Deriving Real-time Monitors from System Requirements Documentation

Dennis K. Peters
CRL, McMaster University
Hamilton, ON Canada L8S 4K1
e–mail: peters@mcmaster.ca

Supervisor: David L. Parnas

Abstract

During system testing, determining if the observed behaviour of a real–time system is consistent with its requirements specification can be difficult. I propose that a system to check the behaviour against the specification, a monitor*, be automatically derived from the requirements documentation. The monitor would model the system requirements as a modified finite state automaton in which the states represent equivalence classes of system histories and transitions are labelled with predicates such that it accepts only executions representing acceptable system behaviour. Investigation into the design of such a monitor, and the process for automatically generating it from reviewable requirements documentation is on–going.*

1. Problem Statement

The process of testing a real–time system typically involves running the system in a test environment, observing its behaviour and comparing it to that required by its specification. In general, making this comparison can be quite difficult since the requirements may be complex, possibly including time constraints and interdependencies. A *monitor* is a system that automatically determines if the observed behaviour is consistent with a given specification.

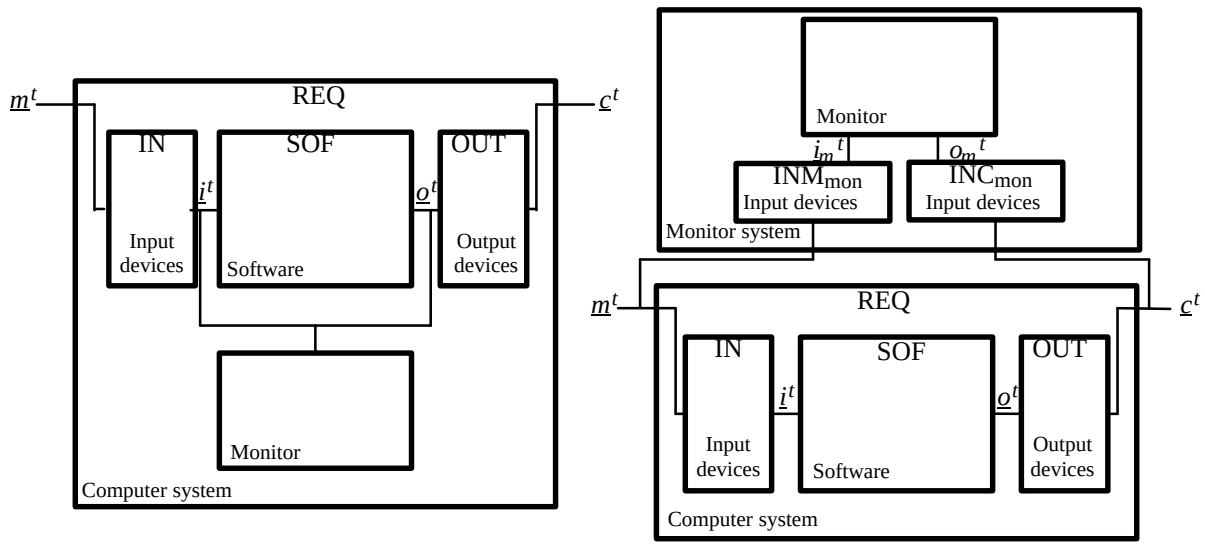
When designing safety– or mission–critical systems, good engineering practice dictates that a clear, precise and unambiguous specification of the required behaviour of the system be produced and reviewed for correctness by experts in the domain of application of the system. Research has demonstrated that such reviews are effective if the system behavioural requirements documentation is written such that:

- it expresses the required behaviour in terms of the quantities from the environment that are monitored and/or controlled by the system,
- it uses terminology and notation that is familiar to, or easily understood by, the domain experts, and
- it is presented in a manner that permits independent review of small parts of the document.[5]

As discussed in [4], [9], [12] and [13], a *(relational) system requirements document* describes a relation, REQ, on vector functions of time representing the environmental quantities that are monitored and controlled by the system. I intend to explore techniques for using reviewable forms of such documentation (i.e. satisfying the above three criteria) to generate a software monitor that will determine if the observed behaviour of some software is consistent with that expressed in the documentation. Such a monitor would be useful, during system testing, for determining if the system is operating correctly, or, in certain safety–critical applications, it may be useful as a redundant monitoring system during operation.

Through this research I hope to answer the following questions:

1. How can a monitor be used to verify conformance with relational requirements documentation?
2. What are the useful classes of behavioural properties that can and cannot be:
   a) specified in relational documentation?
   b) verified using a monitor as described above?
3. Under what conditions can an effective monitor be produced automatically from a relational requirements document? What restrictions on the form or content of the documentation must be imposed?
4. What is the cost (computational and space complexity) of using such a monitor? Are there some optimizations that can be done to reduce this complexity or restrictions on the documentation that will ensure that the complexity is tractable?



A: Internal Monitor **B: External monitor**

Key

The acceptable behaviour of component "name" is described by the relation "REL".

The data represented by x^t is either input or output for the component "name".

Figure 1 – Possible Monitor Configurations

2. Background

The vector functions of time representing the quantities monitored and controlled by the system are denoted by m^t and c^t, respectively. These environmental quantities, in general, cannot be observed directly by monitor software, but must be observed through the monitor system's own input devices (Figure 1B), or surmised by observing the system software input and output values, i^t and o^t (Figure 1A). In order to determine if observed behaviour is acceptable with respect to the system requirements, REQ, the input relation, IN, which characterizes the possible values of i^t for any instance of m^t, and the output relation, OUT, which characterizes the possible values of c^t for any instance of o^t, must be used.

As illustrated in [5], the required system behaviour can be effectively described by describing the values of the controlled quantities in terms of *conditions*—predicates that characterize some aspect of m^t or c^t for a measurable period of time—and *events*—changes in values of conditions. This approach has been shown to be effective for a number of realistic examples (e.g. [2], [5] and [7]) and to satisfy industrial expectations of requirements documentation.[1] A tool–set for specifying and analyzing requirements has been developed [3], but this tool–set does not at this time support automated testing.

3. Solution Approach

The proposed solution addresses this problem in two stages: 1) development of a general design of a monitor (i.e. a *family* of monitors[8]) and, 2) design of a generator to produce instances of the family from relational requirements documentation. Some of the issues to be faced in each of these, and some proposed solutions to them, are discussed in this section.

3.1 Monitor Design

In general, a monitor must detect the occurrence of events relevant to the system and, considering those events that have occurred previously (i.e. the execution so far), determine if the events and the values of the controlled quantities represent acceptable behaviour.

3.1.1 Monitor Input

Since my goal is to automatically derive the monitor from documentation, and it is hoped that the generation tool will be applicable to a wide variety of systems, I will not attempt to generate a general, non–intrusive, mechanism for detecting events, but rather will assume that the monitor will be given, as input, a sequence of 'samples' of the values of $\underline{i}^t$ and $\underline{o}^t$, or $\underline{i}_m{}^t$ and $\underline{o}_m{}^t$, (i.e. $(\underline{i}^t(t_j), \ \underline{o}^t(t_j))$ for j = <1, … , n>). The generation of this sequence is the responsibility of the monitor 'harness' or interface system, which must be designed to meet the specific restrictions imposed by the testing/operation environment. Some possible approaches to this are as follows.

1. Modify the behaviour of the system under test, or the systems that interact with it, to have them report when they detect or generate events. This is only applicable when the sequence of events fully describes the system behaviour.

2. Sample $\underline{i}^t$ and $\underline{o}^t$ at regular intervals, presumably fast enough to detect all events of interest.

3.1.2 Execution History

The proposed approach to evaluating and maintaining the required information about the execution history is to model the system requirements specification as a modified finite state automaton (FSA) that accepts only those sequences of events that represent acceptable behaviour. Each state in the FSA represents an equivalence class of executions (i.e. histories) of the system. Each transition in the FSA is labelled by a predicate (the *transition predicate*) and represents a set of events that have a similar interpretation with respect to the specification (e.g. an acceptable output being produced within an acceptable time limit). The exact form of this FSA is a topic of current research, but I expect that it will include variables representing the values of $\underline{i}^t$, $\underline{o}^t$ and time at the point where the FSA enters each state. The transition predicates will be expressed in terms of these variables (e.g. a transition predicate may be "Output O_1 occurs within T_1 seconds after the occurrence of its triggering event, I_1").

3.1.3 Event Detection

For each input sample, the monitor will evaluate the values of all conditions and compare them to the values of the conditions in the previous state to determine which, if any, have changed, and hence what event(s) have occurred. Note that since a condition may be dependent on time alone (e.g. the amount of time elapsed since the occurrence of an event), it may be necessary for the monitor to independently detect changes in such conditions (i.e. based on an internal clock) and respond appropriately to the event. The transition predicates of the FSA will then be evaluated to determine if the event occurrence is acceptable at this time, and how the FSA state should change in response to it.

3.1.4 Controlled Variables

Each input sample will also be used to verify that the values of the controlled variables are correct with respect to the current FSA state and the values of the monitored variables. This will be done by evaluating the appropriate predicate characterizing the value of each controlled variable.

3.2 Monitor Generation

Generating the monitor involves analysis of the requirements document to determine the set of states of the FSA (i.e. an appropriate set of equivalence classes of executions), the transition predicates for each transition and the predicates characterizing the values of the controlled variables in each state. A few of the issues to be considered in this area are as follows.

3.2.1 State Enumeration

The exact procedure for deriving the set of states is still under investigation, but I expect it will draw on state exploration and reduction techniques such as those presented in [6]. Note that for documentation techniques, such as SCR, that model the requirements using parallel state machines (*mode classes* in SCR), the monitor FSA will represent the parallel composition of those state machines. For example, the set of states in the FSA derived from an SCR requirements specification would be related to the cross product of the mode classes in the specification. Note also that, since the real–time aspects of the system need to be monitored, the FSA will include states representing the period between occurrence of an input event and the output event that it triggers.

3.2.2 Transition and Controlled Value Predicates

As mentioned above, the monitor cannot, in general, directly access the monitored and controlled quantities, so it will see only *internal values*—values of $\underline{i}^t$ or $\underline{o}^t$. Since IN and OUT are relations, each internal value represents a set of possible external values, so the monitor will need to determine if any and/or all of the elements of that set represent acceptable behaviour, depending on how 'pessimistic' the user wants the monitor to be. If the IN^{-1} and OUT relations are 'well behaved' over the domain of interest then this can be done by choosing the worst and/or best case when generating the transition and controlled value predicates.

3.2.3 Predicate Evaluation

In my previous work [10], [11], I have shown how an oracle can be automatically generated from relational program documentation, which gives the characteristic predicate of the relation describing the acceptable start and stop state pairs for a single program. This oracle determines if the values of the program variables in the starting and stopping states of the program execution are in the relation by evaluating its characteristic predicate. A major component of that

work is a procedure for generating C code to evaluate the characteristic predicate of a relation. That procedure can be used again here to generate code for evaluation of both the transition predicates in the FSA implementation and the predicates characterizing acceptable values of controlled variables.

4. Contributions

The primary contribution of this work will be a method of using a reviewable system requirements document to automatically generate a monitor that can determine if observed 'internal' behaviour is consistent with the requirements. Secondary contributions include a statement of the expressiveness of the system requirements specification technique used, and a characterization of the class of system requirements specifications that can be used to generate an effective monitor.

5. Progress

This research is still in its preliminary stages. I am investigating existing methods for specifying system requirements, hybrid automatas and methods for generating real–time oracles.

6. References

[1] S. R. Faulk, J. Brackett, P. Ward and J. Kirby Jr., "The CORE Method for Real–Time Requirements", *IEEE Software*, vol. 9, no. 6 (September 1992), pp. 22–33.

[2] C. L. Heitmeyer and J. McLean, "Abstract Requirements Specification: A New Approach and Its Application,'' *IEEE Transactions on Software Engineering*, vol. 9, no. 5 (September 1983), pp. 580–589.

[3] C. L. Heitmeyer, A. Bull, C. Gasarch and B. G. Labaw, "SCR*: A Toolset for Specifying and Analyzing Requirements," *Proceedings of the Tenth Annual Conference on Computer Assurance (COMPASS '95)*, Gaithersburg, MD, June 25–29, 1995, pp. 109–122.

[4] K. Heninger, J. Kallander, D. L. Parnas and J. E. Shore, "Software Requirements for the A–7E Aircraft", *Naval Research Laboratory Memorandum Report 3876*, November 1978.

[5] K. Heninger, "Specifying Software Requirements for Complex Systems: New Techniques and their Applications", *IEEE Transactions on Software Engineering*, vol. SE–6, no. 1 (January 1980), pp. 2–13.

[6] I. Kang and I. Lee, "An Efficient State Space Generation for Analysis of Real–time Systems", *Proceedings of the International Symposium on Software Testing and Analysis (ISSTA)*, published in *Software Engineering Notes*, vol. 21, no. 3 (May 1996), pp. 4–13.

[7] J. Kirby, "Example NRL/SCR Software Requirements for an Automobile Cruise Control and Monitoring System", *Technical Report TR–87–07*, Wang Institute of Graduate Studies, July 1987.

[8] D. L. Parnas, "On the Design and Development of Program Families", *IEEE Transactions on Software Engineering*, vol. SE–2, no. 1 (March 1976), pp. 1–9.

[9] D. L. Parnas and J. Madey, "Functional Documentation for Computer Systems Engineering", *Science of Computer Programming* (Elsevier), vol. 25, no. 1 (October 1995), pp. 41–61.

[10] D. K. Peters and D. L. Parnas, "Generating a Test Oracle from Program Documentation—work in progress", *Proceeding of the 1994 International Symposium on Software Testing and Analysis (ISSTA)*, (August, 1994), pp. 58–65.

[11] D. K. Peters, *Generating a Test Oracle from Program Documentation*, M.Eng. Thesis, Department of Electrical and Computer Engineering, McMaster University, Hamilton, ON (April, 1995). 97 pgs. Also printed as *CRL Report No. 302*, Telecommunications Research Institute of Ontario (TRIO).

[12] A. J. van Schouwen, "The A–7 Requirements Model: Re–examination for Real–Time systems and An Application to Monitoring Systems", *Technical Report TR 90–276*, Queen's University, Kingston, Ontario, 1990.

[13] A. J. van Schouwen, D. L. Parnas and J. Madey, "Documentation of Requirements for Computer Systems", *Proceedings of the IEEE International Symposium On Requirements Engineering*, January 1993, pp. 198–207.