Research Article

Real-Time Extended Interface Automata for Software Testing Cases Generation

Shunkun Yang,¹ Jiaqi Xu,¹ Tianlong Man,¹ and Bin Liu^{1,2}

¹ School of Reliability and Systems Engineering, Beihang University, Beijing 100191, China

² Science & Technology on Reliability & Environmental Engineering Laboratory, Beijing 100191, China

Correspondence should be addressed to Shunkun Yang; yangshunkun@163.com

Received 8 March 2014; Accepted 1 April 2014; Published 29 April 2014

Academic Editor: Guiwu Wei

Copyright © 2014 Shunkun Yang et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Testing and verification of the interface between software components are particularly important due to the large number of complex interactions, which requires the traditional modeling languages to overcome the existing shortcomings in the aspects of temporal information description and software testing input controlling. This paper presents the real-time extended interface automata (RTEIA) which adds clearer and more detailed temporal information description by the application of time words. We also establish the input interface automaton for every input in order to solve the problems of input controlling and interface covering nimbly when applied in the software testing field. Detailed definitions of the RTEIA and the testing cases generation algorithm are provided in this paper. The feasibility and efficiency of this method have been verified in the testing of one real aircraft braking system.

1. Introduction

Interface automata, as a kind of modeling method that describes the interactions between components and the environment, have been applied gradually in many fields in recent years. By using the input assumption and output guarantee, the interface automata express the sequential relationship of the interaction between the system and the environment to describe the operation of the internal system and the action of the environment interface formally. But when it is applied in the software testing field, there are still some inevitable problems including the limited description ability of temporal information, which leads to a lack of accuracy to describe some embedded real-time system, and the neglect of input controlling and interface covering, which result in that some tests which need interface covering find it hard to use this modal conveniently.

Alur and Dill [1] put forward the timed automata method, which bring accurate time description for different state transition actions, and are able to describe the temporal relationship between different state transition actions accurately

and formally by introducing the time language. But time automata pay more attention to internal information of the system instead of the interaction between the system and the environment. The timed interface automata are proposed on the basis of interface automata by De Alfaro et al. [2], which can describe the delay time and the keep time of the state transition process by introducing the two new parameters, time guard and clock variables. But there are also disadvantages of the timed interface automata. For some real-time system, the input/output signals may be periodic. However, the timed interface automata cannot provide a formalized description for the constraint and the change of a periodic variable.

Besides, the automata method also has some defects when it is applied to the software testing field. As Raffelt et al. proposed the dynamic testing method based on learning automata, as a result of the automata's learning features, the method was combined with many mature learning algorithms to make dynamic tests. At the same time, an application of web services testing is provided [3]. Its limitation is that the method is focused on network services and internal

structure of cellular automata; it may not be suitable for embedded software; Nielsen and Skou proposed a real-time black box conformance testing method for uncertain systems based on timed automata model, and the experimental results show its superiority [4]. But because it is based on the timed automata, the ability of temporal information description of input and output is very limited; a new testing method has been proposed by Liangming et al. based on the interface automata model, it divides state sequences into different sections [5], and the covering algorithm is given. But it focuses on path covering of the automata model, while the interface covering cannot be guaranteed. Thus, when the traditional interface automata method is applied in software testing, especially in the interface testing of real-time system, there are some limitations. That is, the period information of the input and output cannot be described. Besides, it is not convenient to control the input value including its temporal information (especially to automatic control).

To solve the problems above, this paper puts forward the RTEIA on the basis of interface automata and timed automata; its main contributions are as follows.

- (1) By applying the time words to describe the automaton of input/output action, it provides clearer and more detailed temporal information description for the automata model, which contains both the moment and the period information. With the values and the temporal information of input/output packaged, unified actions and descriptions are facilitated.
- (2) By establishing input automata for each input interface to construct internal structure of input automata, this method nimbly solved the problems of input control and interface covering when applied to the software testing field.

The structure of the rest of this paper is as follows. Section 2 focuses on the theory of interface automata and related research background. Specific definition of RTEIA is given in Section 3; Section 4 mainly expounds the test case generation based on this method and presents a generation algorithm which is verified and discussed with an instance. Conclusions and outlooks for future work are in the last section.

2. Preliminary

In software testing, usually there is a very complex interaction between the system and the environment. For the convenience of testing modeling and automatically generating test cases, formalized description of the interface is necessary. The following lists some formal languages that describe the interface interaction based on automata theory.

2.1. Interface Automata. De Alfaro and Henzinger put forward the interface automata in 2001 [6]. Interface automata's main features are as follows: (1) "positive method" is used to define the interface compatibility issues under different environment rules and (2) the game idea is a term used to describe the process of the problem solving.

Unlike the traditional method which records and gathers input domain and output domain, interface automata also describe the temporal characteristics of the input and output, integrate the input assumption and output guarantee, and provide formalized description. Output guarantee reflects the order of the various external services request from the interface which denotes the assumptions made to the environment. Input assumption reflects the order of the services provided to the external by the interface which denotes a description of the automata's own behavior.

Optimistic method is a kind of compatibility judging processing method of the interface and environmental relationship; the traditional method (or named pessimistic methods) is that as long as there is a kind of external environment to make illegal status be achieved, it considers the combined interface incompatible. But the optimistic method believes that as long as there is a kind of external environment which can make the combined interface function normally, it considers that they are compatible. Its advantage is to make the number of interface states greatly reduce after the combination, which makes the interface automata as a lightweight formalized system.

Game ideas are as follows. Interface automata finally attribute the operation of the interface and the environment to a game; the environment is to find a strategy to meet all interface input assumptions as far as possible. If environment wins the game, we can consider that interfaces are compatible; otherwise, they are incompatible.

With the developing of the theory, in addition to the main application to verify components compatibility of the modular development mode, interface automata method was applied to a variety of related fields gradually; that is, because interface automata model's description of the state transition is similar to the web service user transferring and link jumping, the interface automata theory was applied to the description of building web services by Li et al. [7], and the relevant verification algorithm was given; Zhang et al. [8] applied the interface automata to analyze the usability of composite web services building, and then they proposed hybrid interface automaton based on the interface automata method. The algorithm used to check accessibility, well-formedness, and compatibility for the network physical system is given as well. Due to the lightweight and generality of interface automata model, Karimpour et al. [9] combined it with the queuing theory and applied it to the modular software system performance prediction field.

2.2. Timed Automata. Alur and Dill put forward a kind of automata method that contains temporal information named timed automata [1]. Timed automata are a kind of formalized model based on the finite state automata extended with clock variables to describe the continuous changes of the time and analyze the behavior of the real-time systems. An important properties of this theory lies in its ability to detect the reachability of any state in the timed automata.

Timed automata introduced the time words based on the finite state automata and added clock variables into the time words to make the input and output information contain the

temporal information. Timed automata use logical relationship to describe the temporal relationship between different input/output actions by the formalized language. But this method focuses on the internal state transition actions information only and does not take consideration of the interactive actions between the system and the environment.

Bengtsson and Yi did a further study of timed automata on semantics and the implementation algorithm and put forward the timed automata semantic system that can realize the model from the theory to the application. The corresponding algorithm and verification tools were provided as well [10]. Timed automata were applied to the real-time distributed system verification by Wen et al., based on constructing timed automata for controller area network, tasks, and the operating system kernel, respectively, to analyze the system and to verify its validity by experiments [11]. Calder et al. extended the interface automata with another way of temporal information and proposed RTIA real-time interface automata for scenario-based timing specifications [12]. Jin et al. expanded the method of [11] to solve the timed automata modeling problems of complex real-time systems and introduced the hierarchical timed automata (HTA) by partitioning the component level and hiding and expanding execution of corresponding chain to make the description of multilayered components more accurate [13]. Some researches were started in the timed automata model validation and the mutual transformation among different modeling languages, and so forth, [14–17].

2.3. Timed Interface Automata. De Alfaro et al. extended the interface automaton method and proposed the timed interface automata [2].

Interface automata method introduces time constraints in order to describe the real-time system better. It adds a clock variable for interface automata; each state has a predicate made of a set of time variables called time condition. The time condition restrained the occurring conditions of the input/output and also introduced the guard which is used to restrain the input/output that shall meet the conditions.

Because the input/output actions themselves are not time-consuming, the state transition does not require any extra time. So if the executions of the input/output were repeated infinitely, the automata time will stop forever, which is obviously inconsistent with the facts. This action is known as Zeno behavior. To this end, the authors put forward a well-formedness criterion which rules that, in any reachable state, time interface automata should guarantee that the time variable of input/output actions would never stop.

In order to apply the timed automata to web services field, based on the concurrent characteristics of web services, Esmailsabzali et al. put forward Interface automata with complex actions. By introducing complex behaviors that do not interlace with other components for interface automata, this method can describe the characteristics of the concurrent systems better [18]. In view of the pervasive systems, Calder et al. put forward the pervasive interface automata [12], according to the characteristics of the pervasive system that context is closely linked. They expand the concept of input

and output of interface automata theory and introduce the function calling and being called, so that the pervasive interface automata can be described better. Raclet et al. combined the interface automata with modal specification method, summarized the predecessors' combination method, and put forward the modal interface automata [19]. Then, Krka and Medvidovic collated and improved modal interface automata [20].

The methods above are all trying to describe the interaction of the interface including temporal information, but there are still drawbacks. For instance, the timed automata lack the view of the interface and the event description of the timed automata relies on state transition logic relationship. The timed interface automata lack a complete description of period information; besides, the ability of input controlling is scanty. Therefore, this paper presents the RTEIA, which adds clearer and more detailed temporal information description for the automata model. And, by the establishment of input interface automaton for every input interface, it also nimbly solve the input controlling and interfaces covering problems when applied in the test field.

3. The Real-Time Extended Interface Automata

To make up for the existing problem of interface automata model such as incomplete description temporal information and the lack of considering of the input controlling, we introduce time words and input automata for interface automata as the RTEIA.

3.1. Time Language. Considering the defect of the interface automata when it is used to describe a real-time system, we need a language that can describe the temporal characteristics of the system input and output parameters. Therefore, we introduce and improve the concept of time language from the timed automata [1] to describe the temporal character of the real-time system. Time language contains 3 time parameters; as in the case of the dense-time model, the set of nonnegative real numbers, R , is chosen as the time domain. Each input/output σ is coupled with two temporal parameters time sequence τ and period parameter λ as defined below.

Definition 1. A time sequence $\tau = \tau_1, \tau_2, \dots$ is an infinite sequence of time values $\tau_i \in R$ with $\tau_i > 0$, satisfying the following constraints.

- (1) Monotonicity: τ increases strictly monotonically; that is, $\tau_i < \tau_{i+1}$ for all $i \geq 1$.
- (2) Progress: for every $t \in R$, there is some $i \geq 1$ such that $\tau_i > t$.

Definition 2. A period parameter $\lambda = \lambda_1, \lambda_2, \dots$ is an infinite sequence of time values $\lambda_i \in R$ with $\lambda_i \geq 0$.

τ of an input/output σ , λ is used to express the period information. A time word over an alphabet Σ is a pair (σ, τ, λ) where $\sigma = \sigma_1\sigma_2\dots$ is an infinite word over Σ , τ is a time

FIGURE 1: An example of time language.

sequence, and λ denotes the period information. Timed language over Σ is a set of timed words over Σ .

Let one consider some examples of timed languages.

Example 3. Let the alphabet be $\Sigma = (a, b)$ and its time language $L_i \rightarrow (\sigma, \tau, \lambda)$.

To describe: there is no b after time 3.2, and its period is 300. Its time language can be as

$$L_1 = \{(\sigma, \tau, \lambda) \mid \forall i. ((\tau_i > 3.2) \rightarrow (\sigma_i = b, \lambda_i = 300))\}. \quad (1)$$

To describe: a and b are alternate, and, for the successive pairs of a and b , the time difference between a and b keeps increasing, and the period keeps decreasing in the same time. Its time language can be as

$$L_2 = \{((ab)^\omega, \tau, \lambda) \mid \forall i. ((\tau_{2i} - \tau_{2i-1}) < (\tau_{2i+2} - \tau_{2i+1}), \lambda_{2i} > \lambda_{2i+2}, \lambda_{2i-1} > \lambda_{2i+1})\}. \quad (2)$$

To describe: a and b are alternate, the period of a is 100, the period of b is 200, and b should be later than a in 2, which is shown in Figure 1.

Its time language can be as

$$L_3 = \{((ab)^\omega, \tau, \lambda) \mid \forall i. (\tau_{2i} < (\tau_{2i-1} + 2), \lambda_{2i} = 100, \lambda_{2i+1} = 200)\}. \quad (3)$$

3.2. The RTEIA. With the time words as the tool for describing the temporal information, we define the RTEIA based on the interface automata method. The RTEIA packs the temporal information to describe the various actions of the automata clearly.

It is defined as follows.

Definition 4. The RTEIA P is a quintuple; $P = \langle V_P, V_P^{\text{init}}, A_P^I, A_P^O, T_P \rangle$, where

- (1) V_P is a set of states;
- (2) $V_P^{\text{init}} \subseteq V_P$ is the initial set of states, and V_P^{init} contains at least one state; if $V_P^{\text{init}} = \emptyset$, one says P is empty;

- (3) A_P^I, A_P^O are mutually disjoint set of input, output actions; one denotes by A_P the set of all actions, $A_P = A_P^I \cup A_P^O$;

- (4) $T_P \subseteq V_P \times A_P \times V_P$ is a set of transitions.

Definition 5. Set of input/output actions A_P^I, A_P^O are described by $a = (\sigma, \tau, \lambda)$, where

- (1) σ is the value of the input/output;
- (2) τ is the time sequence of the input/output;
- (3) λ is the period information of the input/output.

Definition 6. A controllable operation segment of the RTEIA P is the alternative finite sequence of states and actions: $v_0 a_0 v_1 a_1 v_2 \cdots v_n$, where $(v_i a_i v_{i+1}) \in T_P, 0 < i \leq n$.

One can use an example to explain how a RTEIA describes a system.

Example 7. Figure 2 shows a simple RTEIA, according to the following definition.

RTEIA P is as follows.

- (1) the set of states $V_P = \{v_0, v_1, v_2, v_3\}$.
- (2) the initial set of states $V_P^{\text{init}} = \{v_0\}$.
- (3) the set of input actions $A_P^I = \{a_0, a_1, a_3\}$ and the set of output actions $A_P^O = \{a_2, a_4\}$. The input/output actions a can be described by time words:
 - (a) $a_0 = (\sigma_0, \tau_0 = 0, \lambda_0 = 20)$,
 - (b) $a_1 = (\sigma_1, \tau_1 = 19, \lambda_1 = 12)$,
 - (c) $a_2 = (\sigma_2, \tau_2 = 22, \lambda_2 = 9)$,
 - (d) $a_3 = (\sigma_3, \tau_3 \leq 30, \lambda_3 = 90)$,
 - (e) $a_4 = (\sigma_4, \tau_4 = 31, \lambda_2 = 80)$.
- (4) There are a number of transition chains, like $t_1 = (v_0 a_0 v_1 a_1 a_2 v_2 a_4 v_3)$ and $t_2 = (v_0 a_0 v_1 a_1 a_2 v_2 a_3 v_0 a_0 v_1 a_1 a_2 v_2 a_4 v_3)$, $t_1, t_2 \in T_P$.

Due to the application of time words to describe the input/output actions, we can choose different transition chain to describe the temporal information of the automaton interface interaction.

3.3. Input Control of the RTEIA. For the purpose of applying the RTEIA to software testing, we need to control the input in order to generate test cases. So, we treat the input a^I as a RTEIA I_i . The input automata can cross-link with the device under test and other input source devices, to realize the feedback and control of the input.

Definition 8. One defines the influence interface action as a constraint input, denoted by $\rightarrow a$. In the graphic, dotted arrow is used to denote the constraint input.

Example 9. We can establish the RTEIA model for the two input a_0, a_1 from the automata P in Example 7, which is shown in Figures 2 and 3.

FIGURE 2: A simple example of the RTEIA P . FIGURE 3: Input automata I_0 .

RTEIA I_0 is as follows.

- (1) the set of states $V_P = \{u_0, u_1, u_2, u_3\}$.
- (2) the initial set of states $V_P^{\text{init}} = \{u_0\}$.
- (3) The set of input actions $A_P^I = \{\rightarrow a_2, \rightarrow b_1, b_2\}$ and the set of output actions $A_P^O = \{a_2\}$.
- (4) There are a number of transition chains, like $t_3 = (u_0 b_0 u_1 a_2 u_2 a_0 u_0)$ and $t_4 = (u_0 b_0 u_1 b_2 u_3 a_0 u_0)$, $t_3, t_4 \in T_P$.

RTEIA I_1 is as follows.

- (1) the set of states $V_P = \{w_0, w_1, w_2\}$.
- (2) the initial set of states $V_P^{\text{init}} = \{w_0\}$.
- (3) the set of input actions $A_P^I = \{c_1, c_2\}$ and the set of output actions $A_P^O = \{a_1, b_1\}$.
- (4) There are a number of transition chains, like $t_5 = (w_0 c_1 w_1 a_1 w_0)$ and $t_6 = (u_0 b_0 u_1 b_2 u_3 a_0 u_0)$, $t_5, t_6 \in T_P$.

FIGURE 4: Input automata I_1 . FIGURE 5: Input automata I_0 .

We can find that the feedback of the input a_0 has been described by putting automata P 's output a_2 as input automata I_0 's input. Input automata I_1 are also cross-linked with input automata I_0 . Furthermore, we can control the input of automata P by the design of the internal structure of the input automata I_0I_1 and the manual adjustment of inputs b_0 , c_0 , and c_1 . For example, in different state transition chains like t_3 and t_4 , we can get different a_0 by controlling a_2 and b_1 , which are the input of input automata I_0 .

There are mainly two characteristics of the RTEIA. Firstly, it uses time words to describe the temporal information (contains both time sequence and period information) of the input/output actions of the automata; secondly, it treat the input interface as independent automata, which is used to control the input by the cross-link of the automata and the input automata.

4. Test Cases Generation Based on the RTEIA

Because of the feature of the RTEIA, we apply it to the software testing field. When it is used to describe the software

model, we can generate test cases automatically by some algorithms.

4.1. Input Automata Controlling. In the interface test of real-time system, we can establish automata model for each input as input automata, using the method introduced in Section 3.3, in order to control the input and realize interface covering by design of the input automata's internal structure.

Common input automata for software interface test are shown in Figure 5.

The input automata I_0 are as follows (Figure 4).

- (1) the set of states $V_P = \{u_0, u_1, u_2, u_3, u_4, u_5, u_6, u_{41}, u_{42}, u_{51}, u_{52}, u_{61}, u_{62}\}$.
- (2) the initial set of states $V_P^{\text{init}} = \{u_0\}$.
- (3) the set of input actions $A_P^I = \{\rightarrow a_1, b_0, b_1, b_2\}$ and the set of output actions $A_P^O = \{a_0\}$.

- (4) There are a number of transition chains, like $t_1 = (u_0 a_1 b_0 a_0)$ and $t_2 = (u_0 a_1 u_1 b_0 u_3 b_1 u_5 b_2 u_{52} a_0)$, $t_1, t_2 \in T_P$.

In the input automata, u_0 denotes the initial state of the automata and $\rightarrow a_1$ denotes a constraint input (coming from the main automata or another input automata) of the input automata; it means that this input is restricted by the constraint from other part; u_1 denotes the start state. By inputting different b_0, b_1, b_2 , we can make the input automata I produce normal/abnormal output. u_2 denotes normal state, which will produce a normal test case a_0 , and u_4, u_5, u_6 denote value abnormal state, time sequence abnormal state, and period information abnormal state, respectively. Then, a_0 is produced correspondingly by controlling the input b_2 to make the abnormal value larger or littler. The input automata aim to state all possibilities of the input information, which could be expected or unexpected.

For instance, the state transition chain $t_2 = (u_0 a_1 u_1 b_0 u_3 b_1 u_5 b_2 u_{52} a_0)$ denotes a time sequence abnormal a_0 , and its time value is too large, which means it will be inputted later than the normal one. The example is just a simple input automata model in logic. We can design a more complex internal structure of the input automata to express different test strategy in practical application.

4.2. Test Cases Generation. We simplify the method of software interface covering by the input automata introduced in Section 4.1. When the input automata are finished with the path covering, the interface covering, which means the coverage of the interface status while test cases had been executed, of the software under test is able to be guaranteed too. We present an algorithm to generate test cases for the input automata.

Definition 10. A controllable operation segment $Exec(v_0 a_0 v_1)$ is a state action $(v_0 a_0 v_1)$ from a state transition chain, where $head(Exec(v_0 a_0 v_1)) = v_0$, $tail(Exec(v_0 a_0 v_1)) = v_1$.

We apply a depth-first algorithm to cover these input automata. It will start with the ENTRY. When a controllable operating segment is searched and marked, a new search will start from the tail of the last controllable operating segment. This step will be repeated constantly.

The steps of Algorithm 1 can be summarized as follows.

Step 1. When $i = 1$, find a controllable operation segment that $head(Exec(i)) = \text{ENTRY}$.

Step 2. If all the controllable operation segments have been found or the coverage has been 100%, then go to Step 6; else, go to Step 4.

Step 3. Search for the next unused controllable operation segment which has tail of the last one as the head state; if found, then $i = i + 1$ and go to Step 2; else, go to Step 4;

Step 4. Let $j = i$; record the current sequence number of tested controllable operation segment. Record this operation

chain as $Case(k)$ which starts with ENTRY and ends with END. Run the test and refresh the coverage.

Step 5. Search for another controllable operation segment which has the same head state with $Exec(i)$ and is not marked. If found, $i = j + 1$ and go to Step 2; else, $i = i - 1$ and repeat Step 5. If all the controllable operation segments have marked, then go to Step 6.

Step 6. Display result and finish the algorithm.

We can get all kind of the input time word a using this algorithm in order to realize the interface covering of the software under test.

Then, make these time words as the input fields. Choose input groups from the input fields as the input of the testing using some intelligent algorithms such as genetic algorithm or ant colony algorithm. Finally, generate the test cases set by optimizing of the feedback of the coverage continuously aiming at both the interface covering and the path covering.

4.3. Case Study. We apply the RTEIA and the test case generation algorithm to an interface test of an aircraft braking system. The structure of the system is shown in Figure 6.

Automata P are an aircraft braking system, which has 4 inputs $a_0 a_2 a_3 a_4$, corresponding to 4 input automata $I_0 I_1 I_2 I_3$. There are 3 outputs $a_1 a_5 a_6$; in the same time, a_1 is an input $\rightarrow a_1$ of input automata I_1 which functions as a feedback.

Establish the RTEIA model, which is shown in Figure 7.

The meaning of each parameter is shown in Table 1.

The unit of τ and λ is second and the units of σ are dimensionless, Newton, meter per second, round per minute, and millivolt.

Taking an example of the input a_2 , establish the input automata I_1 , which is shown in Figure 8.

The meaning of the parameters is similar to Figure 5. The constraint input $\rightarrow a_1$ means a feedback that, only when the brake system BIT is successful, can the process begin to work. $u_2 u_7$ are two kind of normal state and u_4, u_5, u_6 are value abnormal state, time sequence abnormal state, and period information abnormal state, respectively. b_2 controls whether we will get a larger or a littler output a_2 .

Using Algorithm 1, we can realize the path covering of the input automata and get a test sequence controlled by $b_0 b_1 b_2$. By running the test sequence, finally, we are able to get the input field of a_2 as follows:

$$A_2 = \{(\sigma_2 = 0, \tau_2 = 10, \lambda_2 = 10), (\sigma_2 = 1, \tau_2 = 10, \lambda_2 = 10), (\sigma_2 = -36, \tau_2 = 10, \lambda_2 = 10), (\sigma_2 = 100, \tau_2 = 10, \lambda_2 = 10), (\sigma_2 = 0, \tau_2 = 16, \lambda_2 = 10), (\sigma_2 = 1, \tau_2 = 17, \lambda_2 = 10), (\sigma_2 = 0, \tau_2 = 1, \lambda_2 = 1), (\sigma_2 = 1, \tau_2 = 10, \lambda_2 = 29)\}.$$

Begin:

$V_0 = ENTRY, i = j = k = 1, Cov = 0$; // j denotes all the controllable operating segments that have been searched already, I denotes the current segments, k denotes the produced test cases, Cov denotes the coverage.

While ($j \leq m$ and $Cov < 100\%$)

{

If (find a unmarked $Exec(j)$ and $head(Exec(j)) == V_0$ and $i > 0$)

{

Record $Exec(j)$ and mark it;

If ($tail(Exec(j)) == END$)

{

Record these execution $Case(k) = Exec(ENTRY \dots) \dots \dots Exec(\dots END)$;

$k = k + 1$;

Run $Case(K)$ and refresh Cov ;

}

$V_0 = tail(Exec(j))$;

$j = j + 1$;

$i = j$

}

$V_0 = head(Exec(j))$;

$I = i - 1$;

}

End.

ALGORITHM 1

TABLE 1: Parameters of the aircraft braking system.

Parameter	Meaning
v_0	State of system BIT
v_1	State of a failed system BIT
v_2	State of a successful system BIT, system startup
v_3	State of taking off
v_4	State of landing
v_5	State of stop turning braking: the braking system is used to completely stop the idling of the landing gear tires after taking off
v_6	State of keeping in land: the braking system is used to lock the landing gear tires after landing
v_7	State of normal braking
v_8	State of antilock braking
a_0	Input: command of the brake system startup
a_1	Output: the result of the system BIT
a_2	Input: the information of takeoff/landing
a_3	Input: the gear's load information, to determine whether the plane has completely off the ground during takeoff
a_4	Input: the gear wheel speed information, to determine when the plane landed at rest, a normal speed, or a too fast speed
a_5	Output: problem report of the system BIT
a_6	Output: the gear hydraulic circuit signal, to control the force created by the hydraulic institute and implement different braking friction strength.

Analogously, we can get the input fields of $a_0 a_3 a_4$:

$$A_0 = \{(\sigma_0 = 1, \tau_0 = 0, \lambda_0 = 0),$$

$$(\sigma_0 = -5, \tau_0 = 0, \lambda_0 = 0),$$

$$(\sigma_0 = 32, \tau_0 = 0, \lambda_0 = 0),$$

$$(\sigma_0 = 1, \tau_0 = 22, \lambda_0 = 0),$$

$$(\sigma_0 = 1, \tau_0 = 0, \lambda_0 = 90)\},$$

$$A_3 = \{(\sigma_3 = 0, \tau_3 = 100, \lambda_3 = 0),$$

$$(\sigma_3 = -9, \tau_3 = 100, \lambda_3 = 0),$$

$$(\sigma_3 = 12, \tau_3 = 100, \lambda_3 = 0),$$

FIGURE 6: The structure of the aircraft braking system.

$$\begin{aligned}
 & (\sigma_3 = 0, \tau_3 = 120, \lambda_3 = 0), \\
 & (\sigma_3 = 0, \tau_3 = 178, \lambda_3 = 0), \\
 & (\sigma_3 = 0, \tau_3 = 100, \lambda_3 = 23)\}, \\
 A_4 = \{ & (\sigma_4 = 0, \tau_4 = 16, \lambda_4 = 1), \\
 & (\sigma_4 = 20000, \tau_4 = 16, \lambda_4 = 1), \\
 & (\sigma_4 = 43000, \tau_4 = 16, \lambda_4 = 1), \\
 & (\sigma_4 = 30000, \tau_4 = 16, \lambda_4 = 1), \\
 & (\sigma_4 = 0, \tau_4 = 10, \lambda_4 = 1), \\
 & (\sigma_4 = 20000, \tau_4 = 10, \lambda_4 = 1), \\
 & (\sigma_4 = 43000, \tau_4 = 10, \lambda_4 = 1), \\
 & (\sigma_4 = 0, \tau_4 = 22, \lambda_4 = 1), \\
 & (\sigma_4 = 20000, \tau_4 = 22, \lambda_4 = 1), \\
 & (\sigma_4 = 43000, \tau_4 = 22, \lambda_4 = 1), \\
 & (\sigma_4 = 0, \tau_4 = 16, \lambda_4 = 0.5), \\
 & (\sigma_4 = 20000, \tau_4 = 16, \lambda_4 = 0.5), \\
 & (\sigma_4 = 43000, \tau_4 = 16, \lambda_4 = 0.5), \\
 & (\sigma_4 = 0, \tau_4 = 16, \lambda_4 = 8), \\
 & (\sigma_4 = 20000, \tau_4 = 16, \lambda_4 = 8), \\
 & (\sigma_4 = 43000, \tau_4 = 16, \lambda_4 = 8)\}.
 \end{aligned} \tag{5}$$

After getting the input fields of all inputs of the RTEIA, we use some classical intelligent algorithms such as genetic algorithm to reduce the useless cases and to improve the testing efficiency. The sets of input fields are the inputs of the algorithm; the coverage of the automata could be used as the feedback indicator; both the interface covering and the path covering could be selected as the multiple targets of the

algorithm. Then, the test set will be got. One typical testing sequence is as follows:

$$\begin{aligned}
 \text{TestCase1} = \text{ENTRY} & \xrightarrow{a_0?=(\sigma_0=1, \tau_0=0, \lambda_0=0)?} v_0 \\
 & \xrightarrow{a_1!=(\sigma_1=1, \tau_1 \leq 15, \lambda_0=10)!} v_2 \\
 & \xrightarrow{a_2?=(\sigma_2=1, \tau_2=17, \lambda_2=10)?} v_4 \\
 & \xrightarrow{a_4?=(\sigma_4=43000, \tau_4=16, \lambda_4=8)?} v_8 \\
 & \xrightarrow{a_6!=(\sigma_6=2, \tau_6 \leq 16, \lambda_6=0)!} \text{END}.
 \end{aligned} \tag{6}$$

This testing sequence means that, firstly, the system receives a startup command input a_0 , which is a normal input; then the system trans to v_0 which denotes the BIT state, and BIT runs successfully and gets a normal output a_1 ; then, system receives the landing information input a_2 ; this input is a time sequence abnormal input; its time is later than the required time in 1 second; then the system trans to state v_4 which denotes the landing state and receives the gear wheel speed input a_4 ; the speed is 43000 which belongs to the fast landing. This input is a period information abnormal input; its period is amplified to 8; then the system trans to state v_8 , outputs signal a_6 of the gear hydraulic circuit to reduce the braking friction strength and prevent the gear locking; then the system ends.

By using the RTEIA to describe the interface testing of this above real-time system with temporal information accurately, testing cases have been generated automatically to improve the testing efficiency successfully.

5. Discussions

In terms of the functioning features, compared with the related interface automata model, such as the interface automata, the timed automata, and the timed interface automata, because of the introducing and improving of the time words, the RTEIA makes the description of temporal information more complete; besides, for establishing input automata, the RTEIA is more powerful when it is applied in the software testing field. The contrast is shown in Table 2.

The RTEIA has superiority in the description of the temporal information and input control in the field of software testing. In terms of the description of temporal information, the interface automata do not contain temporal information; it is difficult to describe periodic signal for the timed automata and the timed interface automata, though there is some improvement. In terms of the judgment of combination and complexity, because the RTEIA concentrates on the field of software testing, it shows different features from the traditional methods. In terms of the input control, unlike the traditional methods, the RTEIA treats every input as independent automata, by which it can control the input conveniently. In terms of the application in software testing field, the RTEIA is more suitable than some other related researches about interface automaton and timed interface

TABLE 2: Comparing RTEIA with other automata models.

Model type	Description of temporal information	Judgment of combination and complexity	Input control	Being applied in the software testing field
Interface automata	No	Yes	No	Yes, focuses on path covering and does not contain temporal information
Timed automata	Yes, can describe time information by logic relationship	No	No	Yes, focuses on collating logic information and path covering
Timed interface automata	Yes, but cannot describe period information	Yes	No	No
RTEIA	Yes, can describe temporal information by time words	No	Yes, can control the input by establishing the input automata.	Yes, can describe temporal information completely and contain interface covering

automata because of their limited temporal description, the interface covering guarantee, and so forth.

In terms of the algorithmic complexity, input interface automata can be represented by a directed graph. The main cost of Algorithm 1 is in the searching for the controllable operation segments corresponding to the distinguishable states. Therefore, if the interface automata are expressed in the form of adjacency matrix, the algorithm complexity would be $O(n^2)$, where n denotes the state number of the longest execution segments; if the interface automata are expressed in the form of adjacency list, the algorithm complexity would be $O(n + m)$, where m denotes the number of the execution segments.

The RTEIA solved the problems of traditional automata modeling language successfully without any increase in cost.

6. Conclusions

This paper proposes a new formalized modeling language named the real-time extended interface automata to describe the temporal information of the interface between the system and the environment. Based on the interface automata, the RTEIA solves the problems of the traditional automata methods by introducing time words to describe the temporal information (including the period information) of input/output actions. Meanwhile, the RTEIA is able to control the input conveniently and to realize both the path and interface covering when applied in software testing field by establishing the input automata. At last, we provided relevant algorithm and process and verified its feasibility and effectiveness by applying this method in the testing of one real aircraft braking system.

Because the RTEIA is a new method, there are still some open questions left to be solved. As part of the future work, we will investigate the ways to describe the feedback and the constraints between the input automata, and so forth. Besides, the RTEIA will be tried to be applied in more cases and fields.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgments

This research is partially supported by the Aeronautics Science Foundation of China (no. 2011ZD51055), Science and Technology on Reliability and Environmental Engineering Laboratory (no. 302367), the National Pre-Research Foundation of China (no. 51319080201), and YETP1121. The authors also would like to express their thanks to the anonymous reviewers whose comments would improve this paper considerably.

References

- [1] R. Alur and D. L. Dill, "A theory of timed automata," *Theoretical Computer Science*, vol. 126, no. 2, pp. 183–235, 1994.
- [2] L. De Alfaro, T. A. Henzinger, and M. Stoelinga, "Timed interfaces," in *Embedded Software*, pp. 108–122, Springer, Berlin, Germany, 2002.
- [3] H. Raffelt, M. Merten, B. Steffen, and T. Margaria, "Dynamic testing via automata learning," *International Journal on Software Tools for Technology Transfer*, vol. 11, no. 4, pp. 307–324, 2009.
- [4] B. Nielsen and A. Skou, "Automated test generation from timed automata," in *Tools and Algorithms for the Construction and Analysis of Systems*, pp. 343–357, Springer, Berlin, Germany, 2001.
- [5] L. Liangming, L. Lei, W. Zhijian, and T. Yelong, "Research on interface automata testing," in *Proceedings of the International Conference on Computer Science and Software Engineering (CSSE '08)*, pp. 743–746, IEEE, December 2008.
- [6] L. de Alfaro and T. A. Henzinger, "Interface automata," in *Proceedings of the 8th European Engineering Conference (ESEC) and 9th ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE-9)*, vol. 26, no. 5, pp. 109–120, ACM, September 2001.

- [7] J. Li, S. Chen, L. Jian, and H. Zhang, "A web services composition model and its verification algorithm based on interface automata," in *Proceedings of the 10th IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom '11)*, pp. 1556–1563, November 2011.
- [8] Y. Zhang, X. Yu, and T. Zhang, "Usability analysis of Web Service behavior based on interface automata," in *Proceedings of the 3rd IEEE International Conference on Software Engineering and Service Science (ICSESS '12)*, pp. 135–138, 2012.
- [9] J. Karimpour, A. Isazadeh, and H. Izadkhah, "Performance prediction of component based software systems using interface automata," in *Proceedings of the CSI International Symposium on Computer Science and Software Engineering (CSSE '11)*, pp. 69–76, June 2011.
- [10] J. Bengtsson and W. Yi, "Timed automata: semantics, algorithms and tools," in *Lectures on Concurrency and Petri Nets*, pp. 87–124, Springer, Berlin, Germany, 2004.
- [11] Y. Wen, J. Wang, and Z. Qi, "Bridging refinement of interface automata to forward simulation of I/O automata," in *Formal Methods and Software Engineering*, pp. 259–273, Springer, Berlin, Germany, 2004.
- [12] M. Calder, P. Gray, A. Miller, and C. Unsworth, "An introduction to pervasive interface automata," in *Formal Aspects of Component Software*, pp. 71–87, Springer, Berlin, Germany, 2012.
- [13] X. Jin, M. Huadong, and Z. Gu, "Real-time component composition using hierarchical timed automata," in *Proceedings of the 7th International Conference on Quality Software (QSIC '07)*, pp. 90–99, October 2007.
- [14] M. T. B. Waez, J. Dingel, and K. Rudie, "A survey of timed automata for the development of real-time systems," *Computer Science Review*, vol. 9, pp. 1–26, 2013.
- [15] B. Bérard, S. Haddad, and M. Sassolas, "Interrupt timed automata: verification and expressiveness," *Formal Methods in System Design*, vol. 40, no. 1, pp. 41–87, 2012.
- [16] S. Nakano and S. Yamaguchi, "An efficient translation method from timed Petri Nets to timed automata," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. 95, no. 8, pp. 1402–1411, 2012.
- [17] R. Gómez, "Model-checking timed automata with deadlines with Uppaal," *Formal Aspects of Computing*, vol. 25, no. 2, pp. 289–318, 2013.
- [18] S. Esmailsabzali, N. A. Day, and F. Mavaddat, "Interface automata with complex actions: limiting interleaving in interface automata," *Fundamenta Informaticae*, vol. 82, no. 4, pp. 465–512, 2008.
- [19] J.-B. Raclet, E. Badouel, A. Benveniste, B. Caillaud, A. Legay, and R. Passerone, "Modal interfaces: unifying interface automata and modal specifications," in *Proceedings of the 7th ACM International Conference on Embedded Software*, pp. 87–96, October 2009.
- [20] I. Krka and N. Medvidovic, "Revisiting modal interface automata," in *Proceedings of the IEEE Formal Methods in Software Engineering: Rigorous and Agile Approaches (FormSERA '12)*, pp. 30–36, 2012.