MINISTRY OF EDUCATION AND TRAINING

THE UNIVERSITY OF DANANG

LÊ THANH LONG

AUTOMATIC TESTING OF INTERACTIVE MULTIMODAL APPLICATIONS

Major: Computer Science

Code: 62 48 01 01

DOCTORAL DISSERTATION

Danang 2017

The doctoral dissertation has been finished at:

THE UNIVERSITY OF DANANG

Supervisor: Prof, Dr. IOANNIS PARISSIS

Assoc. Prof. Dr. NGUYỄN THANH BÌNH

Reviewer 1: ……

Reviewer 2: ……

Reviewer 3: ……

The dissertation is defended before The Assessment Committee at The University of Danang.

Time: …..... h .....

Date: ...../……/…….

The dissertation is available at:

- National Library of Vietnam
- Learning & Information Resources Center, The University of Danang

INTRODUCTION

1. Motivation

Interactive Multimodal Applications (IMAs) support communication with the user through different modalities such as voice and gesture. They can greatly improve human-computer interaction, because they can be more intuitive, natural, efficient, and robust. The flexibility and robustness of IMA result in an increasing complexity of the design, development and testing. Therefore, ensuring their correctness requires thorough validation.

Approaches based on formal specifications automating the development and the validation activities have been proposed to deal with this complexity. In [21], Laya Madani et *al.* present a technique of test case generation for testing CARE properties by means of a synchronous approach. According to the proposed approach, CARE properties are translated into an enhanced version of the Lustre synchronous language. An improved method presented in [22] uses task trees and a fusion model to perform test data generation for IMAs.

The above presented approach uses several notations, inspired fom existing modeling languages, to build test models : *a model of the application behavior, a model of the interactive tasks, operational profiles (annotations on CTT)* and *modality specifications.* The variety of notations makes the modeling process hard. So, as an additional improvement to this previous research work, in this thesis “Automatic Testing of Interactive Multimodal Applications”, our objective is to define a single specification and modeling language, called TTT (Task Tree based Test) making possible to express, in a single and consistent syntax.

We built an automatic test generation approach based on this test modeling language. The approach allows specifying multimodal events of IMA and CARE properties as well as checking the validity of CARE properties. We also develop a tool that automates the test of such IMA.

2. Main Contributions of the Thesis

The thesis has the following main contributions:

(1) On the basis of analyzing the characteristics of ConcurTaskTrees (CTT), a well-know notation for specifying interactive application, extensions of CTT operators with conditional probabilities are proposed in order to take into account IMA. More precisely, the user behavior on IMA is often influenced by conditions on the application and its environment.

(2) A new test modeling language is defined for IMA based on task trees. This language defines all the CTT operators, supports state definition, multimodality and conditional probabilities for IMA.

(3) The transformation rules from CTT into a test model in the TTT language are formally developed.

(4) A solution to generate test data for interactive applications is proposed.

(5) A specification of multimodal interactions and CARE properties is integrated into the TTT language. The specification consists of two different issues when testing IMA: generating tests for multimodal events and checking the validity of the CARE properties.

(6) The TTTEST tool is developed for automating the test of such IMA.

3. The Structure of the Thesis

Chapter 1 presents the background of IMA which includes the definition of multimodality and key features of multimodal interaction. Testing methods for IMA are also summarized in this chapter.

In chapter 2, we present, with more details, the testing approach that we focus on and the related background consisting of task trees, finite state machines, multimodal interaction, CARE properties, operational profiles, conditional probabilities.

In chapter 3, we propose a test modeling language for testing IMA, called TTT, which makes it possible to express scenarios and conditional operational profiles. The transformation rules from CTT specifications into TTT test models are also developed.

Chapter 4 presents the TTTEST tool for testing IMA which includes the test execution environment, and the underlying implementation of this tool. We also introduce two IMA and describe how to test them.

Chapter 1. INTERACTIVE MULTIMODAL APPLICATIONS

In this chapter, we present the background of interactive multimodal application which includes the definition of multimodality and key features of multimodal interaction. Testing methods for IMA are also summarized in this chapter.

1.1. Multimodality

A modality is a channel or path of communication between the human and the computer. It is one of the different senses through which the human can perceive the output of the computer (audition, vision, touch, smell, and taste); modalities will also cover the input devices and sensors allowing the computer to receive information from the human such as speech, pen, touch, manual gestures, gaze and head and body movements. So, a modality is defined as the representational

application, and the physical I/O device used to convey expressions of the representational application.

1.2. Features of Multimodal Interaction

Multimodal interaction can be thought as a sub branch of human-computer interaction using a set of modalities to achieve communication between users and machines. Features of multimodal interaction include the following:

- Permitting the flexible use of input modes, including alternation and integrated use.
- Supporting improved efficiency, especially when manipulating graphical information.
- Supporting shorter and simpler speech utterances than a speech-only interface.
- Giving users alternatives in their interaction techniques.

1.3. Multimodal Fusion

The process of integrating information from various input modalities and combining them into a complete command is referred as multimodal fusion. The early fusion consists in merging the outcomes of each modal recognizer by using integration mechanisms, such as statistical integration techniques, agent theory, hidden Markov models, and artificial neural networks. The late fusion merges the semantic information extracted by using specific dialogue-driven fusion procedures to yield the complete interpretation, such as melting pots [28], and semantic frames [44].

1.4. Design Spaces and Theoretical Frameworks

This section presents a theoretical framework, as well as two design spaces specifically fashioned for analysis of multimodal interaction. The theoretical framework which is presented is the TYCOON framework, created by Martin [25]. For the design spaces, the CASE model [24] classifies IMA and the CARE properties assess the usability of multimodal interaction.

1.5. Introduction to Software Testing

Software testing is an investigation conducted to provide stakeholders with information about the quality of the software product or service under test. Test techniques include the process of executing a program or application with the intent of finding software bugs (errors or other defects), and verifying that the software product is fit for use [24].

1.5.1. Model-Based Testing

A model is a simplified description of a system based on requirements and functions and can help us understanding and predicting the behavior of the system. Model-based testing is a software testing technique in which test cases are derived from a model of the application under test (AUT). Model-based testing has been adopted as an integrated part of the testing process.

There are many benefits of MBT: AUT fault detection, reduced testing cost and time, improved test quality, requirements defect detection, traceability, and requirements evolution. However, MBT is not easy to apply in practice. It is hard to build accurate models, there is a high demand for testers and creating expected output value for the test cases is a difficult problem.

1.5.2. Operational Profile-Based Testing

Operational profile-based testing (OPBT) [49] is a testing technique focusing on usage distributions; it is a sort of combination of model-based testing, random testing, and state-transition-based testing. The operational profile consists of a state machine and probability distributions. The state machine represents the expected behavior of AUT. The probability distribution represents the characteristics of the expected usage of

the AUT, and it is derived based on the survey of operational environments.

1.5.3. Requirement-Based Testing

Requirements-based testing (RBT) is divided in two phases: ambiguity reviews and cause-effect graphing. An ambiguity review is a technique for identifying ambiguities in functional requirements to improve the quality of those requirements. Cause-effect graphing is a test-case design technique that derives the minimum number of test cases to cover 100 percent of the functional requirements.

1.6. Testing Interactive Multimodal Applications

Designing IMA is a complex activity, because of the importance of the human-computer interaction aspect. For the same reason, thoroughly testing such applications is particularly important and requires a lot of effort. Approaches based on formal specifications automating the development and the validation activities have been proposed to deal with this complexity. Testing approaches of IMA consist of: *ICO Method, Algebraic Method, Event B Method, Synchronous Approach and Task tree and Fusion Model Method.*

1.7. Conclusion

This chapter has presented background underlying multimodal interactions. Multimodality, as a concept of human-machine interaction, is defined. Key features as well as cognitive foundations of multimodal interaction helped further underline the capabilities and promises of this new type of human-machine interaction.

Chapter 2. THE CTT MODELING LANGUAGE WITH CONDITIONAL PROBABILITIES

In this chapter, we present the task trees, finite state machines, multimodal interaction, CARE properties, and operational profiles. They are the related backgrounds of our new test modeling language.

We propose to add conditions to operational profiles in task tree and they are a necessary part of the test modeling language.

2.1. Task Trees

Task models [11] are often used in the design of interactive applications and are usually built by human factors specialists. In such models, tasks are represented hierarchically: a task consists of subtasks combined by temporal operators. Therefore, the model describes the subtasks that must be executed to fulfill another, more complex task. A well known notation for task models is ConcurTaskTrees (CTT) [27]. CTT includes four kinds of tasks: User tasks, application tasks, interaction tasks and abstract tasks.

2.2. The Interactive Multimodal Application Memo

The interactive application Memo [4] makes it possible to annotate physical locations with digital stickers ("post it"-like notes). Once a digital sticker has been set to a physical location, it can be read/carried/removed by other users.

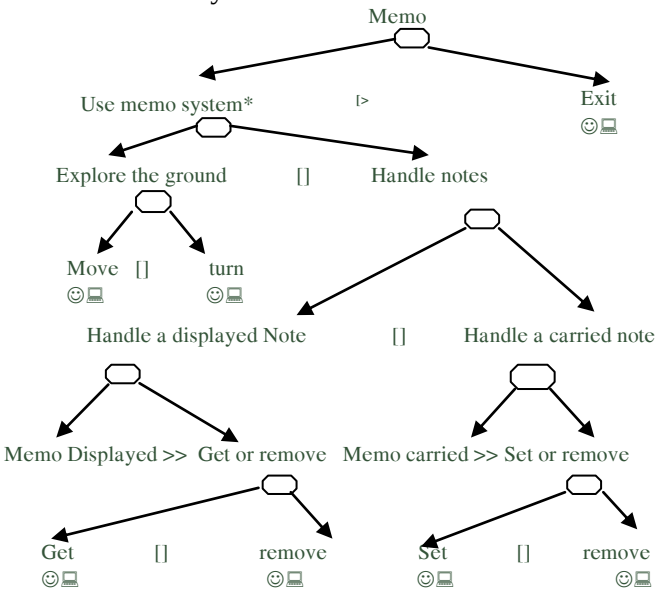

Figure 2.1. The CTT for the Memo application.

A Memo user is equipped with a GPS and a magnetometer enabling the application to compute his/her location and orientation. He/she is also wearing a head mounted semi-transparent display (HMD) enabling the fusion of computer data (the digital notes) with the real environment. Memo provides three main tasks: (1) orientation and localization of the mobile user, so that the application is able to display the visible notes according to the current position and orientation of the mobile user (2) manipulation of a note (get, set and remove a note) and (3) exiting the application.

2.3. Operational Profiles

Operational profiles [28] provide information about the effective usage of an application. In particular, they can be used to guide the test process. For the particular case of interactive applications, operational profiles can be easily defined by assigning occurrence probabilities to some of the described behaviors.

In [20] it is suggested to enhance this test model with operational profile by assigning probabilities to the user actions involved in the tree operators, according to the following syntax:

$T :: t \mid T[]_{pr1,pr2}\, T \mid T|||_{pr1,pr2}\, T \mid T|[]|_{pr1,pr2}\, T \mid T|=|_{pr1,pr2}\, T \mid T[>_{pr} T \mid T>> T \mid T[]>> T \mid T|>_{pr} T \mid T^* \mid T(n) \mid [T]_{pr} \mid$

where t is an elementary task (that is, an application task or a user task).

2.4. Finite State Machines

In [20], Laya Madani et al. proposed a method for automatically generating test data from task trees. The user behavior is automatically extracted from such extended trees as a Probabilistic Finite State Machine (PFSM). The resulting PFSM can then be used to generate test data simulating the user behavior. The PFSM is composed of states, actions and transitions and can be represented using a state diagram. When an interactive application is specified by a finite state machine, the states represent an abstraction of the operating status of interactive applications. The operations can be repeated, so the states can also be repeated. An initial state is a state where the execution of

the interactive applications begins. A final state is a state where the interactive application ends. Inputs are the user's tasks and outputs are application tasks.

2.5. Setting a Level of Abstraction for Testing

level of abstraction of the events sent to the interactive multimodal application must be defined. The level depends on the application properties to be checked and determines which components of the multimodal application will receive test data. In order to identify the levels of abstraction of the events interacting with the IMA, the architecture of the IMA is organized along the PAC-Amodeus software architectural model.

2.6. Generation of Tests at Dialog Controller Level

In order to generate test data at the dialog controller level, Laya Madani et *al.* [20] use task trees. Elementary tasks correspond to complete commands for the controller. In a task tree, a task can be a user task, an abstract task, an application task or an interactive task. User tasks are of no interest for test data generation, since they correspond to cognitive activities without input sent to the application.

Any task T is defined as an I/O finite state machine $M_T = (Q_T , q_{iT} , q_{fT} , I_T , O_T , trans_T)$ where: Q_T is a set of states; q_{iT} is the initial state, the state where the task T starts. q_{fT} is the final state, the state where the task T ends. I_T is a set of application inputs for the task T. O_T is a set of application outputs for the task T. $Trans_T \subseteq Q_T \times (I_T \cup \{\mu\}) \times O_T \times Q_T$ is the set of transitions corresponding to the task T. The input μ is an empty input (no user action). If $(q_T , a, b, s_T) \in trans_T$, then it is written as $q_T - a/b \rightarrow s_T$.

2.7. Taking into Account Conditional Probabilities

In [20] it is suggested to build test models by enhancing task trees with operational profiles assigning probabilities to the user actions involved in the tree operators. Operational profiles provide information about the effective usage of an application. In particular, they can be used to guide the test process. For the particular case of interactive applications, operational profiles can be easily defined by assigning occurrence probabilities to some of the described behaviors. However, such an extension is not sufficient to express all the user actions, especially conditions on user behavior. Several software functions of frequent use have fewer faults than functions of infrequent use and several conditions can affect the fault detection ability of a software testing technique. So, in this thesis we have extended the CTT operators with conditional probabilities. In Table 2.1, we summarize these extensions.

Table 2.1. Extensions of CTT operators with conditional probabilities

CTT operators	CTT operators with conditional probabilities
$T_1[]T_2$	$\boldsymbol{T_1}$ **[]**$(PA_1,PB_1\|cond_1),(PA_2, PB_2\|cond_2),\ldots, (PA_n, PB_n\|cond_n),(PA_0,PB_0)\boldsymbol{T_2}$
$T_1\|\|\|T_2$	$\boldsymbol{T_1}$ $\|\|\|(PA_1,PB_1\|cond_1),(PA_2, PB_2\|cond_2),\ldots, (PA_n, PB_n\|cond_n),(PA_0,PB_0)\ \boldsymbol{T_2}$
$T_1\|[]\|T_2$	$\boldsymbol{T_2}$ **[]**$\|(PA_1,PB_1\|cond_1),(PA_2, PB_2\|cond_2),\ldots, (PA_n, PB_n\|cond_n),(PA_0,PB_0)\ \boldsymbol{T_2}$
$T_1\|=\|T_2$	$\boldsymbol{T_2}$ $\|=\|(PA_1,PB_1\|cond_1),(PA_2, PB_2\|cond_2),\ldots, (PA_n, PB_n\|cond_n),(PA_0,PB_0)\ \boldsymbol{T_2}$
$T_1[>T_2$	$\boldsymbol{T_1}$ **[**$>(PrDeact_1\|cond_1),(\ PrDeact_2\|cond_2),\ldots, (PrDeact_n\|cond_n),(\ PrDeact_0)\boldsymbol{T_2}$
$T_1\|>T_2$	$\boldsymbol{T_1}$ $\|>(PrSus_1\|cond_1),(\ PrSus_2\|cond_2),\ldots, (PrSus_n\|cond_n),(\ PrSus_0)\ \boldsymbol{T_2}$
$[T]$	**[**$\boldsymbol{T}$**]** $(PrA_1\|cond_1),(\ PrA_2\|cond_2),\ldots, (PrA_n\|cond_n),(PrA_0)$

2.8. Evaluation of the Results of Extending the CTT with Conditional Probabilities

Operational profiles provide information about the efective usage of an application. In particular, they can be used to guide the test process. For the particular case of interactive applications, operational profiles can be easily defined by assigning occurrence probabilities to some of the described behaviours. In [4], Laya Madani et *al.* suggested to extend the CTT notation with occurrence probabilities to make possible to specify operational profiles. However, such an extension is not fair with some states of applications. For example, in column 2 of Table 2.2, the test is generated with unconditional probabilitiy for the Memo applications, the action *remove* a note is generated but there is no note in the Memo (at time is 5) because in the time 2, the users *get* a note but the time 4 the users *remove* this note. So it is not logical to extend the CTT with unconditional probabilities.

Table 2.2. Test data generated with unconditional probabilities and conditional probabilities

Time	Test data generated with unconditional probabilities	Test data generated with conditional probabilities
1	Move	Move
2	Get	Get
3	Move	Move
4	Remove	Remove
5	**remove**	**Get**
6	Remove	Remove

In column 3 of Table 2.2, the test generated with conditional probability, the action *get* is generated instead of action *remove* at the time is 5. It is logical, so this means that the CTT needs to be extended to make the definition of such conditions possible.

2.9. Conclusion

In this chapter, task trees are enhanced with operational profiles to make possible the definition of various interaction scenarios. Then, The adequate formal semantics are defined for the CTT operators making possible to translate a task tree into a probabilistic input-output FSM modeling the user behaviors. As an additional improvement to this previous research work, in this thesis we have extended the CTT operators with conditional probabilities.

Chapter 3.TTT: A NEW TEST MODELING LANGUAGE FOR TESTING INTERACTIVE MULTIMODAL APPLICATIONS

3.1. Introduction

In [19], Laya Madani et al. present a technique of test case generation for testing CARE properties by means of a synchronous approach. According to the proposed approach, CARE properties are translated into an enhanced version of the Lustre synchronous language. An improved method presented in [20] uses task trees and a fusion model to perform test data generation for IMA.

As an additional improvement to this previous research work, we have recently proposed an automatic test generation approach based on a new test modeling language, TTT (Task Tree based Test). The decision to create the TTT language is related to the new features defined in chapter 2:

– The TTT language must define all the CTT operators.

– The TTT language must support state definition.

– The TTT language must support conditional probability specifications.

– The TTT language must support multimodality.

3.2. The User Actions Traces

The traces of the user actions are the set of specific actions of the user during his the interaction with the IMA. It includes the

attributes such as time and user actions as speach, gestures, eye contact, mouse, keyboard... These traces are designed as a data table that testers can create by the TTT language. The testers can also insert, update, delete and query data from the table. The table is created at the time the users start to interact with the IMA, and it is removed when the test finishes.

The purposes of storing the traces of the user actions are: (1) to determine the states of the model, (2) to count user actions in order to build conditions that determines next user actions, (3) to create read-only functions in order to calculate conditional probabilities, and (4) to build an oracle to assess the validity of CARE properties.

3.3. Definition of the TTT Language

We use a context-free grammar to describe the TTT language. A context-free grammar can be defined formally by a quad-tuple (N, Σ, P, S), where,

- N is a finite set of non-terminals;
- Σ is a finite set of terminal symbols;
- S is the start symbol
- P is a finite set of production rules with form $v \rightarrow w$, where v are non-terminals, $v \in N$, $w \in (\Sigma \cup N)^*$.

We found that many of the simple expressions in TTT can be described by regular expressions but complex statements must be described by context-free grammars.

3.4. Basic Structure of a TTT Model

In Table 3.1, we define the syntax and semantics of TTT with a context-free grammar to present structure of the TTT model.

Table 3.1. The TTT Syntax

```
1. <tttmodel> ::= <function> ;
2. <function> ::= <function> <statement> ;
3. <statement> ::= ';'
```

```
                | <expr> ';'
                | TESTCTT VAR ';'
                | output '(' VAR ')' ';'
                | input '(' VAR ')' ';'
                | VAR '=' <expr> ';
                | begin <statement_list> end;
    4. <statement_list> ::= <statement>
                            | <statement_list>
    <statement>
    5. <expr> ::=  <expr> '+' <expr>
                | <expr> '-' <expr>
                | <expr> '*' <expr>
                | <expr> '/' <expr>
                | <expr> '<' <expr>
                | <expr> '>' <expr>
                | <expr> GE <expr>
                | <expr> LE <expr>
                | <expr> EQ <expr>
                | <expr> AND <expr>
                | <expr> OR <expr>
                | NOT <expr>
                |'(' <expr> ')'
```

A basic structure of a TTT model consists of a *TESTCTT* block and one or more functions. *TESTCTT* is defined by a set of clauses and the general form of a *TESTCTT* is given in Table 3.2.

Table 3.2. A basic structure of a TESTCTT

```
TESTCTT name;
sets
    basic_types;
var
    local_variables;
init
    initial_state;
begin
invar(i1,i2,...,in);
ctt_operators;
sql_statements;
end;
```

The structure of a *function* is given in Table 3.3. In *function* there are none or many input parameters and there is an output parameter. A *function* has at least one statement. The parameters are given by the <*inputs*>, a comma-separated list of parameter

declarations, each item in the list being a data type followed by an identifier: <*variable-identifier*>.

Table 3.3. A structure of a function

```
function name(<inputs>)returns(<outputs>);
var
    local_ variables;
begin
    statements;
end;
```

3.5. Supporting Conditional Probability Specifications for all the CTT Operators

In this section, we define the syntax for describing the CTT operators, which take into account conditional probabilities. The *ctt_operators* are used to create tasks from conditional operational profiles where the selection of the program inputs is performed with respect to probabilities specified by testers. In Table 3.4, we define the syntax and semantics of TTT in Backus-Naur Form for describing the CTT operators.

Table 3.4. The CTT Syntax

```
<ctt_operator> ::= <choice>|<concur>|<indepen>|<deact>|
                   <suspend>|<option>|<enabling>
<choice> ::= choice(<expression>,<expression>,<expression>,
       < expression>, <expression>,<expression>, <expression>)
<concur> ::= concur(<expression>,<expression>,<expression>,
       <expression>, <expression>,<expression>, <expression>)
<indepen> ::= indepen(<expression>, <expression>,<expression>,
       < expression>, <expression>,<expression>, <expression>)
<deact>::= deact(<expression>,<expression>,<expression>)
<suspend>::=  suspend(<expression>,<expression>,<expression>)
<option>::= option(<expression>,<expression>,<expression>)
<enabling>::=enabling(<expression>, <expression>)
<expression> ::= <assignment_expression>|<expression>
<assignment_expression> ::= <conditional_expression>
                    |<expression> '=' <assignment_expression>
```

```
<conditional_expression> ::= <logical_or_expression> |
                             <logical_and_expression>
<logical_or_expression> ::=
        logical_or_expression> or <logical_or_expression>
<logical_and_expression> ::=
   |<logical_ and_expression> and <inclusive_and_expression>
```

3.6. Supporting state definition

The state of the model corresponds to the past inputs and outputs. To do so, we have to store the user actions traces. We use an SQL-like language to update and search data. We inherit and reduce some SQL statements. In Table 3.5, we define the syntax and semantics of TTT for describing the SQL-like language.

Table 3.5. The SQL-like syntax

```
<sql_statement> ::= <create_table> | <alter_table> |
<drop_table> | <insert> | <delete> | <update>| <select>
<create_table>::= create table <name> (<variable>);
<drop_table>::= drop table <name>;
<insert>::= insert into <name>(<variable>) values
           (<variable>)
<update>::=update <name> set <expression_list> where
           <expression_list>
<delete>::=delete from <name> [where <expression_list>]?
<select>::= select <name> from <name>
           [where <expression_list>]?
<expression_list>::=<expression>|<expression_list>,
           <expression>
```

3.7. Suporting Multimodality

While testing IMA, the number of input events may increase dramatically. Indeed, each input can be produced in several modalities so the number of possible input event combinations can be much bigger than in the case of a single modality. Moreover, the fusion mechanism of IMA depends on a Temporal Window (TW) within which the user event occurs. For example, when two

modalities are used in a redundant way, the resulting events must be combined only when they occur in the same TW.

In this thesis, we build algorithms: *generating tests for multimodal events, checking the validity of CARE properties, equivalence property, redundancy-equivalence property and complementarity Property.*

3.8. Transformation Rules from CTT to Test Model by Using the TTT Language

We propose the following transformation rules in order to map tasks from CTT into statements in the TTT language:

- Rule 1. Interaction tasks in CTT are transformed into outputs of TESTCTT.
- Rule 2. Application tasks in CTT are transformed into inputs of TESTCTT.
- Rule 3. Abstract tasks in CTT for test are mapped into state variables of the TESTCTT. They are declared in the var clause of TESTCTT.
- Rule 4. Operators with conditional probabilities in the augmented CTT for test are represented as predefined functions in TESTCTT.

3.9. Modeling the Interactive Multimodal Application Memo by the TTT Language

In order to better explain how to build the test model in the TTT language, we consider an example about a test model for the interactive multimodal application Memo. *TESTCTT* model for the Memo application is built through four steps:

Step 1. Selecting a test target for the Memo application

Step 2. Defining notations for activities in the model

Step 3. Defining the state variables and data types

Step 4. Writing test scripts for each activity

Table 3.6. TESTCTT model for the Memo application

```
1. TESTCTT Memo;
2. VAR
3. q0, q1, q2, q3 : bool;
4. T, Tout: char;
5. tw : integer;
6. begin
7. INIT (Tout='D')
8. do
9. begin
10.     q0=(Tout<>'D' and Tout <> 'C' and T <>'o');
11.     q1=(T=='o')or(Tout=='G'and T =='g') or
12.        (Tout=='R'and T =='r')or(Tout=='S'and T =='s');
13.     q2 = (Tout=='D');
14.     q3 = (Tout=='C');
15. if (q0)
16. begin
17.      T = Choice('o','',0.5, note_nb()=0,1,
18.          note_nb()>=5,0.1);
19.      insert into u_actions(input) values(T);
20. end
21. if (q1)
22. begin
23.     T = Choice(('o' ,'',0.5, note_nb()=0,1,
24.         note_nb()>=5,0.1);
25.     insert into u_actions(input) values(T);
26. end
27. if (q2)
28.    begin
29.    T= choice('g','r',0.8,note_nb()=0,1,
30.       note_nb()>=5,0.1);
31.       If T ='g'
32.         begin
33.          tw=1;
34.          do
35.          begin
36.           Modalities(Speech_get,Mouse_get,0.3,0.7,
37.                note_nb()=0 ,0,0, note_nb()>=5,0.2,0.8);
38.           Tout = call_Memo(T);
39.           Insert into u_actions(M1,M2,input,ouput)
40.                values(Speech_get, Mouse_get,T,Tout);
41.           Tw =tw+1;
```

```
42.          end
43.          while (tw <=3)
44.       TestRedundantEquivalenceEarly(Speech,Mouse,get,3)
45.            end
46.          else
47.      begin
48.          Tw=1;
49.          do
50.          begin
51.          Modalities(Speech_remove,Mouse_remove,0.8,0.9,
52.               note_nb()=0,0,0, note_nb()>=5,0.9,0.7);
53.          Tout = call_Memo(T);
54.          insert into u_actions(M1,M2, input,output)
               values(Speech_remove, Mouse_remove,T,Tout);
55.          Tw=tw+1;
56.          end
57.             while (tw<=3);
58.           TestEquivalence (Speech, Mouse, get,3)
59.           TestRedundantEquivalenceEarly
60.                           (Speech, Mouse,remove,3);
61.         TestComplementaryEarly(Mouse,Speech,remove, 3);
62.        end;
63.    while (T<>'E');
64.    end
```

```
65.  function note_nb() returns (note_nb: int);
66.  Var get_nb, remo_nb :int;
67.  begin
68.  get_nb= select count(*) from u_actions where input ="g";
69.  remo_nb=select count(*)from u_actions where input ="r";
70.  note_nb= get_nb- remo_nb
71.  end
```

In line 3, we declare state variables to store the states of the model. Variables *qi* (i: 0..3) define the corresponding status of the application (line 10 to 14). If *q0* (resp *q1*) is true then users can "move" or do nothing. If *q2* is true then users can "get" or

"remove" the displayed notes. If *q3* is true then users can "set" or "remove" the carried notes.

Chapter 4. TTTEST: THE SUPPORT TOOL FOR TESTING INTERACTIVE MULTIMODAL APPLICATIONS

4.1. Introduction

In this chapter, we present the tool TTTEST that supports test generation for multimodal events and validity checking for CARE properties of IMA.

4.2. Test Execution Environment

For the purpose of testing IMA, we have built a testing environment, called TTTEST (TTT-based Test), in Figure 4.1.

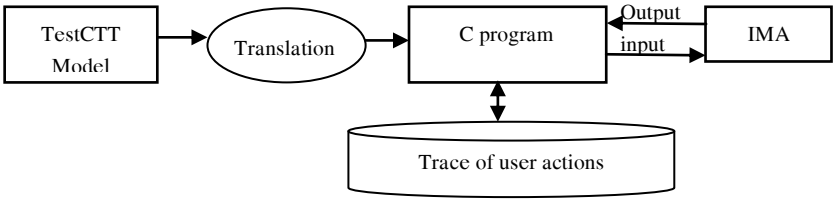

Figure 4.1. The TTTEST Testing Environment.

The TTTEST testing environment consists of four basic components: TESTCTT model specified by the TTT language, C program translated from TESTCTT model, interactive multimodal application under test, traces of the action user. The TTTEST environment activities are described as follows:

– *Step 1.* The TESTCTT model is translated into a C program which is executed.

– *Step 2.* The C program produces output data X from its internal state.

– *Step 3.* Output X is translated into input data for IMA.

– *Step 4.* IMA receives and processes input X and generates output Y.

– *Step 5.* Program C receives Y as input data, updates internal state variable of the model and returns to Step 2.

4.3. The TTTEST Tool

4.4 .Translating TESTCTT Model into C Program

4.4.1. Translation Problems

The translation from TTT into C involves many different problems. The first problem is lexical substitutions. The second problem of translation concerns syntactic transformations. Certain constructs in TTT have equivalent constructs in C, but with a different order of their tokens. The structures of high abstraction level of CTT operators must be converted into structures with more concrete level in C language. Finally, the SQL statements from the TTT language are translated into the C program.

4.4.2. Automatic Translation Solution

To translate automatically from the model TESTCTT into a C program, we use two tools Lex and Yacc. The automatic translation from TESTCTT model into C program is presented as Figure 4.2.

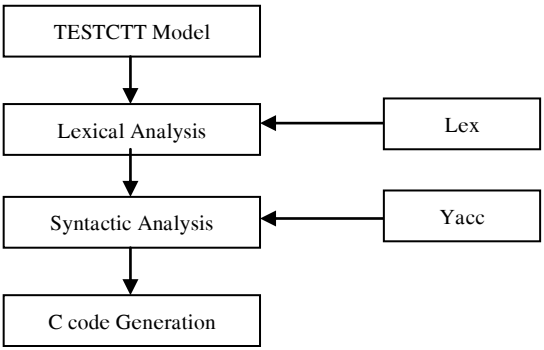

Figure 4.2. Transformation diagrams from TESTCTT model into C program.

4.5. Experimentation

In order to better explain how to automate IMA testing by the TTT language, we consider three examples of testing interactive applications: the NotePad, the Memo and the Map Navigator. The

NotePad is an interactive application, while the Memo and the Map Navigator are IMA.

4.6. Evaluation of the Resulted Test Cases

The purpose of the following experiments is to compare and assess the time for testing of the Memo application manually, by using a C program and by a model written inthe TTT language.We compare the time of two different issues when testing the Memo application: (1) generating tests for multimodal events, and (2) checking the validity of the CARE properties.

To create tests for multimodal events, the user actions such as move, turn, get, set, removeare repeated 100 times and sent to the Memo application. To check the validity of CARE properties, user actions such as get, set, removeare done repeatedly in a TW and transferredto the Memo application.

We conducted three experiments as follows:

Experiment 1: We test the Memo application manually.We collected the complete time of the manualtesting. The Table 4.1 is the result of experiment 1.

Experiment 2: We test the Memo application by writing the C program. The C program generates test data and check the validity of the CARE properties automatically. We collected the complete time of the testing. The Table 4.2 is the result of experiment 2.

Experiment 3: We test the Memo application by writing the model in the TTT language. The Model is translated into aC program that generates test data and checks the validity of the CARE properties automatically. We collected the complete time of the testing. The Table 4.3 is the result of experiment 3.

Table 4.1. Results of the Experiment 1

Completion Time (Minutes)			
Test Data Generation	Checking the validity of Care properties	Failure Analysis Report	Total
30	123	35	188

Table 4.2. Results of the Experiment 2

Completion Time (Minutes)		
Writing C Program	Run the C program	Total
545	2	547

Table 4.3. Results of the Experiment 3

Completion Time (Minutes)			
Writing TestCTT model	Convert to C program	Run the C program	Total
173	2	3	178

In Table 4.1, the completion time of the experiment 1 is 188 minutes, while in Table 4.2, the completion time of the experiment 2 is 547 minutes. In Table 4.3, the completion time of the experiment 3 is 178 minutes. In summary, the results of the three experiments confirmed that the TTT language may help the testers to test the Memo application faster.

Although testing with the TTT language seems only slightly faster than manual testing, it should be kept in mind that using the TTT language makes it possible to replay tests as many times as wanted (when debugging the application) without additional effort while manual testing requires testers to do the work again.

4.7. Conclusion

We built the TTTEST tool that supports automatically testing IMA. The tool supports creating the TESTCTT model in the TTT language and translating the TESTCTT model into a C program. We have described the features of TESTCTT and how the tool was built. Given an IMA, we need to write a model in TTTEST with which the IMA interacts. The model represents the target and the way we want to test IMA.

CONCLUSIONS AND FUTURE WORKS

The Thesis Results

With the initial objectives, the thesis titled "Automatic testing of interactive multimodal applications" has achieved some results as follows:

- We proposed enhancing task tree with conditional operational profiles by assigning conditional probabilities to the user actions involved in the tree operators.
- We defined the TTT language, which is a new test modeling language for IMA.
- We have extended the TTT language to solve two problems: generating test data and checking CARE properties.
- We also build the tool TTTEST that supports automatically testing these applications. We have described the features of the tool and its underlying algorithms.

Perspectives and Future Work

In addition to the results achieved in the thesis, some problems may be posted for further research.

- TTT-based testing methods need more time to construct the model of the IMA. So, we intend to derive techniques for reverse engineering existing IMA by automatic exploration, leading to a partial automatic generation of TESTCTT model.
- We make communication channels among test managers. As future work, we intend to support the construction of reports for test management with test cost, test coverage, and defects found.
- We will research and improve data generation algorithms and check the validity of the care properties that we proposed.
- The TTTEST tool developed in this work can be further extended in the future to study the test data generation to test safety properties of this kind of IMAs.

LIST OF PUBPLICATIONS

1. Le Thanh Long, Nguyen Thanh Binh, Ioannis Parissis, “A New Test Modeling Language for Interactive Applications Based on Task Trees”, In Proceedings of the 4th International Symposium on Information and Communication Technology (SoICT 2013), ACM Publisher, ISBN: 978-1-5403-2454-0, pp.285-293.
2. Le Thanh Long, Nguyen Thanh Binh, Ioannis Parissis, “TTT: A Test Modeling Language for Interactive Applications Based on Task Trees”, in Proceedings of 16th National Conference: Selected Problems about IT and Telecommunication (@ 2013), ISBN: 978-604-67-0251-1, pp.333-338.
3. Le Thanh Long, Nguyen Thanh Binh, Ioannis Parissis, “A solution of generate test data for interactive applications”, In Proceedings of the 7th National Conference on Fundamental and Applied Information Technology Research (FAIR 2014), ISBN: 978-604-913-300-8, pp 134-143.
4. Le Thanh Long, Nguyen Thanh Binh, Ioannis Parissis, “Model-to-C program translation in TTTEST”, in Proceedings of 17th National Conference: Selected Problems about IT And Telecommunication, Ho Chi Minh city (@ 2015), 05-06/11/2015, ISBN: 978-604-67-0645-8, pp 142-149.
5. Le Thanh Long, Nguyen Thanh Binh, Ioannis Parissis, “Testing Multimodal Interactive Applications By Means of The TTT Language”, Domain Specific Model-Based Approaches To Verification And Validation - Amaretto 2016, In conjunction with the 4th International Conference on Model-Driven Engineering and Software Development - MODELSWARD 2016, 19 February, 2016 - Rome, Italy, ISBN: 978-989-758-166-3, pp 23-32.
6. Le Thanh Long, Nguyen Thanh Binh, Ioannis Parissis, “TTTEST : The Tool Support For Testing Interactive Multimodal Applications”, In Proceedings of the International Conference on Electronic, Information and Communication (ICEIC 2016). 27-30/01.2016, IEEE Publisher, pp 78-81.
7. Le Thanh Long, Nguyen Thanh Binh, Ioannis Parissis, “Automatic Testing of Interactive Multimodal Applications”, in Communications in Computer and Information Science (Scopus indexed), 2017, pp 93-113