Test Case Generation Technique for Interoperability Test of Component Based Software from State Transition Model

Wan-Seob Byoun , Cheol-Jung Yoo† , Hye-min Noh and Ok-Bae Chang

Chonbuk National University, Jeonju, Jeonbuk, South KOREA

Summary
With the rapid growth of CBD technology, two or more components from different vendors are integrated and interact with each other to perform a certain function in the component-based software. The main interest of component users who develop applications using components developed already, is to confirm that the component is collaborating with other components according to the requirements. Interoperability test is to check how those components collaborate each other in component user context not in the component development context. Therefore, the research on the interoperability is crucially important. In this paper, a new technique is proposed for generating a test case which is necessary for checking the interoperability of the components in the component-based software.

Key words:
Testing, CBD, Test Case Generation, Interoperability Test

1. Introductions

The main advantage of Component Based Development(CBD) is that it is possible to reuse the high-quality components provided by professional component vendors. The methodology of CBD is steadily growing in the area of software development. Many different methods are being proposed by the CBD users[1].

The main interest of component users who develop applications using components developed already, is to confirm that the component is collaborating with other components according to the requirements and that the component is unified into new framework after completing the application software. In the component-based software, it is necessary to check how those components are integrated into one complete software and collaborate each other in real environment not in the development environment. Therefore, research on the inter-operability of each component in a CBD software is crucially important[2].

However, research on the method of creating a systematic and optimized test process and test suit checking the inter-operability of the components in the CBD software is still in the primitive stage[3]. In common CBD software, it is often impossible to detect the internal errors arising from the incomplete test case. Also the cost of the CBD software can be wasted by employing duplicate test case inadvertently[3].

In this paper, a new technique is proposed for generating a test case which is necessary for checking the inter-operability of the components in the component-based software. The technique is using the state transition model previously proposed by Noh et al.[5]

The structure of this paper is as following. Following the introduction (Chapter 1), descriptions about the previous test methods of CBD software are presented in chapter 2. In chapter 3, a method of creating of test case is proposed based on the specification of the Extended Finite State Machines (EFSM) model. Then an example of application of the method to a real CBD software is given in chapter 4, followed by the results of analyses of the proposed method in chapter 5, and finally, in chapter 6, the conclusion and future research.

2. Test of Component Based Software

Software engineers are aware of the difference in validation process of the component-based software and that of the software developed in a traditional manner[4,5]. The test of the CBD software include both the providers and the users of the components, the former develop the components and the latter realize the application software using the components[6].

The usual test performed by the providers is concentrated on checking accurately the performance of the internal logic, data, and program structure of the components. Therefore, the test by the providers is usually done in the method of white-box test[1]. On the other hand, the test executed by the users of the components is to evaluate the adequateness of each component in the framework of the application software. The user's test normally done in the method of black-box test, which regards the component as a black-box[1].

Following are brief lists of important factors that hinder the test of the CBD software[7][8][9].

• *Test of components in the new environment*
Components are developed in the development environment of the providers, but they are reused by the

This work was supported by the Korea Research Foundation Grant funded by the Korean Government (MOEHRD)"(KRF-2006-105759001).
† corresponding author

users in a completely different environment. The user's environment is sometimes beyond the expectation of the providers so that validation of the inter-operability between the components is regarded as crucially important in testing the CBD software.

• *Difficulty in access to internal information of each component*

Another problem is the low visibility of each component which is subjected to the test. The internal information except the interface of each component is strictly limited. Systematic method of testing the inter-operability of components should take into account such problem of low visibility of each component.

3. Generation of Test Case for Interoperability Test of Components

3.1 Test model

Test models are used as input for producing a test case. Hao defined the state transition specification of EFSM[10]][11][12]. We specify EFSM in the XML DTD format as follows.

```
<!-- === Start Entity Declaration === -->
<!-- === End Entity Declaration === -->

<!-- === Start Element Declaration === -->
<!ELEMENT EFSM (transition)+>
<!ELEMENT transition (from , trans_info , to)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT trans_info (input+ , output+ ,
 predicate+ , action+ ,color )>
<!ELEMENT input (#PCDATA)>
<!ELEMENT output (#PCDATA)>
<!ELEMENT action (#PCDATA)>
<!ELEMENT predicate (#PCDATA)>
<!ELEMENT to (#PCDATA)>
<!-- === End Element Declaration === -->
<!-- === Start Attribute Declaration === -->
<!ATTLIST EFSM      e_id ID #REQUIRED>
<!ATTLIST transition id ID #REQUIRED>
<!-- === End Attribute Declaration === -->
```

We define the elements constituting the EFSM as follows[3].

• State : ‘State’ in state identification table
• Input : external input inducing the behavior identified in use case specificaton
• Output : Response of the system as the result of behavior
• Predicates : Logical expression of the component attributes obeyed after the transition
• Action : behavior to produce the transition
• Color : indicator that the state transition brings the inter-operability or not.

3.2 The concept and process for generating the test case

The purpose of producing the test case proposed in this paper is to create a complete and optimized test case to check the inter-operability between the components. Fig. 1 represents the steps of creating a test case for checking the inter-operability between the components, using the EFSM presented in chapter 3.

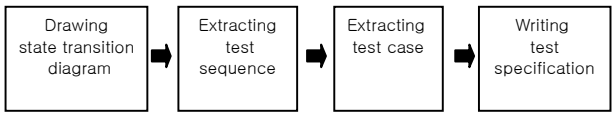

Fig. 1 Steps for generating a test case

3.3 Test cases generation technique

3.3.1 Drawing of a state transition diagram using the EFSM

The EFSM proposed by Noh et al. include information of the state and state transition of the component based software identified in the process of behavior modeling. In addition, it contains the information of the inter-operability of the components. The states are sorted as 'Idle' and 'Normal' states, and the transitions are differentiated as 'White' edge inter-line and 'Black' edge inter-line, in order to exclude unnecessary tests that are not related with the inter-operability. The definition of 'Idle' state and 'Normal' state is as following.

[Definition 1] *'Idle' state is a dormant state without any behavior, waiting for events to occur. 'Normal' state is a state that the system is performing an behavior because of changes in conditions or occurrence of events to the system.*

'White' edge inter-line and 'Black' edge inter-line are defined as following.

[Definition 2] *'White' edge inter-line represents the state transition that is not related with the inter-operability between the two components. 'Black' edge inter-line, to the contrary, represents the state transition that is related with the inter-operability.*

The rules for making the state transition diagram are as following.

[Rule 1] *Transitions between the states are described in the standard notation according to the contents in the 'From-state' items and 'To-state' items of the EFSM specification. The attribute values of components corresponding each state are analyzed and depicted as 'Idle' state and the 'Normal' state depending on the values. The 'White' edge inter-lines and the 'Black' edge inter-lines are differentiated depending on the values of the 'Color' index.*

Fig. 2 shows an example of the EFSM specification, and Fig. 3 shows the state transition diagram of the EFSM specification using the rule 1 above.

```
<transition id = "T1">
  <from>S1</from>
  <trans_info>
    <input>String name; String password</input>
    <output></output>
    <predicates>filledLoginInfo_C1=true</predicates>
    <actions>
      <act1>input 'name' field and 'password' field</act1>
    </actions>
    <color>White</color>
  </trans_info>
  <to>S2</to>
</transition>
<transition id = "T2">
  <from>S1</from>
  <trans_info>
    <input></input>
    <output>Display Error Message;
            Load Registration Form</output>
    <predicates>A_authSuccess=false;
                filledLoginInfo_C1=false;
                clickedSubmit_C1=true;
                reqCustInfoToC2_C1=true;
                checkIdPsw_C1=true;
                dispRegForm_C1=true;
                dispErrMsg_C1=true</predicates>
    <actions>
      <act1>Click 'Submit'Button</act1>
    </actions>
    <color>Black</color>
  </trans_info>
  <to>S5</to>
</transition>
<transition id = "T3">
  <from>S2</from>
  <trans_info>
    <input>String name; String password</input>
    <output>Create Shopping Cart; Display
            'Web Store'Page</output>
    <predicates>A_authSuccess=true;
                filledLoginInfo_C1=true;
                clickedSubmit_C1=true;
                reqCustInfoToC2_C1=true;
                checkIdPsw_C1=true;
                sendCustInfoToC1_C2=true;
            createCart_C3</predicates>
    <actions>
      <act1>Click 'Submit' Button</act1>
    </actions>
    <color>Black</color>
  </trans_info>
  <to>S3</to>
</transition>
```

Fig. 2 An example of the EFSM specification

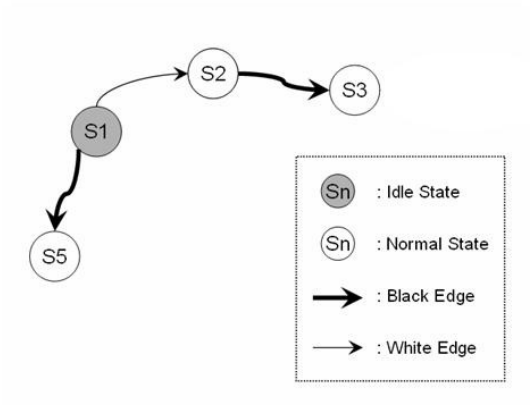

Fig. 3 An example of a state transition diagram

3.3.2 Extracting a test sequence using the state transition diagram

Most important step in extracting a test sequence is to find the acyclic path from the state transition diagram. The rule of finding acyclic path from the state transition diagram which contains cycles is as following. Fig. 4 shows the process of finding strongly connected components (SCC).

[Rule 2] *Generate all possible acyclic paths without repeating apex, and search all the strongly-connected components (SCC) that have maximal bi-directional paths between all the pairs of apexes.*

[Rule 3] *Generate sets of final paths by combining the SCC into the acyclic paths generated by the[rule 2].*

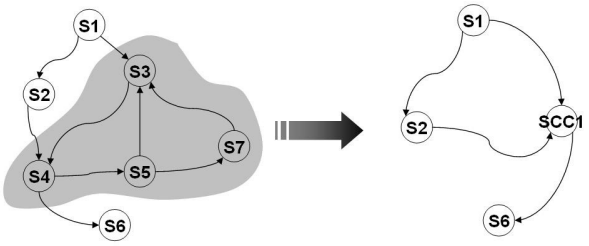

Fig. 4 Reachability Graph and its Corresponding DAG

Algorithm for generating a test sequence

Hao et al. proposed an algorithm for generating a test sequence which contains only the factors of inter-operability from the state transition diagram[10]. In this paper, advancing the Hao's algorithm, we propose a new algorithm to generate a test sequence for testing the inter-operability, using the information contained in the EFSM and the graphic properties in the state transition diagram as described above. Fig. 5 depicts the algorithm for extracting a test sequence using the [Rule 2] ~ [Rule 3], excluding the portion from the diagram that is not related with the inter-operability. The super edge inter-line described in the algorithm is defined as following.

> ***[Definition 6]*** *The super edge inter-line is the edge line directly connecting the root node to the first node with a black edge inter-line.*

The basis for excluding the states and the edges that are not related with the inter-operability is whether or not the transition between the states of the components contains the inter-operation, and whether or not it affects the previous transition or the following transition related with the inter-operability. The rule described in the algorithm for excluding the states and the edges irrelevant with the inter-operability is as following.

> ***[Rule 4]*** *Replace all the white edges that connect the root node and the first node with a starting black edge by the super edges. Exclude all the ending white edges connecting to the idle nodes. Exclude all the ending white edges connecting to the terminal nodes.*

White edges are representing the state transition without the inter-operability in the state transition diagram. They should be excluded from the analyses, but they should be included as one transition since the *transition* from the root node to the node with first inter-operability must be taken into account. This process is described in the 1st ~ 4th rows of the algorithm. Idle nodes are representing the states that the system is waiting for events with no action applied to. White transitions to the idle states are irrelevant with the inter-operability between components, so that the edges corresponding the white transitions are excluded from the diagram. This is described in the 5th row of the algorithm. After applying the above process, there occur terminal nodes. White transitions to the terminal nodes are eliminated from the diagram since they are irrelevant with the inter-operability. This is presented as the 6th row of the algorithm. The 7th row of the algorithm describes the process to create all the acyclic paths by applying the [Rule 2] ~[Rule 3] to the state transition diagram produced by applying the [Rule 4]. All the acyclic paths thus created become the the test sequence.

Test sequence generation algorithm

from the state transition diagram,
1 from the node v with starting black edges
2 to all the white edges from the root node to node v,
3 add super edge from the root node to node v.
4 eliminate all the starting white edges from the root node to node v, except the super edges.
5 eliminate all the ending white edges to idle nodes.
6 eliminate all the ending white edges to the terminal nodes
7 create all the acyclic paths using the algorithm to search the acyclic paths

Fig. 5 Algorithm creating a test sequence

Fig. 6 shows the process for generating a test sequence by applying the algorithm described above to the diagram shown in Fig. 4.

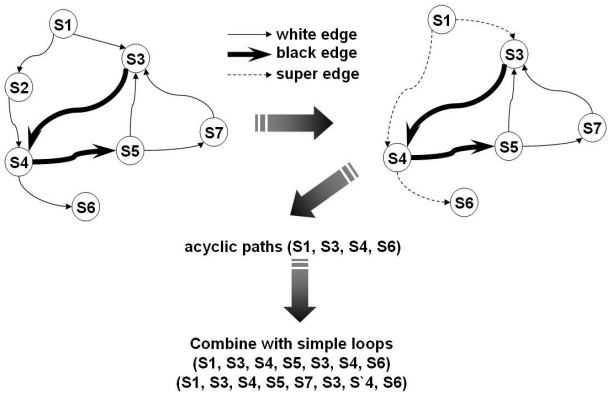

Fig. 6 Generation of test sequence using the algorithm described in Fig. 5.

3.3.3 Test case specification

The test case specification usually contains the information in Table 1. [13]

Table 1: Test case information

Test case specification identifier	Specify the unique identifier assigned to this test case specification
Test items	Identify and briefly describe the items and features to be exercised by this test case
Input specification	Specify each input required to execute the test case
Output specification	Specify all of the outputs and features required of the test items
Environment needs	Specify all of the hardware and software constraint of the test items
Special procedural requirements	Describe any special constraints on the test procedure that execute this test case
Intercase dependencies	List the identifier of test cases that must be executed prior to this test case

4. Example of Application

4.1 Extracting the test sequence

Fig. 7 depicts the state transition diagram based on the EFSM.

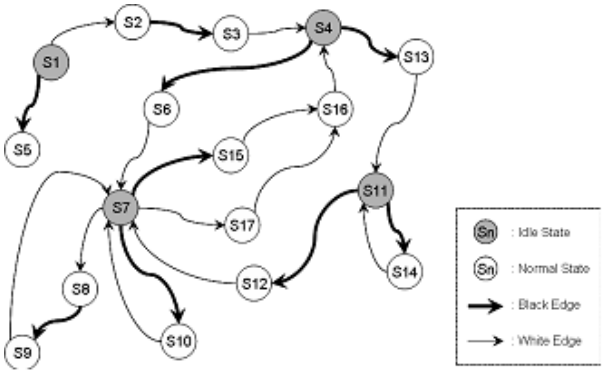

Fig. 7 state transition diagram

Fig. 8 represents the results of executing the algorithm from the 1st to the 5th row to generate a test sequence.

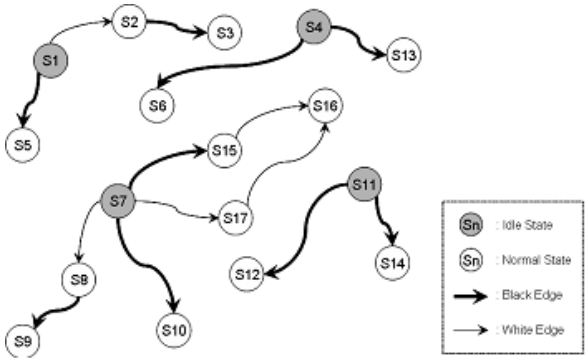

Fig. 8 eliminating the ending white edges to the idle nodes.

Fig. 9 is depicting the results of performing the 6th row of the algorithm generating a test sequence.

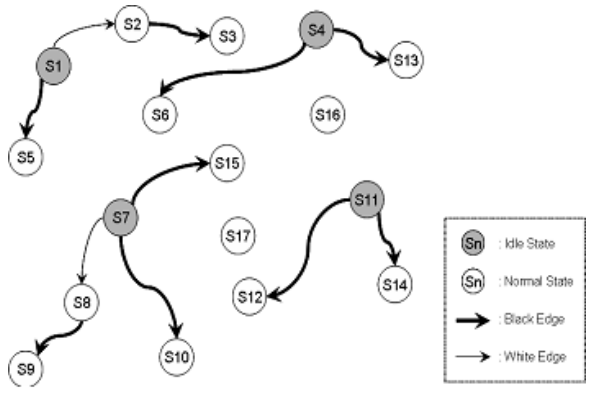

Fig. 9 eliminating all the ending white edges connecting to the terminal nodes.

The results of test sequence produced by executing the 7th to 9th row in the test sequence generation algorithm are as following;

S1 → S2 → S3
S1 → S5
S4 → S6
S4 → S13
S7 → S8 → S9
S7 → S10
S7 → S15
S11 → S12
S11 → S14

4.2 Generating a state transition table for test cases

Table 2 is partial list of state transition table for generating test cases based on the test sequence produced as above.

Table 2 : example of state transition table

No	*From State*	*Precondition*	*Input*	*Postcondition*	*To State*
T01	S1	dispLoginPage Idel_C1=true	String name String password	filledLoginInfo_C 1=true	S2
T02	S2	filledLoginInf o_C1=true	String name String password	A_authSuccess=tr ue filledLoginInfo_C 1=true clickedSubmit_C1 =true reqCustInfoToC2_ C1=true checkIdPsw_C1=t rue sendCustInfoToC1 _C2=true createCart_C3	S3
…	…	…	…	…	…

The test cases are produced based on the state transition table of Table 2 and the specification of EFSM. Table 3 shows the test case specification generated by the 'T01' state transition.

Table 3 : specification of a test case

Test Case Identifier	T01	
State Transition	S1 → S2	
Test Item	Use case ID.	U001. User Authentication
	Interoperability	none
	Behavior	input customer's id and password in id and password field
Input	String name String password	
Output	Nothing	
Procedural requirements	Customer must located in log-in page	
Intercase dependencies	Nothing	

5. Results and Discussions

In this paper, we define the quality of the test case from the completeness and optimization of the test case. The completeness of the test case is the measure of ability of detecting errors, and the optimization is representing how much the irrelevant test cases are excluded from the test case.

First of all, an example of application software is analyzed and the constituting components and the methods related to the inter-operability are identified. The completeness of the test case proposed in this paper is measured by the number of components and methods identified. The number of transition in the state transition diagram are compared before and after applying the test sequence generation algorithm. This indicates how the irrelevant test cases are excluded in checking the inter-operability of the components. The results are given in Table 4.

Table 4 : Completeness and optimization a 1

Completeness	Number of methods	Number of method contained in the test case
	37	35
Optimization	Number of transitions before applying the algorithm	Number of transitions after applying the algorithm
	22	11

Table 4 shows that the test case generated by using the method proposed in this paper scores high both in the completeness and optimization. Therefore, the method of generating a test case proposed in the paper seems to be a useful tool in the evaluation of CBD software.

6. Conclusions and Future Works

In this paper, a new method of creating a test case for checking the inter-operability of the components in CBD software. The Extended Finite State Machines (EFSM) model which contains the information of the inter-operability is used in generating for the test case. The test case thus created is proved to be effective in testing the inter-operability of the components.

The significance of the method of creating the test case proposed in this paper is as following:

First, the low visibility problem of a component of incomplete information can be overcome. This is done by performing a black-box type test of the CBD software, by comparing the resultant outputs and the expected outputs corresponding to various inputs.

Second, it is often possible to generate a duplicated and incomplete test case if ordinary test models are used in testing the inter-operability. To overcome such a problem, in this paper, we proposed a new method of generating a test case that test only the inter-operability between the components.

Third, an algorithm is presented to produce a test sequence that can eliminate the duplicated test cases by using the test models with the information related with the inter-operability. In this paper, EFSM is used as the input to generate a test case for checking the inter-operability. Test models play crucial roles in extracting high-quality test cases. Therefore, creating a test model that are formal and complete is important in generating a successful test case[14][15]. Further studies are necessary to define and develop test models that are suitable for the purpose of each test.

References

[1] Hans-Gerhard Gross, Component-Based Software Testing with UML, Springer, 2004

[2] J. Clark, C. Clarke, S. DePanfilis, G. Granatella, P. Predonzani, A. Sillitti, G. Succi, and T. Vernazza. "Selecting Components in Large COTS Repositories". Journal of Systems and Software, 2005.

[3] Hye-Min Noh, Ji-Hyun Lee, Cheol-Jung Yoo, and Ok-Bae Chang. "Behavior Modeling Technique Based on EFSM for Interoperability Testing", ICCSA 2005, LNCS 3482. pp. 878-885, 2005.

[4] E.J. Weyuker. Testing Component-Based Software: A Cautionary Tale". IEEE Software, 1998.

[5] E.J. Weyuker. The Trouble with Testing Components,. In Component-Based Software Engineering, Heineman/Councill (Eds). Addison-Wesley, 2001.

[6] Crnkovic, I. "Component-Based Software Engineering for Embedded Systems", ICSE 2005 Proceedings, pp. 712-713, 2005.

[7] J. Gao. "Challenges and Problems in Testing Software Components". In Workshop on Component-Based Software Engineering(ICSE 2000), Limerick, June 2000.

[8] J. Z. Gao. H.-S.J. Tsao, and Y. Wu. Testing and Quality Assurance for Component-Based Software Engineering, Artech House, 2003.

[9] G.T. Heinman and W.T. Councill (Eds), Component-Based Software Engineering. Adisson-Wesley, Boston, 2001.

[10] R. Hao, D Lee, R. K. Sinha, N. Griffeth, "Integrated System Interoperability Testing with Applications to VoIP", IEEE/ACM Transactions on Networking, Vol. 12, Issue 5, pp. 23-836, 2004

[11] N. Griffeth, R. Hao, D. Lee, R. K. Sinha, "Interoperability Testing of VoIP Systems", Global Telecommunications Conference, Vol. 3, pp. 1565-1570, 2000.

[12] R. Hao. "Protocol Conformance and Interoperability Testing Based on Formal Methods". PhD Thesis, Tsinghua University, P.P.China, 1997.

[13] IEEE. IEEE Standard for Software Test Documentation, IEE Std 829. 2000.

[14] D. Lee, M. Yannakakis, "Principles and Methods of Testing Finite State Machines-A Survey", Proceedings of the IEEE, Vol. 84, Issue 8, pp. 1090-1123, 1996

[15] MIAO Hauikou and LIU Ling, "A Test Class Framework for Generating Test Cases from Z Specifications", 6th IEEE International Conference on Complex Computer Systems(ICECCS'00), Tokyo, Japan. pp. 164-171, 2000.

Wan-Seob Byoun received B.S. degree in Mathematics Education in 1990, and M.S. degree in Computer Science in 1996 from Chonbuk National University, Jeonju, South Korea. During 1997-2001, he was a full-time PhD student of the Software Engineering Laboratory, Chonbuk National University. Since then, he has been studying the interoperability test of component-based software. He is now a commissioner special of computer education in Chonbuk Educational District of Korea.

Cheol-Jung Yoo received the B.S. degree in Computer and Statistics from Chonbuk National University, Jeonju, Korea, in 1982, M.S. degree in Computer and Statistics from Chonnam National University, Kwangju, Korea, in 1985, and Ph.D. degree in Computer and Statistics from Chonbuk National University, Jeonju, Korea, in 1994. He is currently an associate Professor, Department of Computer Science, Chonbuk National University, Jeonju, Korea. His research interests are software development process, software quality, component software, software metrics, software agent, GNSS, GIS, education engineering, and cognitive science etc.

Hye-Min Noh received the B.S., M.S and Ph. D. degrees in Computer Science from Chonbuk National University, Jeonju, Korea in 2000, 2002 and 2006. His research interests are component based software development, formal method, embedded software architecture, software testing, software measurement and software development process etc.

Ok-Bae Chang received the B.S. and M.S. degrees from Korea University, Seoul, Korea in 1973 and Ph. D. degree from UC Santa Barbara, in 1988. He is currently an Professor, Division of Electronic and Information, Chonbuk National University, Jeonju, Korea. His research interests are numerical Analysis, software development process, software quality, software metrics, education engineering and discrete mathematics etc.