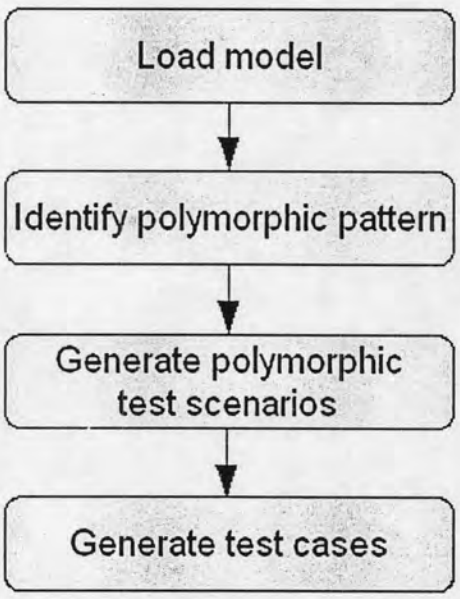

Figure 3.2 Steps in test case generation

Test case generation starts with model loading. The model may include UML sequence diagrams and other related information sources, e.g. class diagrams. An interaction from each sequence diagram is determined whether it is a polymorphic interaction. In order to do so, it is checked against each form of the polymorphic

interaction patterns. If the interaction is polymorphic, test scenarios are generated according to a selected polymorphic adequacy criterion. Each test scenario includes an expected message sending sequence and a set of test input criteria. After this step, actual test input are generated from the test input criteria.

In chapter 8, a tool implemented based on this concept is demonstrated to show the capability of the approach against a number of examples of UML sequence diagrams.