CHAPTER IX

CONCLUSION

9.1 Research Result

This research proposes an approach for generating test cases from interaction diagrams, UML sequence diagrams in particular. The approach focuses on generating test cases for polymorphic interactions. Since a polymorphic interaction may have its behavior during runtime different from what is stated in its interaction diagram, the test approach identifies how to cover test scenarios necessary to demonstrate correctness of the interaction under test. In other words, a polymorphic entity may be substituted with an object of one of subclasses of the class defined for the polymorphic entity. The approach concerns testing of the interaction against those subclasses. The findings from the research are listed as followed.

1. To generate test cases, an adequacy criterion is required to be defined. The approach defines 5 adequacy criteria for testing polymorphic interactions. The adequacy criteria vary in coverage. They range from Base class coverage, which completely ignores polymorphism issue, to Strong inheritance coverage, which requires all subclasses of all polymorphic entities to be tested in every possible combination. In the evaluation of the research, Base class coverage is mostly left out, since it is not exactly relevant to polymorphism. The usage of the adequacy criteria are demonstrated in chapter 4 and later in chapter 8, where a tool is implemented based on the adequacy criteria. In chapter 8, test cases are generated based on the adequacy criteria. The subsumption hierarchy shown in chapter 4 is confirmed with the results from chapter 8.

2. In order to complete test operation, the approach also covers verification of the test execution. A model for message sending sequence is designed for representing message sending sequence both specified in interaction diagrams and occurred during test execution. A program under test is instrumented so that the message sending sequence is captured. The captured message

sending sequence is verified against the expected message sending sequence from the interaction diagram to identify whether the test execution results in the expected manner. The verification procedure is presented in chapter 6, and in chapter 8 it is implemented by the tool. The result from the evaluation of the tool shows that the procedure can tell the difference between correct and incorrect implementation.

3. To generate test cases for a polymorphic interaction, polymorphic assignments in the interaction must be identified, and their behavior must be known. In chapter 5, three patterns of polymorphic assignment are presented. The patterns help in defining how to identify and control a polymorphic assignment so that the polymorphic entity is substituted with an object of a particular subclass for test. The three patterns are basis of test case generation.

4. The patterns in chapter 5 are defined based on general interaction diagrams. However, the information from the diagrams alone is not sufficient for test case generation. More information, e.g. the condition which an object of a particular subclass is assigned to the polymorphic entity, is required. How to specify such information is defined in chapter 7. An extension to UML sequence diagrams for specifying this type of information is presented in the chapter. The extension is in a form of tagged values which are standard elements in UML semantic. The necessary information must be supplied through these tagged values in order to have a sequence diagram ready for test case generation. Later in chapter 7, the test case generation procedure is presented. The procedure primarily relies on the information from the tagged values defined in the extension for generating test cases and test data.

In chapter 8, a tool is implemented based on the concepts in the previous chapters. The tool is evaluated against a number of sequence diagrams which contain the extension as described in chapter 7. It is demonstrated that the tool is capable of generating test cases for sequence diagrams with any of the patterns of polymorphic assignment. Generating test cases for a polymorphic interaction with more than one

polymorphic assignment is also possible. The generated test cases are also used for test execution. The actual message sending sequence is captured from the execution and verified against the expected message sending sequence of the corresponding test case. The implementation of the tool shows that the approach presented in the research is possible, reasonable, and reliable. Although the tool is not implemented to cover all aspects of test case generation regarding polymorphic interactions, it shows that the test approach is capable of generating test cases of interaction diagrams in known patterns, and the approach is systematic enough that the test case generation process is automatic.

9.2 Limitations

There are two limitations regarding the test approach.

1. The patterns of polymorphic assignment may not cover all cases in real world. Although the patterns cover all essential behaviors of polymorphic assignment, they do not cover all possible variations of the patterns. A pattern may have more than one possible form as shown in the discussion section of chapter 5. Unless all variations of patterns are identified, human judgment is required in this case. In addition, the extension of UML sequence diagram may be affected by the variations as well.
2. The test cases generated by the test approach are not themselves executable due to some difficulties.

There are two major obstacles that make generation of executable test cases difficult, if not impossible. First, it is usually the case that the generated test data are not sufficient for test execution. Although, the approach supports using test data generation techniques, there are cases where exact test data are not possible to be generated. For example, an interaction, which contains Simple polymorphic assignment, usually results in test cases, which specifies what class of an object to be supplied as an input of the interaction. This is simply not sufficient for test execution, since how to obtain an object of such class is not specified. An object of the class may be instantiated through one of

its constructor, obtained from a factory method, or cloned from a prototype object. Moreover, the object may be required to be in a particular state, but how to set the object to the state is practically unknown. This type of information is not supplied from and not directly relevant to interaction diagrams; therefore, test case generation based on only interaction diagrams is not the solution to this problem.

The other issue regarding test execution is implementation of test driver or test code. Although this issue is not as severe as the previous one, it requires a great deal of effort to define test driver generation. Since it is implementation specific, issues in certain programming languages and platforms must be taken into account. In addition, different architectures may require different ways of testing. If executable test cases are to be achieved, a very strict scope must be defined. However, executable test cases are out of the scope of the research.

9.3 Future Works

The research may be taken further to solve issues not currently covered. Here are ideas that may be explored further.

1. More variants of patterns of polymorphic assignment may be discovered. The variants may not have significant effects on the test approach, since test case generation relies on the information from the patterns, not the forms. Nevertheless, the variants may have a great effect on the tool implementation. If the discovered variants are comprehensive enough, it is possible to build a tool that can systematically generate test cases for real world problems with very little human effort. This is very desirable in software testing domain where most of operations require a lot of human effort.
2. Test execution could be made possible. Rather than stop at generating test cases for human, executable test cases may be generated instead. This will save a lot of effort used in writing a test driver for each test case. As stated in the previous subsection, this requires quite a lot of effort in defining implementation specific techniques. A particular set of programming languages, platforms, and architectures to be supported must be strictly defined. However, the issue of object creation and object state setting are the major obstacles. Additional

information is required possibly in a different form, e.g. from another diagram. Moreover, additional sophisticated test data generation techniques may also be required to solve test data generation issues.

3. It is interesting to see the test approach being applied to other interaction diagrams, for example UML collaboration diagram. Since different diagrams have different aspects in modeling a system, applying the test approach to other diagrams may show new aspects of information for test case generation and may uncover a new way of test case generation.