An Approach to Concurrent TTCN Test Generation

BI Jun (毕 军) and WU Jianping (吴建平)

Department of Computer Science and Technology, Tsinghua University
Beijing 100084, P.R. China

Received August 10, 1998; revised November 20, 1998.

Abstract The basis of distributed system conformance testing is to test the conformance of each entity with its standard. This paper addresses the approach to entity conformance testing based on concurrent TTCN. First a preliminary framework for entity conformance testing is introduced and a specification model CEBE is presented. Then a test generation method, which could directly derive concurrent TTCN test suite from CEBE, is proposed.

Keywords conformance testing, distributed system, CEBE, concurrent TTCN

1 Introduction

The entity in distributed systems is more complex and general than traditional peer-to-peer protocol entity[1]. An entity in distributed systems could be either a system or a single object in a system. It is associated with a set of interaction points and a set of concurrent behaviors at these interaction points. In this paper, we give a specification model called Concurrent External Behavior Expression (CEBE) to specify the entity and propose a new approach to generate concurrent TTCN (denoted as C-TTCN in this paper) test suite from CEBE. CEBE could directly specify concurrent external behaviors (both I/O actions and data) at different interaction points of an entity in distributed systems and reduce internal states, actions and data. Therefore it is an ideal model to formally specify the entity in a complex distributed system and to generate (combine data flow and control flow) C-TTCN test suite. We developed a protocol integrated test system PITS and an automatic test generation tool TUGEN. These methods have been applied in real test activities.

2 A Framework for Entity Conformance Testing

2.1 Test Architecture

In [2–4], some problems of distributed system conformance testing have been studied. However, most of these studies didn't give a general framework or approach. In the universe of discourse of testing, the behaviors at interaction points are very complex in a real distributed system. To reduce the complexity, a distributed system has to be decomposed through a process of abstraction. Given several abstract levels (viewpoints), in each viewpoint, the complexity of distributed environment and concurrent behaviors will be decreased. In each viewpoint, we could test concurrent behaviors of an entity at interaction points associated with this entity. After each entity is tested, we could give the conformance verdict of the whole-distributed system.

Concurrent TTCN is an extension of test suite notation TTCN, and it can specify behaviors occurring concurrently. In a test suite specified by C-TTCN, the test system consists of one MTC (Main Test Component) and zero or more PTCs (Parallel Test Components). Each TC can be related with one or more PCOs (Points of Control and Observation). The Information exchange between TCs and SUT (System under Test) is carried out through these PCOs, while the information exchange between TCs is performed through CPs (Coordinated Points) by using CMs (Coordinated Messages). CMs exchanged between TCs are used to coordinate the operation between test components.

The test architecture should deal with these concurrent behaviors at several interaction points. Test system is the implementation of a test suite based on C-TTCN. It carries out experiments by

Supported by the National Natural Science Foundation of China, Grants No. 69473011 and No.69682002.

executing test cases and observing results. C-TTCN is very suitable for the specification of system, which consists of several testers. MTC plays the role of main lower tester. MTC's behaviors are defined by the first behavior tree in each test case, and MTC is often used to create or terminate PTCs, manage CPs between MTC and PTC, receive the preliminary result from PTCs and give out the final test verdict. PTC plays the role of parallel lower tester or upper tester. The behaviors of PTC are defined by local behavior tree in a test case or a test step.

2.2 CEBE Based Entity Specification

To describe entities in a complex distributed system, we define a specification model CEBE. In [5], we presented a model EBE to specify peer-to-peer protocol entity. CEBE is the extension of EBE and aims to improve the specification abilities of concurrent behaviors and interaction points, according to the characteristics of entities in the distributed system.

Definition 2.1. *CEBE*

A CEBE is a 7-tuple $\langle S, G, \Sigma, A, R, s_0, v_0\rangle$, *where*:

1. S *is a finite set of external states, whose elements are pause states of interactions exchanged between the entity and its external environment*; 2. G *is a set of gates over which a CEBE can communicate*; 3. Σ *is a set of data on CEBE*; 4. A *is a set of external actions on CEBE*; 5. R *is a set of logic relations of transitions between external states*; 6. $s_0 \in S$ *is the initial state, which represents the initial external state*; 7. v_0 *is a set of initial assignments of* Σ *(initial values of all variables and timers)*.

Definition 2.2. *Data*

$\Sigma = IO \cup VAR \cup CONS \cup TIMER \cup PARA$ *is a set of data on CEBE, where*:

1. IO *is a set of input or output events (i.e., ASPs and PDUs)*; 2. VAR *is a set of variables (including integer, Boolean, octetstring, bitstring, etc.)*; 3. $CONS$ *is a set of constants (including integer, Boolean, octetstring, bitstring, etc.)*; 4. $TIMER$ *is a set of timers (second and microsecond)*; 5. $PARA$ *is a set of parameters of events in* IO. *For an event* $e \in IO$, *parameters* $p_1, p_2, \ldots, p_n$ *are denoted as* $e.p_1, e.p_2, \ldots, e.p_n$, e *is denoted as* $e(e.p_1, e.p_2, \ldots, e.p_n)$.

Definition 2.3. *Behavior*

A behavior b *is the element* g *of* G *with a list of input and output events and their values declaration. Then the set of* b *is denoted as*:

$B = \{\langle g_i, ?r_i\ (r_i.p_1, r_i.p_2, \ldots, r_i.p_n), !s_i\ (s_i.p_1, s_i.p_2, \ldots, s_i.p_n)\rangle | g_i \in G, r_i, s_i \in IO, r_i.p_i, s_i.p_i \in PARA\}$, *where symbol "!" indicates the output of an event* s_i *to external environment and symbol "?" indicates the input of an event* r_i. *The absence of* r_i *or* s_i *could be denoted as symbol "–". For example* $b = \langle g_i, ?r_i\ (r_i.p_1, r_i.p_2, \ldots, r_i.p_n), !- \rangle$. *Here we define some notations*:

(1) $b_1 \gg b_2 \gg \cdots \gg b_n$, *behaviors are sequential*; (2) $b_1[]b_2[]\ldots[]b_n$, *behaviors are equally valid alternatives (choice)*; (3) $b_1\|b_2\|\cdots\|b_n$, *behaviors are parallel*.

Definition 2.4. *Action*

Action is a set of (concurrent) behaviors. $A = \{a_i | a_i \in A\}$, $a_i =_{\text{def}} b_j \in B | a_i[]a_i | a_i \gg a_i | a_i \| a_i | (a_i)$, *where the operator significance is* '( )' > '[]' > '≫' > '‖'.

For instance, "$a_1 = a_2\|a_3 = b_1\|(b_2 \gg (b_3[]b_4))$" denotes there are two concurrent actions: "a_2" and "a_3". Moreover, a_2 is a behavior b_1, and a_2 is the sequential action "$b_2 \gg b_3$" or "$b_2 \gg b_4$".

Definition 2.5. *Transition relation*

$R \subseteq S \times A \times S \times P \times PowerSet(O) \times PowerSet(F)$ *is a set of transition relations, where*:

1. A *is a set of actions*; 2. S *is a set of states of* CEBE; 3. $P(\Sigma) : PowerSet(\Sigma) \to \{\text{TRUE}, \text{FALSE}\}$ *is a set of predicate functions* ($=, \geq, \leq, >, <, \wedge, \vee, \neg$, *Timeout, etc.*); 4. $O(\Sigma) : PowerSet(\Sigma) \to PowerSet(\Sigma)$ *is a set of operation functions* ($:=, +, -, *, /$, *StartTimer, CancelTimer, etc.*); 5. $F(\Sigma) : PowerSet(\Sigma) \to PowerSet(\Sigma)$ *is a set of functions on* CEBE (*i.e. abs*(), max(), min(), *etc.*).

Example 2.1.

Predicate :	$p_1 : (a > b) \wedge (c > 1)$,	then $p_1(\{a, b, c\}) = \text{TRUE}$;
Operation :	$o_1 : c := a + b$,	then $o_1(\{a, b, c\}) = \{c\}$;
	$o_2 :$ StartTimer T_1,	then $o_2(\{T1\}) = \{T1\}$;
Function :	$f_1 : c := \max(a, b)$,	then $f_1(\{a, b, c\}) = \{c\}$.

The intuitive meaning of $r \in R$ is that if CEBE is in state s and enabling action a is offered, then enabling predicate p is evaluated on the current assignment of variables. When p is true, CEBE will go into the new state s' and the environment is updated by operation o and function f.

The absence of predicate could be denoted as TRUE, the absence of operation or function could be denoted as NIL.

2.3 The Approach to Entity Conformance Testing

The essence of both EBE and CEBE is to describe data in addition to control behaviors from viewpoint outside the system. [6] gives a translation from LOTOS or Estelle to ETS. In [7], the ETS based semantics for Z is presented. CEBE is similar to EBE and could be obtained from ETS by eliminating internal actions. Therefore, CEBE could be obtained manually or translated from standard FDTs. The approach to entity conformance testing is shown in Fig.1. In [8], we proposed a specification framework for distributed system based on ETS. With a translation method, the CEBE of an entity can be obtained from its ETS. In CEBE, we only define external states and external actions. Therefore the model of CEBE would be simpler than other models and it is ideal for complex entities in distributed systems. Then we select CEBE as the specification model of entity. In Subsection 2.1 C-TTCN is used in test systems, so we propose a C-TTCN test suite generation method from the entity specification of CEBE. There are two phases in our test generation method:

Phase 1: Derive test sequences from CEBE; Phase 2: Generate the test suite in C-TTCN.

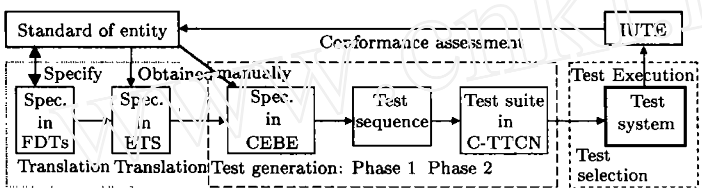

Fig.1. An approach to entity conformance testing in distributed systems.

3 C-TTCN Test Generation

3.1 Test Sequence Derivation

Algorithm 3.1. SIP Generation

Step 1. Identify all possible valid IPs (Interaction Paths) from CEBE specification by imposing constraints set (see Algorithm 3.2) on IPs. An IP is an externally observable track on which a sequence of interactions (including concurrent behaviors) between entity and its external environment occurs, starting from initial state s_0 and ending in s_0. Any loop interaction in an IP is traveled only once.

Step 2. Generate SIPs (I/O Subpaths) from the above IPs through checking the dependencies between relations in each IP. An SIP is the externally observable track $r_1, \ldots, r_k$ in an IP, where:

1. $r_1 = \langle s_{r1}, a_{r1}, s'_{r1}, p_{r1}, O_{r1}, F_{r1} \rangle$ is the first transition relation when it satisfies:

a) a predicate $p_{r1}(\Sigma_{pr1})$, and/or; b) a set of operations $\{\ldots, o_{rli}(\Sigma_{o_{rli}}), \ldots\}$, and /or; c) a set of functions $\{\ldots, f_{rli}(\Sigma_{f_{rli}}), \ldots\}$;

2. $r_k = \langle s_{rk}, a_{rk}, s'_{rk}, p_{rk}, O_{rk}, F_{rk} \rangle$ is obtained when logical relations satisfy:

a) $(\forall i, (\Sigma_{p_{rk}} \cap o_{r_{li}}(\Sigma_{o_{rli}})) \neq \emptyset) \vee (\forall i, (\Sigma_{p_{rk}} \cap f_{rli}(\Sigma_{f_{rli}})) \neq \emptyset) \vee (\forall i, \forall j, (\Sigma_{o_{rkj}} \cap o_{rli}(\Sigma_{o_{rli}})) \neq \emptyset)$ $\vee(\forall i, \forall j, (\Sigma_{orkj} \cap f_{rli}(\Sigma_{f_{rli}})) \neq \emptyset) \vee(\forall i, \forall j, (\Sigma_{f_{rkj}} \cap o_{rli}(\Sigma_{o_{rli}})) \neq \emptyset) \vee(\forall i, \forall j, (\Sigma_{f_{rkj}} \cap f_{rli}(\Sigma_{f_{rli}})) \neq \emptyset)$, and; b) a predicate p_k $(\Sigma_{p_{rk}})$, and/or; c) a set of operations $\{ \ldots, o_{kj}(\Sigma_{o_{rkj}}), \ldots\}$ and/or; d) a set of functions $\{ \ldots, f_{kj}(\Sigma_{f_{rkj}}), \ldots\}$.

Example 3.1. In r_1, o_1: $c := a + b$; o_2: StartTimer T_1; In rk, $p_1 : (a > 0) \wedge (b > 1)$; p_2: Timeout T_1; $o_3 : d := c - 1$;

Then, $o_1(\{a, b, c\}) = \{c\}$, $o_2(\{T_1\}) = \{T_1\}$;

$p_1(\Sigma_1) = (\{a, b\}) =$TRUE, $p_2(\Sigma_2) = (\{T_1\}) =$TRUE, $o_3(\Sigma_3) = (\{c, d\}) = \{d\}$;

(1) $\Sigma_1 \cap o_1 = \emptyset$; (2) $\Sigma_1 \cap o_2 = \emptyset$; (3) $\Sigma_2 \cap o_1 = \emptyset$; (4) $\Sigma_2 \cap o_2 = \{T_1\} \neq \emptyset$; (5) $\Sigma_3 \cap o_1 = \{c\} \neq \emptyset$; (6) $\Sigma_3 \cap o_2 = \emptyset$;

In this example, rk is associated with r_1 in (4) and (5). (4) means that r_1 starts a timer T_1 and the predicate of rk is the expiration of T_1. (5) means that r_1 changes the value of c and c is used in rk. Either of the two situations is satisfied ("$\vee$"), then $r1$ and rk should be in one SIP.

The number of different test sequences of an entity in complex distributed system and the length of some test sequences can be potentially infinite, because there are many recursive behaviors and a practically infinite number of parameter value combinations. It thus becomes necessary that the depth and breadth of the test tree have to be restricted. Instead of insetting a restriction into the test suite generation algorithm, we introduce a flexible and user-definable restriction mechanism, the so-called constraints set. In this context a constraint is a Boolean predicate that has to hold for each IP.

Algorithm 3.2. Constraints Set

Rule 1. Let the visited function denote the number of times transition R_{ij} is visited in an IP. For each transition R_{ij}, min_visited(R_{ij}) defines minimal times the transition R_{ij} has to be visited in an IP and max_visited(R_{ij}) defines maximal times. Min_visited(R_{ij}) and max_visited(R_{ij}) are called constraints on transitions. An IP fulfills constraints on states if and only if min_visited(R_{ij}) $\leq$ visited(IP, R_{ij}) $\leq$ max_visited(R_{ij}). Thus constraints on transitions specify how many times each transition must be visited and allowed to be visited in an IP, in order to control test coverage on the test suite.

Rule 2. Let the assigned and used functions denote the number of times a variable is assigned a value in an IP. The constraint on variables can be similarly expressed in terms of min_assigned(VAR_i), min_used(VAR_i), max_assigned(VAR_i), and max_used(VAR_i). With these functions, we can select test cases according to variables in which we are interested.

Rule 3. A set of constraints on timers specifies a range for the number of times the actions StartTimer(T_i), CancelTimer(T_i) and Timeout(T_i) can/must be performed for each timer $T_i \in TIMER$. With timer constraint set, we can only select these SIPs with right time relation.

3.2 Test Suite Generation

Based on the SIP, a test tree with the correctness, nondeterministic and defense branches can be derived, and one kind of verdicts (PASS, FAIL or INCONC) is assigned to each leaf of the test tree. The test tree can be mapped to a test case in C-TTCN. Finally, these test cases are grouped according to some rules, and are mapped to a complete test suite in C-TTCN. In a test case specified by C-TTCN, the behaviors of each test component are expressed with a behavior tree (BT) in test case. BT is a tree-like presentation of temporal relations between test events. In order to get formal definition of the semantics, we need to introduce the Test Behavior Expression (TBE) defined below to express the behaviors of test components in algebraic form.

Definition 3.1 (The Syntax of a TBE). *TBE* $B =_{\text{def}}$ *stop* | *exit* | *id?a; B* | *id!y; B* | $B[]B$ | $B \gg B|[q];B\ |(I := val);B|(B)|$ *start tid val; B|cancel tid; B| timeout tid; B| create id B| done id.*

The operator significance is ';' > '[]' > '$\gg$'. 'stop' denotes termination of a running. 'exit' denotes a successful quitting from an attached TBE. '*id?a; B*' and '*id!y; B*' represent respectively a receiving event and a sending event at PCO or CP identified by 'id'. ';' denotes sequential composition. '[]' denotes choice. '*q*' is the qualifier which is a Boolean expression, its value is either TRUE or FALSE. '$I := val$' is an assignment, where '*I*' is an identifier and 'val' is a value. '$\gg$' denotes the attaching operation. 'start' and 'cancel' are a pair of timer operators. 'timeout' represents the expiration of a timer. 'create id *B*' creates a PTC named id whose behavior tree is *B*, while 'done id' waits until the PTC named id has finished its running.

To understand the algorithm, here we use TBE to describe the rules for C-TTCN generation. In the real tool TUGEN, the test suite in C-TTCN MP is generated.

Algorithm 3.3. C-TTCN Test Case Generation

Rule 1. Each SIP is mapped to one C-TTCN test case;

Rule 2. In one SIP, if there are several relations $r_1, r_2, \ldots, r_n$, then attach tree $i+1$ to tree i;

TBE$_M$: $r_1 \gg r_2 \gg \cdots \gg r_n$

Rule 3. In an action a of relation r, if $b_1 \gg b_2 \gg \cdots \gg b_n$, then $b_1, b_2, \ldots, b_n$ are behaviors and it is successfully completed if b_{i+1} happens immediately after the successful completion of b_i ($i = 1, 2, \ldots, n-1$):

TBE$_M$: create P_1 TBE$_1$; start t_1; (CP1?CM_1; down P_1; stop) [] (timeout t_1;stop)

TBE$_1$: $b_1 \gg b_2 \gg \cdots \gg b_n \gg$ (CPi!CM_i; stop)

Rule 4. In an action a of relation r, if $b_1[]b_2[]\cdots[]b_n$, then an alternative is matched if behavior b_i corresponding to the alternative happens first:

TBE$_M$: create P_1 TBE$_1$; start t_1; (CP1?CM_1; down P_1; stop) [] (timeout t_1; stop)

TBE$_1$: ($b_1 \gg$ (CPi!CM_i; stop)) [] ($b_2 \gg$ (CPi!CM_i; stop)) [] $\cdots$ [] ($b_n \gg$ (CPi!CM_i; stop)

Rule 5. In an action a of relation r, if $b_1||b_2||\cdots||b_n$, then $b_1, b_2, \ldots, b_n$ are started in parallel and the parallel tree completes only if $b_1, b_2, \ldots, b_n$ complete:

TBE$_M$: create P_1 TBE$_1$; $\cdots$; create P_n TBE$_n$; start t_1; (CP$_1$?CM_1; down P_1; start t_2; ($\cdots$; (CPn?CM_n; doneP_n; stop) [] (timeout t_n; stop)); $\cdots$) [] (timeout t_2; stop) [] $\cdots$ [] (CPn?CM_n; down P_n; start t_2; ($\cdots\cdots$) [] (timeout t_2; stop)) [] (timeout t_1; stop)

TBE$_i$: $b_i \gg$ (CPi!CM_i;stop)

Rule 6. In each behavior b of action a in a relation r, $\langle g_i, ?r_i(r_i.p_1, r_i.p_2, \ldots, r_i.p_n), !s_i(s_i.p_1, s_i.p_2, \ldots, s_i.p_n)\rangle$, then

TBE$_i$: $g_i!r_i; g_i?s_i$; CP1!CM_1; stop

Rule 7. In declaration part, timer, variable and constant could be mapped from TIMER, VAR, and CONS with the initial value from v_0;

Rule 8. In the declaration part, ASP, PDU could be mapped from IO with parameters PARA for the events;

Rule 9. In constraint part, the timer, variable, ASP and PDU could be mapped from each event e_i $(e_i.p_1, e_i.p_2, \ldots, e_i.p_n)$ in b with values of parameters.

The above methods have been implemented in an automatic tool TUGEN. The CEBE parser module takes a CEBE-NF specification as input, and produces the CEBE-BT as output. It includes a lexical scanner and a syntax parser. The lexical scanner breaks up the input into tokens. The syntax parser produces CEBE-BT by using these tokens and the generation grammar of CEBE-BF. The model "translator" transforms SDL, Estelle, LOTOS specification or ETS to CEBE or EBE. TUGEN constraints are specified by using a default scheme and a constraint editor. They are initialized to a default value when CEBE parser produces internal data structure representation. IP and SIP generation modules and test suite derivation module are used to generate a test suite in C-TTCN based on the output of CEBE parser. For detailed information about the design of each module, please see [9].

4 Conclusions

As already mentioned, this paper presents C-TTCN based entity conformance testing in distributed systems. Because CEBE only describes external behaviors and data of a system, it is suitable for entities in complex distributed systems. The related studies have been done about the translation from standard FDTs such as LOTOS, Estelle, SDL, and Z to ETS. Because CEBE could be obtained from ETS, it may be translated from these standard FDTs. When an entity is formally specified by CEBE, its C-TTCN test suite can be generated from this CEBE, integrating the features of data-flow testing and control-flow testing, supplying constraint set scheme by analyzing the fault coverage. We will study CEBE model and C-TTCN generation algorithm to improve the efficiency of the C-TTCN test suite generated. For example, we want to combine TBEs of different PTCs in one test case to deduce the size of behavior tree in this test case.

References

[1] Farooqui K, Logrippo L, de Meer Jan. The ISO model for open distributed processing. *Computer Networks & ISDN Systems*, 1995, 27: special issue.

[2] Loureiro A A F *et al.* Checking unstable properties in distributed testing of communication protocols. In *9th International Workshop on Testing of Communicating Systems*, IFIP, 1996.

[3] Kim M, *et al.* An approach for testing asynchronous communicating systems. In *9th International Workshop on Testing of Communicating Systems*, IFIP, 1996.

[4] Wong A C Y *et al.* A framework for distributed object-oriented testing. In *10th International Conference on Formal Description Techniques for Distributed Systems*, IFIP, 1997.

[5] Wu J, Chanson S T. Testing sequence derivation based on external behavior expression. In *2nd International Workshop on Protocol Test Systems*, IFIP, 1989.

[6] Wu J, Chanson S T. Translation from LOTOS and Estelle specifications to extended transition system and its verification. In *2nd International Conference on Formal Description Techniques for Distributed Systems*, IFIP, 1989.

[7] Derrick J *et al.* Supporting ODP — Translating LOTOS to Z. In *1st International Workshop on Formal Methods for Open Object-Based Distributed Systems*, IFIP, 1996.

[8] Bi J, Wu J, Chen X. Towards the formal model and testing of distributed systems. In *10th International Conference on Parallel and Distributed Systems*, IASTED, 1998.

[9] Wang J, Hao R, Wu J. TUGEN: An automatic test suite generator integrating data-flow and control-flow methods. In *International Conference on Communications, IEEE*, 1998.

BI Jun received his B.S., M.S. and Ph.D. degrees from Tsinghua University. He is a lecturer at Dept. of Computer Science and Technology, Tsinghua University. His research interests include high-speed networks and Internet, formal methods, network protocols, and network routing.

WU Jianping received his B.S., M.S. and Ph.D. degrees from Tsinghua University. He is a Professor at Dept. of Computer Science and Technology, Tsinghua University. He is also the director of the CERNET Technical Board. His research interests include high-speed networks and Internet, protocol engineering, network management, and network security.