European Intellectual Property Operations

PATENT: GB2397905
DATE PUBLISHED: 2004-08-04
Bill Mitchell, Paul Baker, Dave King

PART I - TECHNICAL INFORMATION

1. Title of the Invention: **Method for Categorising Test Scripts from Weights Derived from a Partial Order**

   Add Useful Key Words:
   Test Scripts, Message Sequence Charts, Automatic Testing, Test Categorisation

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

Requirements for systems involving communicating processes (e.g. communication systems) are often described with the aid of graphical notations, such as: Unified Modelling Language (UML), Message Sequence Charts (MSC) or Specification and Description Language (SDL). In using these notations to define requirements engineers tend to think of the most obvious behaviours of the system first (e.g. common system scenarios). However, depending upon the semantics of the graphical notation, specifications may also include non-obvious behaviours that at first glance are not apparent to the engineer when developing the specification.

During system development, software tools are frequently used to automatically generate test scripts from these specifications. These tools are capable of exploring all possible behaviours of the system, thereby producing potentially large numbers of tests; including both obvious and non-obvious scenarios. This can lead to the following problems:

1. Too many tests are generated. In this case there are physically too many tests to execute within the available time.
2. Some of the generated tests may be potentially redundant in the scope of black box testing, even though they represent valid behaviours of the system.

The invention described herein addresses these problems by:

1. Categorising tests that are automatically generated from a specification according to some suitable numerical weight. The idea is to define a weighting scheme that differentiates the most obvious behaviours from the non-obvious behaviours. In doing so, the user can quickly assess which test scripts represent the obvious behaviours, and consequently can concentrate on making sure that the most obvious behaviours are covered first.
2. Removing those tests that are not applicable in the scope of black box testing. For example, a test script that only contains events that are sent to the system under test is generally useless for black box testing.

Such techniques are also applicable during the:

- Determination of system scenarios for feature interaction studies – Feature interaction is a new area of study within the Motorola UK Research Lab.
- Debugging of system requirements

References

[1] Paul Baker, "*Generating Concurrent Test Scripts from Message Sequence Charts*", European Research Laboratory

[2] Bill Mitchell, "*Method for Generating Coordinating Messages for Distributed Test Scripts",* Motorola UK Research Lab technical report.

[3] Paul Baker, Paul Bristow, Clive Jervis, David King, Bill Mitchell, Chris Waters, "*The Design and Implementation of a Test Script Generator: ptk",* International Conference on Functional Programming.

[4] Gerard J. Holzmann, Doron A. Peled, and Margaret H. Redberg, "*Design Tools for Requirements Engineering*", Bell Labs Technical Journal, Winter 1997.

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

There are no really comparable technologies. Lucent have a tool (see [4]), which allows the user to analyse the requirements for racing conditions, but they do not provide any significant analysis beyond that.

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:

Notes: In general, the various graphical notations that are used for describing requirements can all be represented in terms of partial orders. We use the notation (a<b), to denote that a must happen before event b. As such a set of partial orders represents all the possible ways in which the events in a requirements specification can occur.

The invention is based upon the idea of using a set of partial orders to assign weights to a resulting set of tests scripts. Test scripts are then used to check that a system implementation conforms to its corresponding specification.

Firstly, we describe how to assign weights based on any arbitrary partial order, and then we explain how this technique can be applied to a particular partial order representing a requirements specification.

General Weights from Partial Orders

A relation < between pairs of elements of X is called a partial order when the following holds:

1. If $x<y$ and $y<z$ then $x<z$.
2. It is not the case that $x<x$ for any x in X.

Suppose we have a set of events E and for each event e we are given a partial order $<(e)$ of E. We will use these partial orders to define a weight for any sequence of events. This will generalise to test scripts since they can be described as sequences of events of the system.

First, we define a function *distance(m,a)*, which takes an event a and number m as input and returns, as output, the set of events b for which there exists distinct events a_i, for i ranging from 1 to m, such that:

$$a = a\_0 <(a)\ a\_1 <(a)\ \ldots <(a)\ a\_m = b$$

Next we define a function *d(a,b)*, which given events a and b occurring in a sequence s, defines the number of events to the right of a that are before b. If b does not occur to the right of a in s then this number is undefined.

Further, we define *n(s, a, m)* to be the maximum value for *d(a, b)* when $b \in distance(m,a)$. Notice that there may not be any such b, so this number is not always defined.

Finally, we define *n(s, a)* to be the value *n(s,a,m)*, where m is the first number for which *n(s,a,m)* is defined. If there is no value of m where *n(s, a, m)* is defined, then *n(s, a)* is defined to be infinite.

The weight *w(s)* of a sequence s is defined as the sum of the square of the values *n(s, a)* where a ranges over the send events in s. Notice that this definition depends on the partial orders $<(a)$ defined for each event a.

This completes the first part of the method that has been presented as a general method of defining a weight for a sequence of events based on a set of partial orders. The next part of the method defines partial orders based on the visual lay out given by a system requirement, the resulting weights can be used to categorise the test scripts for debugging purposes or test selection.

The Visual Weight

A requirements specification (e.g. Message Sequence Chart) states in what order events can occur with respect to one another. Hence, we can say that the requirements define a partial order < of the events contained within the system.

Firstly, we define an initial event to be any event x, where there is no event y where $y<x$. An initial event can be regarded as one of the first events that can happen according to the requirements specification.

Next we define the time step for an event x within the requirements as the maximum number of events which lie between it and any initial event. That is the time step of x is one less than the length of the longest sequence $a\_1 < a\_2 < \ldots\ < a\_n = x$, where a_1 can be any initial event. Intuitively the time step of x represents the first clock tick when x can occur, if we assume events are synchronised by some global clock, and all events fire at the first instance they can. We will say that two events are partners if they have the same time step (intuitively they can occur simultaneously under with the above clock scenario).

Finally, we define the partial orders $<(a)$ for each event a, which are then used to define the weights *w(s)* as above for any test

scripts, where $<(a)$ is defined as follows:

When the set of partners for a is not empty define $<(a)$ by:
1. $a<(a)p$, for any partner p of a,
2. $p<(a)a'$ for any event, where $a<a'$ and p is a partner of a,
3. $x<(a)y$ whenever x is not a and there is a sequence of inequalities:
   $$x < a\_1 < a\_2 < ... < a\_k < y$$
   Such that no two consecutive events are both send or receive events (They must switch from send to receive or vice versa).

When the set of partners for a is empty, define $<(a)$ by:
4. $x<(a)y$ whenever there is a sequence of inequalities: $x < a\_1 < a\_2 < ... < a\_k < y$
   Such that no two consecutive events are both send or receive events.

Example 1
Let us consider the directed graph illustrated in figure 1.

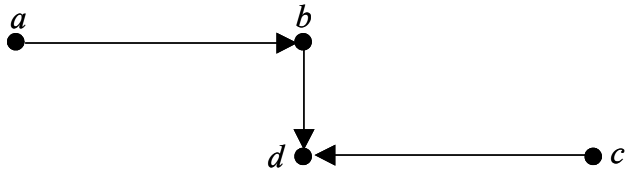

Figure 1

We can easily see from this graph that the initial event set is $\{a,c\}$, and the time steps for each event are calculated as $(a,0)$, $(b,1)$, $(d,2)$, and $(c,0)$. From this information we can determine that only events a and c have the same time step, hence are partners.

Given the partner information we can now define the partial orders for each event in the graph, using the rules 1-4 above:
$<(a) = (a<c), (c>b), (c<d)$
$<(b) = (c<d), (a<b)$
$<(c) = (c<a), (a<d), (a<b)$
$<(d) = (a<b), (c<d)$

Finally, we define the weight for a sequence s (which contains the letters a, b, c and d, in some order) as, $w(s)=n(s,a)^2 + n(s,c)^2$. For example, if we consider the sequence $<a,b,c,d>$ it would result in a weighting of 4.

Example 2
Lets consider the Message Sequence Chart (MSC) illustrated in figure 2.

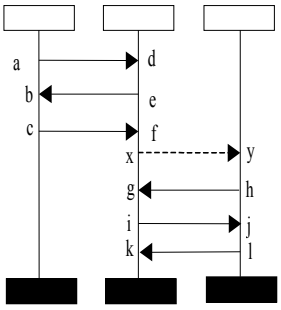

Figure 2

The dotted arrow represents a message which we will add shortly, but which we do not consider initially. A possible test behaviour represented by this example could be:

$$<a, d, e, b, c, f, h, g, i, j, l, k>$$

This sequence follows the visual order of events. Working out the weight for this sequence of events yields the value 36. However, if we introduce the dotted arrow into the requirements then we can have the sequence:

$$\langle a, d, e, b, c, f, x, y, h, g, i, j, l, k \rangle$$

This sequence has the weight 0. The reason the weight was initially so high is that *a* and *h* were partners because they both had time step 0. Whereas, in the second trace *h* is no longer a partner of *a*, it now has time step 8, and *a* has time step 0. What this example illustrates is that even though a specification appears to be fairly obvious, that by using weights we can provide the user with information that can be used during the validation of system requirements.

Example 3

Lets consider the case where we want to test the implementation of the system – called the System Under Test (SUT). In such cases, test scripts can only verify the observable events - those events that can be physically monitored by the test system. In order to monitor such events a test script will generally send some event *e* that acts as trigger for the SUT to respond with an appropriate message. Generally, it does not make sense to have test scripts that send messages to the SUT, for which no subsequent response is received. For such tests sequences this method would yield an infinite weighting because there would be no event *b* further down the trace such that $e <_{(e)} b$. This is a very clear signal that the test may be wrong. The test script contains a message for which it will not be able to verify the response of the SUT.

Benefits/Impact

It is commonly understood that testing can consume 50% of the overall development time of a new system. Using this method we can provide engineers with information that is useful for validating system requirements, as well as selecting the most obvious traces first.

----------END----------