Testing Multimodal Interactive Applications by Means of the TTT Language

Le Thanh Long[1], Nguyen Thanh Binh[2] and Ioannis Parissis[3]
[1]*Department of Computing, Duy Tan University, 182 Nguyen Van Linh, Da Nang, Vietnam*
[2]*Department of Computing, The University of Danang, University of Science and Technology, 54 Nguyen Luong Bang, Da Nang, Vietnam*
[3]*Grenoble INP LCIS, Univ. Grenoble Alpes, F-26902 Valence, France*

Keywords: Interactive Multimodal Applications, Test Modelling Language, CARE Properties.

Abstract: Developing interactive applications is a complex activity as they must deal with various kinds of human-computer interactions. This is especially true when these interactions use multiple modalities (voice, gesture…). As a result, thoroughly testing such applications is particularly important and requires more effort than for traditional interactive applications. In this paper, we propose an approach for automating the test generation of such multimodal applications. This approach is based on the definition of a new test modelling language, TTT. Test models provided in TTT can be translated intest generators.TTT deals with a well-known class of multimodality properties: the CARE properties. The whole approach is illustrated on a case study.

1 INTRODUCTION

Interactive Multimodal Applications (IMA) support communication with the user through different modalities, such as voice or gesture. They have the potential to greatly improve human-computer interaction, because they can be more intuitive, natural, efficient, and robust. Flexibility is obtained when the user can use equivalent modalities for the same tasks while robustness can result from the integration of redundant or complementary inputs.

The CARE properties (Complementarity, Assignment, Redundancy, Equivalence) can be used as a measure to assess the usability of the multimodal interaction (Coutaz et al., 1995). Equivalence and Assignment represent the availability and, respectively, the absence of choice between multiple modalities for performing a task while Complementarity and Redundancy express relationships between modalities. The flexibility and robustness of multimodal applications 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. They adapt existing formalisms to the particular context of interactive applications. Examples of such approaches are the Formal System Modelling (FSM) analysis (Duke and Harrison, 1993), the Lotos Interactor Model (LIM) (Paternò and Faconti, 1993), the Interactive Cooperative Objects (ICO) (Palanque and Bastide, 1995) or formal methods such as B (Aıt-Ameur and Kamel, 2004). Model-based testing methods focusing on the specification of the user behaviour have also been studied. For instance, the method presented in (Richard et al., 1997) relies on the specification of a finite state machine.

In (Ter Beek et al., 2009), Maurice H. terBeek *et al.* propose stochastic modelling and model checking to predict measures of the disruptive effects of interruptions on user behaviour. The approach also provides a way to compare the resilience of different interaction techniques to the presence of external interruptions that users need to handle. In (Palanque et al., 2009), P. Palanque *et al.* presents an approach for investigating in a predictive way potential disruptive effects of interruptions on task performance in a multitasking environment.

In (Kamel and Aït Ameur, 2007), N. Kamel *et al.* propose a formal model allowing representing the

Long, L., Binh, N. and Parissis, I.
Testing Multimodal Interactive Applications by Means of the TTT Language.
DOI: 10.5220/0005842500230032
In *Proceedings of the International Workshop on domAin specific Model-based AppRoaches to vErificaTion and validaTiOn (AMARETTO 2016)*, pages 23-32
ISBN: 978-989-758-166-3

input multimodal user interaction task and the CARE usability properties. Once the multimodal interaction task model is designed, the corresponding property is checked using the SMV (Symbolic Model Verifier) model-checker. They also propose an approach for checking adaptability properties of multimodal User Interfaces (UIs) for systems used in dynamic environments like mobile phones and PDAs (Kamel et al., 2008). The approach is based on a formal description of both the multimodal interaction and the property. The SMV model-checking formal technique is used for the verification process of the property. In (Mohand-Oussaïd et al., 2015), L. Mohand-Oussaïd *et al.* present a generic approach to design output multimodal interfaces. This approach is based on a formal model, composed of two other models: semantic fission model for information decomposition process and allocation model for modalities and media allocation to composite information. An Event-B formalization has been proposed for the fission model and for allocation model. This Event-B formalization extends the generic model and supports the verification of some relevant properties such as safety or liveness.

The synchronous approach has been proposed to model and verify by model-checking some properties of interactive applications (Madani et al., 2005), but its applicability is limited to small pieces of software.

In (Madani and Parissis, 2009), 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 (Madani and Parissis, 2011) uses Task trees and a fusion model to perform test data generation for interactive multimodal applications. As an additional improvement to this previous research work, we have recently proposed an automatic test generation approach based on a new test modelling language, TTT (Task Tree base Test) (Le et al., 2014) The main new feature of the TTT language is that it supports conditional probability specifications, used to express advanced operational profiles. Such conditional specifications may depend on the history of the user actions. A test generation engine makes it possible to produce test data compliant with such a description. For this, user actions are stored during the test execution.

In this paper, we extend the above mentioned work in order to take into account multimodality. The TTT language is extended to specify multimodal events of IMA and CARE properties as well as to check the validity of CARE properties. Hence, multimodal test data can be automatically generated from a TTT specification of IMA.

The paper is organized as follows: In Section 2, we provide the necessary background. Section 3 presents the extension of theTTT language for generating tests and checking the validity of CARE properties. A case study is presented in Section 4.

2 BACKGROUND

2.1 Task Trees

Task trees are often used in the design of interactive software applications (Paternò et al., 1997) to hierarchically build task models. A well-known notation for such task models is ConcurTaskTree (CTT). CTT includes four kinds of tasks: User tasks (no interaction with the application, just an internal cognitive activity such as thinking about how to solve a problem), application tasks (application performance, such as generating the results of a query, no interaction with the user), interaction tasks (involving user actions with immediate feedback from the application, such as editing a document) and abstract tasks (tasks composed of other subtasks). A CTT abstract task is composed of subtasks connected by means of temporal operators, for example, there is an enabling operator denoted by >> which specifies that one task enables a second one when it terminates.

A CTT model is mainly intended to help designers to define interactive applications. However, it has been shown that the same notation can be also used to define test models describing the interaction between the user and the application and providing valuable information about the possible user behaviour.

2.2 Finite State Machines

Finite State Machines (FSMs) are widely used to model the behaviour of interactive applications. This model includes the states, the actions and the transitions presented by a state diagram (Madani and Parissis, 2009). 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. Initial state is a state that interactive applications begin to be used. Final state is the state where the interactive

application ends. Inputs are the user's tasks and outputs are application tasks.

2.3 Multimodal Interaction: CARE Properties

An interactive multimodal application uses at least two modalities (keyboard, speech, mouse...) for a given direction (input or output). Within a multimodal application, modalities can be used independently, but the availability of several modalities naturally raises the issue of their combined use (fusion of modalities). When talking about test data generation, we are mainly concerned with inputs, so in this paper we focus on multimodal input interaction.

The combined use of modalities is constrained by temporal constraints. It can be carried out sequentially or concurrently (Coutaz et al., 1995) within a Temporal Window (TW), that defines a time interval. The modalities of a set M are used concurrently if they are used at the same instant. The modalities of a set M are used sequentially within the TW, if there is at most one active modality at every instant and if all the modalities in this set are used within the TW. The concurrency and the sequencing express a constraint on the interaction space. The absence of a temporal constraint means that the duration of the TW is infinite. The CARE properties form an interesting set of relations that are relevant when characterizing multimodal applications. The Assignment implies that a single modality is assigned to a task. The Equivalence of modalities implies that the user can perform a task using a modality chosen amongst a set of modalities. The Complementarity denotes several modalities that convey complementary chunks of information. The complementary modalities must be used simultaneously or sequentially within the same TW. The Redundancy indicates that the same piece of information is conveyed by several modalities. Redundant modalities must also be used simultaneously or sequentially within the same TW.

2.4 Operational Profiles

Operational profiles (Musa, 1993) 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 behaviours. In (Madani and Parissis, 2009), the CTT notation was extended with occurrence probabilities to make possible to specify operational profiles.

2.5 Generating Test Data for IMA

Task trees are used in the design of interactive applications. To generate automatically the test data from task trees, the task tree is translated into a probabilistic finite state machine (PFSM).

It is assumed that the PFSM is simulated while the interactive application under test is executed and that inputs and outputs are exchanged between them on-the-fly. During the simulation, assuming the PFSM to be in a given state, an input is chosen according to the probabilities of the outgoing transitions of this state. The chosen input is then sent to the interactive application, the resulting application outputs are read and the next state computed, and so on.

2.6 The Interactive Multimodal Application Memo

The interactive application "Memo" (Madani and Parissis, 2009) 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. A Memo user is equipped with a GPS and a magnetometer enabling the application to compute his/her location and orientation. S/he 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. So, the mobile user can get a note and carry it while moving. S/he can set a carried note to a specific place or delete a visible or carried note.

Figure 1 shows an extended CTT for the Memo application (interaction tasks are represented by ☺💻). To generate test data, the task tree is translated into a PFSM. The PFSM is simulated while the interactive application under test is executed and that inputs and outputs are exchanged between them on-the-fly. It is thus possible to describe abstract interaction scenarios as task trees, and observe the behaviour of the interactive application under test. Figure 2 shows a PFSM example for the Memo application.

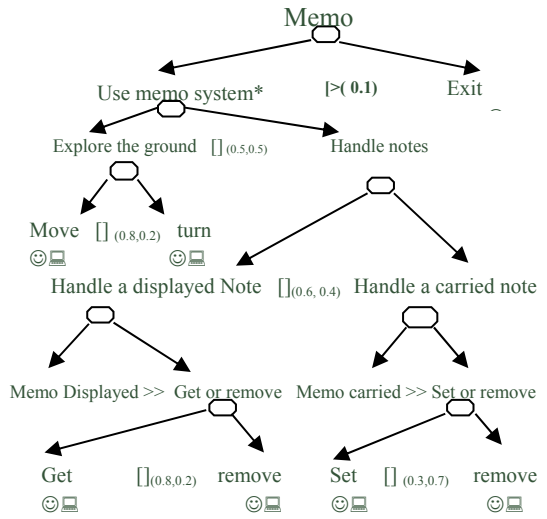

Figure 1: Example of Task tree model.

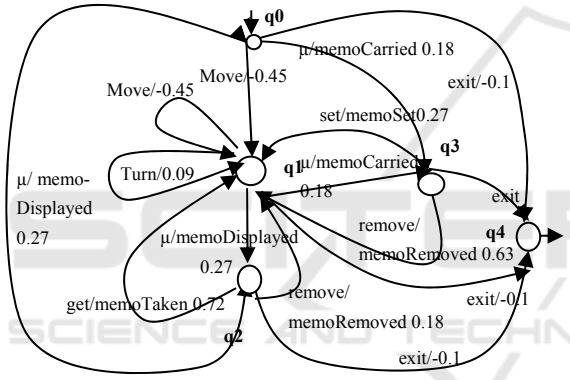

Figure 2: FSM Example for the Memo application.

Figure 3 shows a fusion model for the Memo application.

```
Tasks ( get, set, remove, move, turn, exit);
Modalities (Speech (get, set, remove),
Mouse(get, set, remove),
Keyboard(get, set, remove, move, turn, exit));
Equivalence ((Speech, Mouse, Keyboard),
             (get, set, remove));
Assignment ((Keyboard), (move, turn, exit));
```

Figure 3: Example of fusion model.

2.7 Taking into Account Conditional Probabilities

The above presented approach uses several notations, inspired from existing modelling languages, to build test models: a model of the application behaviour (a task tree), a model of the interactive tasks (FSM), operational profiles (annotations on the task tree), and modality specifications. The variety of notations makes the modelling process hard. Moreover, operational profiles cannot be defined using conditions (however, an occurrence probability is often assigned to an event according to a condition).

Therefore, we have proposed the test modelling language TTT (Le et al., 2013) allowing to express:

- Scenarios for interactive applications.
- Conditional probability specifications for task trees.
- The "traces" of the user actions and read-only functions on these traces.
- Expected properties of the application.

The conditional probability specifications for task trees must be defined in the test model. This means that the TTT language is designed to allow the definition of variables, for example, *Cond = (X > 5),* where *X* is an application input or output variable. Moreover, there are a lot of "rich" conditions that need to be expressed, for example, *Cond=F(parameter)>5* where *F* is a function that can return a float value.

2.7.1 The TTT Language

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.

<tttmodel> ::= <testctt><function>+
<testctt> ::= <testctt_name><testctt_set>
<testctt_var><testctt_init><begin_end>
<testctt_name>::=TESTCTT<name>
<testctt_set> ::= sets <basic_type>+ ;
<testctt_var> ::= var<local_variable>+ ;
<testctt_init> ::= init<initial_state>+ ;
<begin_end> ::= begin <statement>+ end;
<statement>:=<beginend>|<invar_operator>|
<ctt_operator>|<sql_statement>|
<conditional_struct> |<iteration-statement>
<begin_end> ::= begin<statement>+ end;

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 the tester.

<ctt_operator> ::= <choice> | <concurrency> | <deact> | <sr> | <option> | <enabling> | <iteration> | <fiteration>

We save all the past actions of the users and build functions on them. Functions are intended to be part of the conditions. We use an SQL-like language to update and search the data. We inherit and reduce the following SQL statements:

<sql_statement> ::= <create_table> | <alter_table> | <drop_table> | <insert> | <delete> | <update>| <select>

2.7.2 Test Execution Environment

For the purpose of testing interactive applications, we have built a testing environment (Le et al., 2014), called TTTEST (TTT-based Test), in Figure 4.

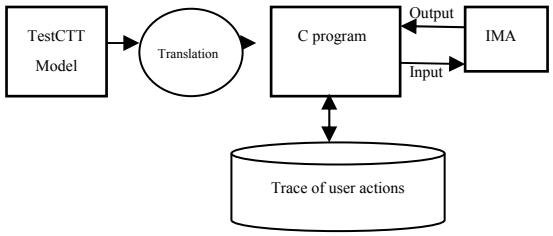

Figure 4: The TTTEST testing evironment.

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

- Step 0: The TESTCTT model is translated into a C program which is executed.
- Step 1: The C program produces output data X from its internal state.
- Step 2: Output X is translated into input data for IMA.
- Step 3: IMA receives and processes input X and generates output Y.
- Step 4: Program C receives Y as input data, updates internal state variable of the model and returns to Step 1.

A TESTCTT model is specified withTTT language. We translate a TESTCTT model into a C program which implements the corresponding test generator. The translation, the details of which are presented in (Le et al., 2014), involves many different steps. The simplest steps are lexical substitutions; operators and keywords in TTT are replaced by corresponding C operators and tokens. The second level of translation involves syntactic transformations. Certain constructs in TTT have equivalent constructs in C, but with differing orders of the tokens. The structures of high abstraction level of CTT operators must be converted into structures with more concrete level in the C language. Finally, the SQL statements from the TTT language are translated into equivalent C statements.

3 TAKING INTO ACCOUNT 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 single modalities. Moreover, the fusion mechanism of IMA depends on TWs 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 sameTW.

The above observations suggest that there are two different issues when testing IMA: (1) generating tests for multimodal events, (2) checking the validity of the CARE properties. While the first issue is strictly related to test generation, the second one should be part of a test oracle. We propose to extend the TTT language to deal with both issues, as described in the following subsection.

3.1 Generating Tests for Multimodal Events

To simulate user behaviours for IMA, we use a test data generation technique based on conditional operational profiles. We add the operator *modalities* to the TTT language to generate tests for multimodal events. The syntax of modalities operator is the following:

<modalities>::=modalities (<expression-list>)

The tester can use *modalities*(E_{M1}, E_{M2},…, E_{Mn}, p_1,p_2,…,p_n, $cond_1$, p_{11},p_{12},…,p_{13}, $cond_2$, p_{21},p_{22},…,p_{23}), where $i \in [1,n]$ E_{Mi} are events, p_i are probabilities; $cond_i$ are conditions; and $p_{i,j}$ ($i \in [1,n]$, $j \in [1,n]$) are conditional probabilities. The semantics of this operator is expressed as follows:

```
{ n is a random real number in [0,1]
  n= rand(1)
if (Cond1== TRUE) {
if (n<= P11) EM1= 1 else EM1 = 0;
      if (n<= P12) EM2= 1 else EM2 = 0;
   …
if (n<= P1n) EMn= 1 else EMn= 0;
}
   elseif (Cond2== TRUE) {
      if (n<= P21) EM1= 1 else EM1 = 0;
      if (n<= P22) EM2= 1 else EM2 = 0;
   …
if (n<= P2n) EMn= 1 else EMn= 0;
   }
   else {
      if (n<= P1) EM1= 1 else EM1 = 0;
      if (n<= P2) EM2= 1 else EM2 = 0;
   …
```

```
    if (n<= Pn) EMn= 1 else EMn= 0;
       }
    }
```

Consider the following example:

Modalities (speech(Remove), mouse(Remove), keyboard(Remove),0.5,0.5,0.5,note_nb()=0,0, 0,0,note_nb()>=5,0.5,0.9,0.7);

The events *Speech(remove), Mouse(remove)* and *Keyboard(remove)* are generated along probabilities 0.5, 0.5, 0.5 respectively. If there is no note, the user cannot remove, so probabilities are 0, 0, 0. But if there are more than 5 notes, the user will use other probabilities for these events. The events generated are presented in Table 1.

Table 1: Events are generated by Modalities operator.

Time	sR	mR	kR	Memo
1	0	0	0	…
2	0	0	0	Se
3	0	1	1	Tak
..	…	…	…	…

In Table 1, we use the abbreviation *sR*, *mR* and *kR* respectively for *speech(Remove), mouse(Remove)* and *keyboard(Remove)*. At the time 1, there is no note in the Memo, the user do not use any event. But at the time 3 when a note is visible (Set (Se) occurred in the previous step) the user takes it (Tak) by *mouse(Remove)* and *keyboard(Remove)*.

3.2 Checking the Validity of CARE Properties

3.2.1 Equivalence

Let M_1, M_2 be two modalities. Let E_{M1}, E_{M2} be two expressions along M_1, M_2 respectively. Two modalities M_1, M_2 are equivalent with respect to task T, if every task $t \in T$ can be activated by E_{M1} or E_{M2}. Equivalence admits a single input event to be propagated. We add the operator *TestEquivalence*(E_{M1} ,E_{M2}, *T, tw)* into TTT language to check the validity of the *Equivalence* property.

The syntax of *TestEquivalence* operator is the following:

<TestEquivalence>::= TestEquivalence(<expr>,<expr>,<expr>,<expr>)

The tester can use *TestEquivalence*(E_{M1} ,E_{M2}, *T, tw)* and the meaning of this operator is expressed as follows:

```
1.begin
2. T1 = select distinct Tout from U_ACTIONS
where EM1= EM1 and time between(now()-tw)
and now();
3. T2 = select distinct Tout from U_ACTIONS
 where EM2 = EM2 and time between(now()-tw)
 and now();
4. if ((T == T1) and (T ==T2))
5.      IsEquivalence=  True
6.  else begin
8.  output("EM1 and EM2 are not equivalent");
9.  stop program;
10.end
11.end
```

T1 and *T2* are two tasks corresponding to two events *E1* and *E2* in U_ACTIONS table (lines 2,3). If task *T1* is different from task *T2*, events *E1* and *E2* are not equivalent (line 8). The program under test will be stopped (line 9).

Table 2 shows an extract example of the execution trace, the result of *TestEquivalence(speech, mouse, get, 7)*.

Table 2: The result of TestEquivalence.

Time	Speech(get)	Mouse(get)	Tout	TestEquivalence
1	1		get	
2				Speech(get)=
3		1	get	Mouse(get)

It can be observed that when the user does *speech(get)*, *Tout* is equal to *"get"* in time 1. When the user uses the mouse to choose "get" *(mouse(get))* *Tout* is equal to *"get"* in time 3. So event *speech(get)* is equivalent to event *mouse(get)*.

3.2.2 Redundancy-Equivalence

If there are several input events, redundancy requires the fusion process to choose one event among those of all the available modalities. Equivalence admits a single input event to be propagated.The Redundancy-Equivalence input events which are temporally close are merged and the associated output task is enabled as soon as the required inputs have been identified. The occurrence of one event of every modality in the current TW is enough to enable the output task. It is possible that several events of the same modality occur in this window. In that case, the task is computed according to the last event of each modality.

We add operator *TestRedundant_EquivalenceEarly* into TTT to test the Redundancy-Equivalence of two events E_{M1} and E_{M2} in early fusion strategies. The syntax of *TestRedundant_EquivalenceEarly* operator is the following:

<testRE>::= *TestRedundant_EquivalenceEarly (<expr>,<expr>,<expr>,<expr>)*

The tester can use *TestRedundant_EquivalenceEarly (E_{M1}, E_{M2}, TaskTM1M2, tw)* and the semantics of this operator is as follows:

```
1.begin
2.T_out = select distinct Tout from
   U_ACTIONS where((EM1 = EM1)or (EM2 =EM2))
and (time between(now() - tw)and now())
3.T_out_nb = select count(Tout) from
U_ACTIONS where((EM1 = EM1)or(EM2 = EM2))
 and (time between(now() - tw)and now())
4.if((Tout_nb==1)and(T_out==taskTM1M2))then
5.  output ("EM1 and EM2 are redundant -
equivalent");
6.else
7.  begin
8.output ("EM1, EM2 are not redundant-
equivalent");
9.  stop program;
10.end
11.end
```

T_out is the task that is generated in Temporal Window. *Tout_nb* is the number of tasks generated from the event E_{M1} or E_{M2} (line 5). The Redundancy-Equivalence property of two events E_{M1} and E_{M2} is tested by condition (line 4): *((Tout_nb == 1) and (T_out = taskTM1M2))*. If there is only one task generated in TW and *T_out* is the *taskTM1M2*, *EM1* and *EM2* are Redundant-Equivalence. Table 3 shows an extract of the execution trace resulting from *TestRedundant_EquivalenceEarly (Speech_T, Mouse_T, TaskTM1M2, 5)* with *Tw* = 5.

Table 3: The result of TestRedundant_EquivalenceEarly.

Time	E_{M1}	E_{M2}	Tout	*TESTRedundant_ EquivalenceEarly*
1	Speech_Task		Task	
2		Mouse_Task		Speech_Task,
3	Speech_ Task			Mouse_Task are
4	Speech_ Task			Redundant-
5		Mouse_ Task		Equivalence

3.2.3 Complementarity (C)

Let M_1, M_2 be two modalities. Let E_{M1}, E_{M2} be two expressions along M_1, M_2 respectively. Two modalities M_1, M_2 are complementary with respect to a set of Task *T*, if every task $t \in T$ can be activated by E_{M1} and E_{M2}. E_{M1} and E_{M2} must occur in the same *TW, i.e.Abs((time(E_{M1}) – time(E_{M2})) < Tw.*

The complementary input events which are temporally close are merged and the associated output task is enabled as soon as the required inputs have been identified. The occurrence of one event of every modality in the current TW is enough to enable the output task. It is possible that several events of the same modality occur in this window. In that case, the task is computed according to the last event of each modality.

We add operator *TestcomplementaryEarly (E_{M1}, E_{M2}, TaskTM1M2, tw)* into TTT language to test the complementary of two events E_{M1} *and* E_{M2}. The syntax of *TestcomplementaryEarly* operator is the following*:*

<testcom>::=*TestcomplementaryEarly(<expr>, <expr>,<expr>,<expr>)*

The tester can use *TestcomplementaryEarly (E_{M1}, E_{M2}, TaskTM1M2, tw)* and the behavior of this operator is as follows:

```
1.begin
2.EM1_out = select top 1 EM1 from U_ACTIONS
where(time between (now()-tw) and now ())
order by time desc
3.EM2_out = select top 1 EM2 from U_ACTIONS
where(time between (now()-tw) and now ())
order by time desc
4.T_out = select distinct Tout from
U_ACTIONS
5.if ((EM1== EM1_out) and (EM2 == EM2_out)
and(T_out ==taskTM1M2)) then
6.  output ("EM1 and EM2 are complementary");
7.else
8.  begin
9.   output ("EM1 and EM2 are complementary");
10.  stop program;
11. end
12.end
```

EM1_out and *EM2_out* are the last events occurred in the Temporal Window. *T_out* is the task occurred in the Temporal Window. The Complementarity of two events E_{M1} and E_{M2} is tested by condition (line 5): ((E_{M1} == *EM1_out) and* (E_{M2} == *EM2_out)and (T_out ==task)*). If E_{M1} and E_{M2} are last events in TW and *Tout* is the *taskTM1M2* then E_{M1} and E_{M2} are complementary. Table 4 shows an extract example of the execution trace resulting from *TestComplementaryEarly (Speech_T1, Mouse_T2, Task12, 5) with Tw = 5.*

Table 4: The result of TestComplementaryEarly.

Time	**E_{M1}**	**E_{M2}**	**Tout**	***TestComplemen_ taryEarly***
1	Speech_T1			
2		Mouse_T2		Speech_T1,
3	Speech_T1			Mouse_T2 are
4	Speech_T1			complememtary
5		Mouse_T2	TaskT12	

4 TESTING THE MEMO APPLICATION

The TESTCTT model of Memo is built through four steps: (1) selecting a test target; (2) designing

notations of activity in the model; (3) designing the state variables and selecting data types for variables; (4) writing test scripts for each activity. Figure 5 presents a part of this test model.

```
1. TESTCTT Memo;
2. VAR
3. q0, q1, q2, q3, q4 : 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
    (Tout=='R'andT=='r')or(Tout=='S'and T=='s');
12. q2 = (Tout=='D');
13. q3 = (Tout=='C');
14. if (q0)
15. begin
16. T = Choice('o','',0.5, note_nb()=0,1,
17. note_nb()>=5,0.1);
18. insert into U_ACTIONS(input) values(T);
19. end
20. if (q1)
21. begin
22. T = Choice(('o' ,'',0.5, note_nb()=0,1,
23. note_nb()>=5,0.1);
24. insert into U_ACTIONS(input) values(T);
25.    end
26. if (q2)
27. begin
28. T   =   choice('g'   ,'r',0.8,note_nb()=0,   1,
    note_nb()>=5,0.1);
29. if T ='g'
30.        begin
31. tw=1;
32.         do
33.         begin
34.          Modalities(Speech_get,Mouse_get,
35.            0.3,0.7,note_nb()=0 ,0,0,
36. note_nb()>=5,0.2,0.8);
37.          Tout = call_Memo(T);
38.         Insert into U_ACTIONS(M1,M2,input,
39.      output)values(Speech_get,Mouse_get,T,
40.                          Tout);
41.          Tw =tw+1;
42.          end
43.          while (tw<=3)
44. TestRedundantEquivalenceEarly_
45.              (Speech, Mouse, get, 3)
46.         end
47.       else
48.   begin
49.       Tw=1;
50.       do
51.       begin
52. Modalities(Speech_remove,Mouse_remove,
53.            0.8,0.9,note_nb()=0,0,0,
54. note_nb()>=5,0.9,0.7);
55.         Tout = call_Memo(T);
56. insert       into       U_ACTIONS(M1,       M2,
    input,       output)       values(Speech_remove,
    Mouse_remove,T,Tout);
57.       Tw=tw+1;
58.       end
59.          while (tw<=3);
60. TestRedundantEquivalenceEarly_
61.              (Speech, Mouse, remove, 3);
62.     end;
63. while (T<>'E');
64. end
65. FUNCTION note_nb() returns (note_nb: int);
66. varget_nb, remove_nb :int;
67.   begin
68. get_nb=  select  count(*)  from  U_ACTIONS  where
    input ="g";
69. remove_nb= select count(*) from U_ACTIONS where
    input ="r";
70. note_nb= get_nb- remove_nb
71.  end
```

Figure 5: The test model for Memo in TTT.

Based on the rules described in section 5, the TESTCTT model is transformed into a C program. After the translation is completed, the C program is compiled and executed to generate test data.Table 5 shows an extract of the execution trace and the result of *TestRedundantEquivalenceEarly*(lines 43,59).

Table 5: An extract of the execution trace and the result of *TestRedundantEquivalenceEarly.*

Time	E_{M1}	E_{M2}	TM1M2	Output
1	Move			M_Displayed
2	Speech_get		Get	M_Taken
3		Mouse_get		
4		Mouse_get		
5	Move			M_Display
6	Speech_remove		Remove	M_Remove
7		Mouse_remove	Remove	M_Remove

In line 1, the user *moves* and a note appeared (*M_Displayed*). The test generator produces input data *Speech_get* (choice between *get* or *remove* in the state q2). In lines (2, 3, 4) because of the redundancy mode, the user actions *Speech_get*, *Mouse_get* are sent through the Memo causing only one action *Get* and Memo returns output *M_Taken* (line 2). The test generator calculates and determines the application is in state *q1*. In state *q1*, TESTCTT model generates input data *move* (choice between *move* or "-" in the state *q1*). When the user moves, a note appeared on the Memo (*M_Display*) (line 5). The user removes this note (lines 6,7). The user actions *Speech_remove*, *Mouse_remove* are sent to Memo causing two actions *Remove* and Memo returns two outputs *M_Remove*. So *Mouse_remove*, *Speech_re-move* are not Redundant-Equivalent, therefore the C program stops the Memo application and displays a message "*Mouse_remove* and *Speech_remove* are not Redundant-Equivalent".

5 CONCLUSIONS

IMA are intuitive, natural, efficient, and robust. The flexibility and robustness of multimodal applications

are increasing the complexity of the design, development and testing. We have built a new modelling language TTT to test interactive applications. For multimodal applications, we have extended the TTT language to solve two problems: generating test data and checking CARE properties. We defined a new operator *Modalities* to generate tests for multimodal events. The CARE properties are tested by the *TestEquivalence, TestRedundant_EquivalenceEarly* and *TestcomplementaryEarly* operators.

REFERENCES

Cortier, A., D'Ausbourg, B., and Aıt-Ameur, Y., 2007. Formal validation of java/swing user interfaces with the event B method. *In HCI (1),* pages 1062–1071.

Dittmar, A., 2000. More precise descriptions of temporal relations within task models. *In Interactive Applications: Design, Specification, and Verification, 7th International Workshop DSV-IS, Proceedings*, pages 151–168, Limerick, Ireland.

Bouchet, J., Madani, L., Nigay, L., Oriat, C. and Parissis, I. 2007. Formal testing of multimodal interactive systems. *In EIS'2007 Engineering Interactive Systems*, Salamanca, Spain, 36-52.

D'Ausbourg, B., 1998. Using model checking for the automatic validation of user interface systems. *In Design, Specification and Verification of Interactive Systems'98, Proceedings of the Fifth International Eurographics Workshop,* pages 242–260, Abingdon, United Kingdom.

Coutaz, J., Nigay, L., Salber, D., Blandford, A., May, J. and Young, R. M. 1995. Four easy pieces for assessing the usability of multimodal interaction: the care properties. *In INTERACT*, 115-120. Chapman & Hall.

Du Bousquet, L., Ouabdesselam, F., and Richier, J.-L. 1998. Expressing and implementing operational profiles for reactive software validation. *In 9th International Symposium on Software Reliability Engineering,* Paderborn, Germany.

Duke, D. J. and Harrison, M. D., 1993. Abstract interaction objects. *Computer Graphics Forum*, 12(3):25–36.

Paternò, F. and Faconti, G., 1993. On the use of LOTOS to describe graphical interaction. *In HCI'92: Proceedings of the conference on People and computers VII,* pages 155–173, New York, NY, USA. Cambridge University Press.

Bouchet, J., and Nigay, L., 2004. ICARE: a component-based approach for the design and development of multimodal interfaces. *In Extended abstracts of the 2004 Conference on Human Factors in Computing Systems, CHI 2004*, pages 1325–1328, Vienna, Austria, 24 - 29.

Le TL., Nguyen TB, Parissis I., 2013. A New Test Modeling Language for Interactive Applications Based on Task Trees*, In Proceedings of the 4th International symposium on information and communication Technology,* pp.285-293.

Le TL., Nguyen TB., Parissis, I., 2014. A solution of generate test data for interactive applications, *In Proceedings of the 7th National Conference on Fundamental and Applied Information Technology Research (FAIR'7),* pp.134-143.

Madani, L. and Parissis, I. 2009. Automatically testing interactive applications using extended task trees. *J. Log. Algebr. Program*., 78(6):454-471.

Madani, L., Oriat, C., Parissis, I., Bouchet, J., and Nigay, L., 2005. Synchronous testing of multimodal systems: An operational profile-based approach. *In 16th International Symposium on Software Reliability Engineering (ISSRE 2005),* pages 325–334, Chicago, IL, USA, 8-11.

Madani, L. and Parissis, I., 2011. Automatically testing interactive multimodal systems using task trees and fusion models. *In 6th international workshop on Automation of software test (AST '11),* Hawai, USA.

Du Bousquet, L., Ouabdesselam, F. , Richier, J.-L. and Zuanon, N., 1999. Lutess: a specification driven testing environment for synchronous software. *In 21st International Conference on Software Engineering,* pages 267-276. ACM Press.

Musa, J. 1993. Operational Profiles in Software-Reliability Engineering. *IEEE Software*, 14–32.

Palanque P., Bastide R., 1995. Verification of Interactive Software by Analysis of its Formal Specification. *INTERACT'95,* Norway.

Richard K. Shehady and Daniel P. Siewiorek, 1997. A method to automate user interface testing using variable finite state machines. *In FTCS '97: Proceedings of the 27th International Symposium on Fault-Tolerant Computing (FTCS '97*), page 80, Washington, DC, USA, 1997. IEEE Computer Society.

Aıt-Ameur, Y. and Kamel, N., 2004. A generic formal specification of fusion of modalities in a multimodal HCI. *In Rene Jacquart, editor, IFIP Congress Topical Sessions,* pages 415–420. Kluwer, 2004.

Ter Beek, M.H., Faconti, G.P. Massink, M., Palanque, P.A. and Winckler, M., 2009. Resilience of Interaction Techniques to Interrupts: A Formal Model-Based Approach. *In Human-Computer Interaction - INTERACT 2009: Part I of the Proceedings of the 12th IFIP TC 13 International Conference on Human-Computer Interaction (INTERACT'09),* Uppsala, Sweden (T. Gross et al., eds.), Lecture Notes in Computer Science 5726, Springer, Berlin, 494-509.

Paternò, F., Mancini, C., and Meniconi, S., 1997. ConcurTaskTrees: A Diagrammatic Notation for Specifying Task Models. *In Proceedings of the 6th IFIP TC 13 International Conference on Human-Computer Interaction (INTERACT'97),* Sydney, Australia (S. Howard, J. Hammond, and G. Lindgaard, eds.), Chapman & Hall, Boca Raton, 362-369.

Palanque, P., Winckler, M., Ladry, J.-F., Ter Beek, M.H., Faconti, G., and Massink, M., 2009. A Formal Approach Supporting the Comparative Predictive Assessment of the Interruption-Tolerance of Interactive Systems. *In Proceedings of the ACM SIGCHI Symposium on Engineering Interactive Computing Systems (EICS'09),* Pittsburgh, PA, USA (G. Calvary, T.C.N. Graham, and P. Gray, eds.), ACM Press, 211-220.

Kamel, N. and Aït Ameur, Y., 2007. A Formal Model for *CARE Usability Properties Verification in Multimodal HCI. In Proceedings of the IEEE International Conference on Pervasive Services (ICPS'07),* Istanbul, Turkey, IEEE Computer Society, 341-348.

Kamel, N., Aït Ameur, Y., Selouani, S.-A. and Hamam, H., 2008. A formal model to handle the adaptability of multimodal user interfaces. *In Proceedings of the 1st International ICST Conference on Ambient Media and Systems (AMBI-SYS'08),* Quebec, Canada (B. Liang and R.M. Whitaker, eds.).

Mohand-Oussaïd, L., Aït-Sadoune, I., Aït Ameur, Y., and Ahmed-Nacer, M., 2015. A formal model for output multimodal HCI - An Event-B formalization. *Computing 97*, 713-740.