MOTOROLA LABS
European Intellectual Property Operations

PATENT: GB2365156
DATE PUBLISHED:2002-02-13
Bill Mitchell, Paul Baker, Clive Jervis

PART I - TECHNICAL INFORMATION

1. Title of the Invention: **Method for Generating Coordinating Messages for Distributed Test Scripts**

   Add Useful Key Words:
   Automated Testing, Message Sequence Charts, Concurrent Testing, Distributed Testing

2. What was the problem(s) to be solved by the invention or what was the need(s) for the invention:

In recent years the use of automated testing techniques has reduced cycle time greatly with the introduction of formal specifications languages (e.g. MSC, ASN.1, SDL, TTCN etc), simulation and model checking tools. The next step has been to explore further opportunities for reducing cycle time. In doing so, we have to increase the use of and combine the various formal notations used within our development processes. As a consequence Motorola's UK research lab (MUKRL) has produced a method (and supporting tools e.g. PTK) for generating test scripts from requirements specifications described using Message Sequence Charts (MSC). In automatically generating tests we are checking that the implementation of the system is correct with respect to its specification. In other words, the implementation exhibits all the behaviours defined by its specification.

Current approaches for generating tests produce scripts that execute a single sequence of actions that verify the implementation under test (IUT) has performed correctly. However, in recent years testing languages, such as TTCN, have been enhanced to facilitate the notion of concurrent testing. Where, a concurrent test script consists of a number of concurrent test components that interact to perform a particular test. This invention describes how to produce the co-ordination messages that are necessary to synchronise concurrent test components in order to check that an IUT exhibits the correct order of events. Note that we assume co-ordinating messages have a suitably small latency (delay), and the specification does not contain any race conditions [1].

In order for these concurrent test components (CTC) to test the IUT they need to communicate at various stages of a test to ensure that:
(1) Messages they received from the IUT are correct responses to messages sent to the IUT from other parts of the system,
(2) CTCs send particular messages in the correct order with respect to one another, as laid down by the defining specification.

Such co-ordination is achieved by the transmission of co-ordinating messages between the different CTCs. However, the challenging problem is deciding what co-ordinating messages are required to carry out a particular concurrent test. If no assumptions are made about the latency (delay) associated with a co-ordinating message it is not possible to use them during the computation of a test [2]. Hence, the invention describes methods for introducing co-ordinating messages that perform tasks (1), and (2) above. The result of these methods is to produce a list of similar sub-specifications, each one corresponding to a particular concurrent test from which it is possible to automatically generate the concurrent test scripts. In constructing each sub-specification the co-ordinating messages force the CTCs to exchange their messages with each other and the IUT in a fixed order. At each stage there is only one choice of event which can occur next in each PTC if the test is proceeding correctly. The list of actions for each PTC in the order specified by the MSC gives a concurrent test of the original MSC.

References

[1] Bill Mitchell, "*Concurrent test script generation for Message Sequence Charts*", UK Motorola Research Laboratory.
[2] Jens Grabowski, Beat Koch, Michael Schmitt, Dieter Hogrefe, "*SDL and MSC Based Test Generation for Distributed Test Architectures*" ,Elsevier Preprint, December 1998

3. What are the closest known technologies? Why don't they resolve the problem(s) or fulfil the need(s):

Tool vendors such as Telelogic, Objecttime and Rational Rose all produce test scripts from specifications such as MSC. However, none of these support concurrent test script generation.

4. How does this invention resolve the problem(s) or fulfil the need(s) in a new way? Include labelled drawings, flow charts or block diagrams to help explain the invention:

Preferred Embodiment

The invention involves a number of methods that manufacture a set of MSCs. The purpose of these methods is to construct the set of co-ordinating messages required to perform concurrent testing of an MSC. Let M represent the MSC from which we wish to construct a set of concurrent test scripts. The methods described are the main method CM, and a pre-processing method PRE-CM, together with an auxiliary method after_join required in the description of CM.

To explain the methods the following abbreviations are used.

We call an event that is not a co-ordinating event a proper event. For a given MSC M, and two events a, b in M let M(a < b) denote that M specifies that a occurs earlier than b. We use M(a ~ b) to abbreviate that M specifies b to occur after a, and there is no other event c in M which M specifies as occurring between a and b. We use M(a \ b) to abbreviate that M(a < b) and also there are no proper events which M specifies as occurring between a and b . For an event a in MSC M use M(a, next) to represent the list of events b for which M(a \ b). We write a => b to denote a message, where a is the send event and b is the receive event.

The first method PRE-CM pre-processes M as follows.

For every pair of events a, b occurring in the CTCs where M(a < b), and there are events c and d in the IUT where M(a ~ c), M(c < d), and M (d ~ b), add a co-ordinating message x => y where we pick the locations of x and y so that M(a ~ x) and M(y ~ b). Such co-ordinating messages are only guaranteed to be implementable when M is race free.

Having added these co-ordinating messages we can now remove the IUT, its internal structure is no longer relevant to the task of generating co-ordinating messages for the second task, co-ordinating the CTCs.

To explain the method after_join we use the following test when deciding if we can introduce a new co-ordinating message between two events.

Define the staircase_test between a and b in MSC M to succeed exactly when b is a send event and for every proper event c such that M(c \ b) then M(c < a). We write staircase_test(M, a, b) to represent true if this test holds, and to represent false when this test fails. See figure 1

Before describing the complete method we first introduce an auxiliary method after_join. When given an event x, a list of events e_1, e_2, ... , e_k (where k is any value), and an MSC M, this method adds new co-ordinating messages which produce a new MSC in which the events e_1, e_2, ... , e_k all occur later than x. If it is not appropriate to introduce new co-ordinating messages in this way the method produces a value ‘skip’. See figure 3 for an overview of the method.

The method after_join (x, e_1, ... , e_k, M) proceeds as follows. First we make a copy of the MSC which we call M’. The method adds new co-ordinating messages to M’ described as follows.

For each event e_i in the list e_1, ... , e_k do the following:

if x and e_i are both receive events do nothing , continue to the next case which is e_(i+1)

if M'(x < e_i) continue to the next case which is e_(i+1) (we regard this as being true if x = e_i)

if M'(e_i < x) stop the method with value 'skip'

if x or e_i is a send event then

if staircase_test (M', x, e_i) succeeds
then
add a coordinating message c_1 => c_2 to M'
so that in the new MSC, M'(x ~ c_1), and M'(c_2 ~ e_i)

if staircase_test (M', x, e_i) fails
then
stop the method with value 'skip'

After finishing the above procedure the method after_join (x, e_1, ... , e_k, M) has added various co-ordinating messages to manufacture a new MSC M'. In M' we have M'(x < e_i) for each i between 1 and k. Notice that if the method is applied to x when the list contains no e_i events then the method does nothing, so can be thought of as manufacturing just a copy of M.

We can now explain the main method CM as follows. The method can be applied to a list of events and an MSC. Let a_1, ... , a_k, be a list of events and M be a given MSC. Begin method CM with the MSC M under consideration and an empty set S of MSCs. The method incrementally adds new MSCs to this set as it traverses through the events in M. See figure 4 for an overview of the method.

1) For each event a_i in the list a_1, ... , a_n do the following

Calculate the value of the method after_join on the list a_1, ... , a_n with a_i removed from the list, and the MSC M.

Is this value 'skip'?

If so do nothing, except proceed to the next value a_(i+1) and continue the method from step 1.

If the value is an MSC M' then apply method CM to MSC M' together with the list of events consisting of M'(a_i, next) and the list a_1, ... , a_n with a_i removed. This constructs a set of MSCs, add all of these to the set S being constructed. Now return to step 1 and continue with the next case e_(i+1)

2) When all the events in the list a_1, ... , a_n have been processed according to step 1the method finishes, the set S now contains various MSCs with coordinating messages incorporated in them.

This method allows us to construct a set of MSC that correspond to the set of concurrent tests we wish to perform on M.
First apply the pre-processing method PRE-CM to M to construct a new MSC M1. Now remove all events of the IUT from M1. These are no longer required to co-ordinate the CTCs and would interfere with CM.

Next apply method CM to the list a_1, ... , a_k of initial events from the PTCs (an event is initial if there is no event prior to it in the MSC) and the MSC M1. The result is a list of MSCs, each of which corresponds to a concurrent test we wish to perform on M. Figure 2 illustrates how to calculate the list of intitial events in an MSC.

Finally we may choose to generate test scripts from each of the resultant message sequence charts. In doing so we must decide on verdicts which are to be associated with each test. Verdicts can be PASS, FAIL or INCONCLUSIVE. How the verdicts are assigned depends on the latency assumptions which are applied to the system.