공학 연구논문집 제26권 제2호 pp. 79~86, 울산대학교
Journal of Engineering Research Vol. 26, No. 2, pp. 79~86 1995.
University of Ulsan.

사건중심 실시간 시스템을 위한 테스트 시퀀스의 발생[1)]

이명준 · Sang H. Son
전자계산학과 · Department of Computer Science
University of Uisan · University of Virginia

<요 약>

사건중심 실시간 시스템의 명세기법인 Modechart를 이용하여 성형적으로 명세된 실시간 시스템의 구현을 검증하기위한 방법을 제시한다. 제안된 방법을 통하여, 주어진 실시간 시스템의 Modechart 명세와 그 구현간에 상이점이 있는지를 검증하기 위한 테스트 시퀀스를 발생하고, 가상환경하에서 이 발생된 테스트 시퀀스를 이용하여 그 구현이 명세와 일치되는 지를 검증할 수 있다.

Generating Test Sequences for Event-Driven Real-Time Systems

Myung-Joon Lee · Sang H. Son
Department of Computer Science
University of Uisan · University of Virginia

<Abstract>

We present a method for testing event-driven real-time systems based on Modechart specifications. From a Modechart specification, the proposed method generates test sequences for checking whether there is a discrepancy between a Modechart specification and its implementation. The implementation can be tested, under virtual environment, by the generated sequences for conformance to its specification.

---

1. The preliminary version of the paper appeared in the Proceedings of 2nd IEEE workshop on Real-Time Applications.

1. Introduction

A real-time system supports time-critical applications such as aircraft avionics, robotics, process control and traffic control. A failure in meeting timing constraints associated with those applications may result in catastrophic consequences. Since design errors of those systems are one of the main sources of the failure, much work [5, 10, 4] has been done over the past several years to find formal methods for specifying real-time systems more rigorously and natually, and for verifying the properties automatically that must be satisfied by the specifications of those systems. Among those formal methods, Modechart [8, 3] is well-known for specifying event-driven real-time systems, where an appropriate response to certain events must satisfy timing constraints. The semantics of Modechart can be defined using RTL (Real-Time Logic) [7]. There are useful tools [4, 14] developed for supporting the Modechart specification of real-time systems and the verification of the desirable system properties.

Real-time applications, i.e., implementations of those real-time systems, should be tested before being actually used. If the testing of a real-time application is performed under the associated real environment, it would be expensive and time-consuming. Moreover, when a failure occurs in the process of testing, it is hard to determine whether the failure is due to some inappropriate setting of the perating environment, or due to some incorrectness in implementing the specification for the real-time application. Thus, it is desirable to test real-time applications for conformance to its specification under virtual environment.

To address this problem, in this paper, we propose a technique for testing whether there is a discrepancy between a Modechart specification and its implementation. As is the case with most techniques for protocol testing [13, 1], our technique is based on transition testing — forcing the Modechart implementation under test to experience every transition in the specification. Thus, the principles of our technique might be easily applied to other specification methods based on finite state machines such as CRSM [10].

2. System Model

We use the restricted form of Modechart recently presented by Yang, Mok and Wang [14]. It can be described briefly as follows. A modechart is a parallel mode composed of a finite set of serial modes Each serial mode consists of a finite set of atomic modes and a set of labeled transition edges between atomic modes. There is a distinguished atomic mode called flinitial mode.

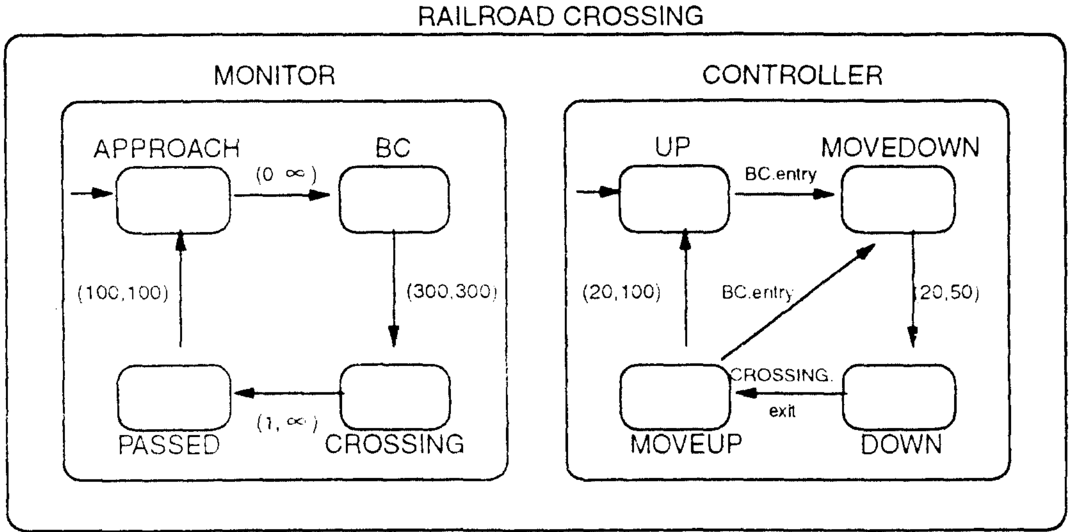

Figure 1. A Modechart Specification for Railroad Crossing System

A label of each edge is either triggering condition (a disjunctive normal form of events or a timing condition (r and d are discrete times, and denote delay and deadline, respectively.) Here, an event is either an entry event a.entry for some atomic mode a or an exit event a.exit for some atomic mode a or a transition event a → a′ for some edge <a, a′>.

The following railroad crossing problem and the Modechart specification in Figure 1 is a famous example mentioned very often [8, 12, 14].

"The gate at a guarded railroad crossing is to be software controlled, and since the gate cannot control the train, a real-time solution is needed. There is an early warning signal at a distance from the crossing that gives notice to the gate controller that a train is approaching (mode transition APPROACH → BC), and it is known that it takes the train at least 300 time units to reach the crossing from the signal (BC → CROSSING). It is also known that the time required to lower the gate is between 20 and 50 time units (MOVEDOWN → DOWN). The controller itself can detect the departure of the train (CROSSING → PASSED), and it requires between 20 and 100 time units to raise the gate (MOVEUP → UP). It is also known that trains are scheduled so that it takes at least 100 time units from the time a train leaves the crossing until the next train reaches the early warning signal (PASSED → APPROACH)."

3. Testing Method

When testing protocol implementations specified by finite state machines, UIO(Unique Input Output) sequences are usually used as state signatures, each of which is a sequence of input/output pairs that can identify the state of a machine, not requiring the name of the state. Such a black box approach might not be used directly to test event-driven real-time systems specified in Modecharts, because each atomic mode* * a is essentially associated with system events (a.entry, a.exit, transition from a or to a), each of which may cause a transition in other serial modes, as well as may start an associated real-time behavior. In other words, events themselves constitute the input/output of the implementation of each serial mode. Hence, the behavior of the implementation of each serial mode can be checked by stimulating or observing those input/output events. Regardless of whether serial modes communicate through sharing variables or passing messages, we assume that those events can be stimulated or observed by an external tester. Otherwise, the implementation cannot be tested by the external tester.

In addition, the events not explicitly represented in Modechart specification should be taken into consideration. For example, the transition from MOVEDOWN to DOWN in the railroad crossing example is performed actually after the external behavior associated with MOVEDOWN is completed.

Thus, for each atomic mode a associated with certain external behavior, we consider a. done as an implicit event, indicating the completion of the behavior and causing the explicit event a.exit. In this paper, we associate an implicit event with the source atomic mode of each transition which has a timing condition, and regard it as an input for the implementation under test, according to the model as illustrated in Figure 2. In this model, input events and implicit events are stimulated by the external tester, whereas output events are observed by the tester.

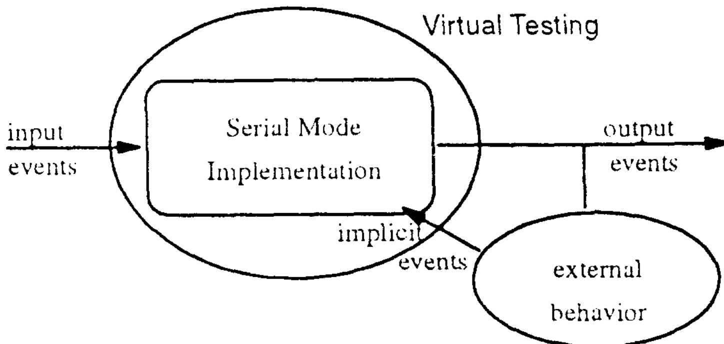

Figure 2. System Model for Testing

Since a serial mode specification consists of labeled transitions, it is necessary to test whether or not each labeled transition is implemented exactly according to the specification. A test case for a labeled transition is a sequence of input/output events such that if any possible transition is described by the sequence, that transition is identical to the transition under test. When testing a transition t = <a, a'> with triggering condition e, the two output events a.exit and a'. entry should be observed in that order as a response to the input event $e = e_1 \vee e_2 \vee \cdots e_n$. Thus, the test case for t is defined:

Testcase(t) = e_1 / a.exit / a'.entry / reset
Path(a_0, a) / e_2 / a.exit / a'.entry / reset /
...
Path(a_0 , a) / e_n / a.exit / a'.entry

For testing a transition with a timing condition, we need more assumptions: (1) Every implementation of a serial mode under test accepts a special input event reset which initialize the serial mode to its initial mode.(2) Every implementation of a serial mode under test generates a special output event time_error whenever the timing condition is not satisfied. (3) As an interval between delay and deadline, a sequence of adjecent times is enforced on every implementation of any timing condition even when the erroneously implemented timing condition does not match the specified one. (Most current real-time programming languages [2, 6] have programming constructs supporting this assumption, enabling us to avoid testing each member in the timing condition unreasonably).(4) The amount of time for passing or receiving events is ignorable. (This can be achieved by preparing the virtual testing environment properly.) In addition, the tester can wait for a specified amount of time units; wait(w) denotes the action of the tester waiting for w time units. Since it is impossible to wait(inf), we will use wait(INF) where INF is the appropriate value proposed by the system designer.

For the convenience of description, let Path(a0, a) denote a sequence of input/output events which brings the serial mode into the atomic mode a, starting from the initial mode a sub 0. Path for each transition is defined as follows: for a transition <a, a'> with a timing condition (r), Path(a, a') is (wait(r)/ a. done / a.exit / a' .entry), whereas for a transition <a, a'> with a triggering condition, Path(a, a') is (e/ a.exit / a'. entry) which is Testcase(<a, a'>).

Under these assumptions, for testing a transition t = <a, a'> with the timing condition (r), it suffices to check the time boundaries of the timing condition, i.e., times r-1 and d+1 for failure and times r and d for success: Testcase(t) = (wait(r-1) / a.done / time_error / reset / Path(a_0, a) / wait(d+1) / a.done / time_error / reset / Path(a_0, a) / wait(r) / a.done / a.exit / a'. entry / reset / Path(a_0, a) / wait(d) / a.done / a. exit / a' .entry).

In the case of r = 0, the subsequence "wait(r-1) / a.done / time_error / reset / Path(a_0, a) " should be removed from the above definition; also, in the case of d = inf, the subsequence "Path(a_0, a) / wait(d+1) / a.done / time_ error / reset" should be

removed. Obviously, in the case of r = d, the subsequence "reset / Path(a_0 , a) / wait(d) / a.done / a.exit / a' .entry" is unnecessary.

To combine test cases to form a test suite for testing an implementation of a serial mode entirely, let a tour of a serial mode be a finite nonnull sequence of consecutive transition edges that starts and ends at the initial mode. For simplifying the discussion, consider a postman tour of which is a tour containing every transition edge of at least once, assuming that there is a path between any two atomic modes. Then a test suite for is a sequence generated by combining test cases for transitions in a postman tour of orderly. In the railroad crossing system, a postman tour of CONTROLLER mode is

(<UP, MOVEDOWN>, <MOVEDOWN, DOWN>, <DOWN, MOVEUP>, <MOVEUP, MOVEDOWN>, <MOVEDOWN, DOWN>, <DOWN, MOVEUP>, <MOVEUP, UP>).

For efficient testing, clearly, the test cases for transitions occurring redundantly in the tour can be replaced in the test suit by Path for those transitions. According to the above discussion, a test suite for CONTROLLER in the rail-road crossing system is presented in Appendix.

The implementation scheme for the proposed testing method may vary on the communication methods between serial modes in the real-time application under test.When serial modes communicate through some type of communication channels by exchanging messages, the testing method might be implemented by stimulating and/or observing those channels.

4. Concluding Remarks

Due to the lack of rigorous specification and testing methods, real-time systems have been developed from an informal specification and verified and tested with ad hoc techniques or with expensive and extensive simulations [11]. While the formal specification and verification of real-time systems has received much attention recently, the problem of testing implementations of specifications written in those methods have not been considered yet.

We have challenged the problem of generating test sequences for implementations of specifications written in Modechart, one of the successful formal methods with supporting tools. As a result, the problem of testing whether or not the transitions in a Modechart specification are implemented correctly, has been addressed. Future work will include the development of a method for testing the transitions which are not specified but happen to be implemented. Also, based on the same principle, a virtual testing method for real-time applications specified by CRSM will be considered. We believe that the testing problem is essential in developing real-time applications and more research needs to be performed in this important area.

References

[1] A.V. Aho, A.T. Dahbura, D. Lee and M. Umit Uyar, "An optimization technique for protocol conformance test generation based on UIO sequences and Rural Chinese Postman Tours," Trans. on Communications Vol. 39, No. 11, Nov. 1991.

[2] T. Baker and O. Pazy, "Real-time features for Ada9X," roc. of 12th Real-Time System Symposium, 1991

[3] P.C. Clements, C.L. Heitmeyer and B.G. Labaw, "Applying formal methods to an embedded real-time avionics system, "roc. of IEEE Real-Time Applications Workshop, New York, NY, May 11-12, 1993.

[4] P.C. Clements, C.L. Heitmeyer, B.G. Labaw and A.T. Rose, "MT: A toolset for specifying and analyzing real-time systems, "roc. of 14th Real-Time System Symposium, 1993.

[5] IEEE Trans. on Software Engineering, Special issue on specification and analysis of real-time systems, SE-18, Sep. 1992.

[6] Y. Ishikawa, H. Tokuda and C.W. Mercer, "An object-oriented real-time programming language," IEEE Computer Vol. 25, No. 10, Oct. 1992.

[7] F. Jahanian and A.K. Mok, "Safety analysis of timing properties in real-time systems," IEEE Trans. on Software Engineering SE-12, Sep. 1986.

[8] F. Jahanian and D.A. Stuart, "A method for verifying properties of Modechart Specifications," Proc. of 9th Real-Time System Symposium, 1988.

[9] D. Scholefield, "The Formal Development of Real-Time Systems: A Review,"Department of Computer Science Technical Report YSC-145, University of York, Feb. 1992.

[10] A. Shaw, "Communicating real-time systems," IEEE Trans. on Software Engineering SE-18, No. 9, Sep. 1992.

[11] J.A. Stankovic, "Misconceptions about real-time computing," IEEE Computer, Oct. 1988.

[12] D. Stuart, "Implementing verifier for real-time systems," Proc. of 11th Real-Time System Symposium, 1990.

[13] K. Sabnani and A. Dahbura, "A protocol test generation procedure," Computer Networks and ISDN Systems Vol. 15, No. 4, Sep. 1988.

[14] J. Yang, A.K. Mok and F. Wang, "Symbolic model checking for event-driven real-time systems," Proc. of 14th Real-Time System Symposium, 1993.

Appendix

A test suite for CONTROLLER mode implementation in the railroad crossing:.
(BC.entry / UP.exit / MOVEDOWN.entry / wait(19) / MOVEDOWN.done / time_error / reset / Path(UP, MOVEDOWN) / wait(51) / MOVEDOWN.done / time_error / reset / Path(UP, MOVEDOWN) / wait(20) / MOVEDOWN.done / MOVEDOWN.exit / DOWN.entry/ reset /Path(UP, MOVEDOWN) / wait(50) / MOVEDOWN.done / MOVEDOWN.exit / DOWN.entry /CROSSING.exit / DOWN.exit / MOVEUP.entry / BC.entry / MOVEUP.exit / MOVEDOWN.entry / Path(MOVEDOWN, DOWN) / CROSSING.exit / DOWN.exit / MOVEUP.entry / wait(19) / MOVEUP.done / time_error / reset / Path(UP, MOVEUP) /wait(101) / MOVEUP.done / time_error / reset / Path(UP, MOVEUP) /CROSSING.exit / DOWN.exit / MOVEUP.entry /wait(20) / MOVEUP.done / MOVEUP.exit / UP.entry / reset /Path(UP, MOVEUP) /CROSSING.exit / DOWN.exit / MOVEUP.entry /wait(100) / MOVEUP.done / MOVEUP.exit / UP.entry)

where

Path(UP, MOVEDOWN) = BC.entry / UP.exit / MOVEDOWN.entry,\ Path(MOVEDOWN, DOWN) = wait(20) / MOVEDOWN.done / MOVEDOWN.exit / DOWN.entry, Path(UP, MOVEUP) = BC.entry / UP.exit / MOVEDOWN.entry / wait(20) / MOVEDOWN.done / MOVEDOWN.exit / DOWN.entry CROSSING.exit / DOWN.exit / MOVEUP.entry.