Automated Testing of Real-Time Systems

Dennis K. Peters
Faculty of Engineering and Applied Science
Memorial University of Newfoundland

November 10, 1999

Abstract

A fundamental, but often neglected, requirement for automated testing of software and systems is that there must be an oracle—a means of determining if the behaviour exhibited by the system under test is acceptable or not. Testing of real-time systems requires that the oracle check that the system satisfies both behavioural and timing constraints, so human oracles are typically insufficient.

In this talk I will briefly present a real-time system specification technique that is both understandable to domain experts, and precise enough to be used to automatically generate an oracle. I will illustrate how this technique has been used to find previously undetected errors in a real-time system.

1 Introduction

Computer systems are increasingly being used in situations where their correct behaviour is essential for the safety of people, equipment, the environment and businesses. In many cases there are real-time requirements on the behaviour of these systems—failure to satisfy timing constraints is as costly as responding incorrectly.

When designing such safety- or mission-critical real-time 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.

After the system has been implemented, it should be tested to ensure that its behaviour satisfies the requirements. Testing requires that there be an *oracle*—a means of determining if the observed behaviour is acceptable or not.[1] In the case of real-time systems, the oracle problem can be quite complex since the requirements may be extensive and include strict timing restrictions. One solution to this problem is to use a *monitor*: a system that observes and analyses the behaviour of another system (the *target system*).

If the system requirements documentation is written in a form that has precise interpretation, then it should be possible to use this documentation to help produce a monitor. This paper presents some results from an investigation of techniques for automatically generating such a monitor from system requirements documentation.

2 System Requirements

As pointed out, e.g. in [2, 3], it is important when specifying system and software requirements to distinguish quantities that are in the environment, i.e., external to the system, from those that are internal to the system or artifacts of the description of either the requirements or the design. According to the "Four Variable Model"[2, 4], environmental quantities are those that are independent of the chosen solution and are apparent to the "customer"; they are the best quantities to use when describing the requirements for the system. These quantities, which can be modelled as functions of time, will include such things as physical properties (e.g., temperature, pressure, location of objects), values or images displayed on output display devices, settings of input switches, dials etc., and settings of controlled devices. An *environmental state function* is a function of time that gives the value of the relevant environmental quantities during system operation, and thus represents the behaviour of the system.

The environmental quantities can be classified into two (not necessarily disjoint) sets: the *controlled* quantities—those that the system may be required to change the value of, and the *monitored* quantities—those that should affect the behaviour of the system. The *monitored* and *controlled state functions* are derived from the environmental state function by including only the monitored, or controlled quantities, respectively, and are denoted $\underline{m}^t$ and $\underline{c}^t$.

The *system behavioural requirements* characterize the set of acceptable behaviours. Since the system is expected to observe the monitored quantities and control the values of the controlled quantities accordingly, it is natural to express this as a relation, **REQ**, which is a subset of the possible environmental state functions. A behaviour is acceptable if and only if REQ($\underline{m}^t, \underline{c}^t$) is $\underline{\textit{true}}$. Note that, since implementations will invariably introduce some amount of unpredictable delay, or inaccuracy in the measurement, calculation, or output of values, **REQ** will not be functional for real systems, i.e., there will be more than one acceptable $\underline{c}^t$ for a given $\underline{m}^t$.

Specification Technique

In this work, system requirements are documented using a technique that is based on the method presented in [2], which developed from a project at the US Naval Research Laboratory (NRL) to specify the requirements for the operational flight program of the A-7E aircraft[5]. Continued work at the NRL in this area has resulted in the "Software Cost Reduction" (SCR) approach[6], which uses a similar notation and interpretation, and has been shown to be effective for a number of real examples. A tool-set for specifying and analysing requirements documents that are written using the NRL version of this approach has been developed.

Readers interested in a complete description of these specification techniques are referred to [6, 7]. The remainder of this section highlights some of the main points.

All systems are required to change the value of their controlled quantities in response to changes in the monitored quantities, e.g., a user pushing a button to request some action, or a physical quantity crossing some threshold. As illustrated in [8], such behaviour can often be effectively described in terms of *conditions*—predicates defined in terms of the environmental state function—and *events*—changes in value of one or more conditions at some instant. For conditions c_1 and c_2, the notations @T (c_1) and @F (c_1) are used to denote the events of c_1 becoming true or false, respectively, and @T (c_1) WHEN(c_2), and @F (c_1) WHEN(c_2) denotes the events of c_1 becoming true or false, respectively, under the constraint that c_2 is true.

Since, it is frequently the case that many histories are equivalent with respect to their impact on future behaviour, and each controlled quantity is usually affected by a small number of conditions, the requirements can often be concisely described using mode classes, as follows.

A *mode class* is an equivalence relation on histories of the system, which partitions the set of possible histories into equivalence classes, known as the *modes* of that mode class. The system response to events is the same for all histories that are in the same mode. Mode classes can be modelled by finite state machines (FSM) in which the states represent modes, and the state transition relation gives the required response (mode change) to events.

The value of controlled values are expressed as relations in terms of the current mode and the values of monitored quantities. For example, in Figure 1, the value of the Boolean controlled quantity, ${}^{mc}\mathsf{penDown}$, is required to be $\underline{\textit{true}}$ when the system is in modes ${}^{Md}\mathsf{Forward}$ or ${}^{Md}\mathsf{Reverse}$, and $\underline{\textit{false}}$ otherwise.

$^{mc}\mathsf{penDown}$

$=$

$\mathrm{p_T} : H_2$ $\mathrm{r_T} : H_1 = G$ Vector	$^{Md}\mathsf{Forward} \vee {}^{Md}\mathsf{Reverse}$	$^{Md}\mathsf{Starting} \vee {}^{Md}\mathsf{Home} \vee {}^{Md}\mathsf{Done}$
$^{mc}\mathsf{penDown}$	$\underline{\mathit{true}}$	$\underline{\mathit{false}}$

Figure 1: Controlled Value Relation

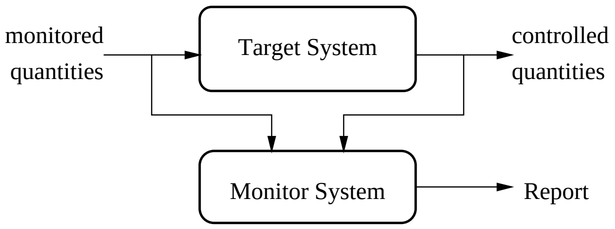

Figure 2: System Monitor

3 Monitors for Real-Time Systems

Testing a real-time system typically involves running the target system in a test environment, observing its behaviour and comparing it to that required by its specification. Making this comparison can be quite difficult since the requirements may be complex. As illustrated in Figure 2, a *monitor* is a system that observes the behaviour of a system and determines if it is consistent with a given specification. That is, an ideal monitor reports the value of REQ $(\underline{m}^t, \underline{c}^t)$. A monitor can be used to check the behaviour of a target system either concurrently with the target system or post-facto, using some form of recording of the behaviour.

In general, the monitor software cannot directly observe the environmental quantities, but must make use of some input devices, which necessarily introduce some measurement error and delay. The effect of these errors is to create situations in which the monitor cannot determine if a particular observation represents an acceptable behaviour or not. Depending on how the measurement errors are accounted for in the design of the monitor software, it may report either “false negatives” or “false positives” for some situations. The accuracy of a monitor can thus be characterized by the size of the set of false positives or false negatives that it may report. Interested readers are referred to [7] for a more complete discussion of these issues.

When testing real-time systems, the presence of a monitor that can act as an oracle significantly reduces the cost and improves the usefulness of the testing. Non-automated methods of determining the acceptability of system behaviour are highly error-prone and time-consuming, and are unlikely to be able to reliably detect subtle or short-duration errors.

The requirements specification techniques discussed above are sufficiently precise to allow monitor software to be automatically generated from the documentation. We have developed a tool to generate such a monitor and have applied it to a realistically sized robotics example, which was used as a course project at McMaster University[9]. Using these techniques, previously undetected, short-duration errors in the behaviour of the target system were found. The performance of the monitor was fast enough to keep pace with the robot movement and complete the analysis for each event before the next one occurred.

4 Conclusions

A specification of acceptable system behaviour is essential before a system can be effectively tested. This work has demonstrated that, if the specification is written in a notation that has a precise interpretation, then it can be used to automatically produce a monitor, which can act as an oracle. This offers significant improvements over methods where the monitor is implemented manually since testers can be assured that the oracle and the documentation agree, and any changes in the requirements can be quickly translated into a new monitor. Such testing is faster and more reliable than when non-automated oracles are used.

References

[1] E. J. Weyuker, "On testing non-testable programs," *The Computer Journal*, vol. 25, no. 4, pp. 465–470, 1982.

[2] A. J. van Schouwen, "The A-7 requirements model: Re-examination for real-time systems and an application to monitoring systems," Tech. Rep. TR 90-276, Queen's University, Kingston, Ontario, 1990. also printed as CRL Report No. 242, Telecommunications Research Institute of Ontario (TRIO).

[3] P. Zave and M. Jackson, "Four dark corners of requirements engineering," *ACM Trans. Software Eng. and Methodology*, vol. 6, pp. 1–30, Jan. 1997.

[4] D. L. Parnas and J. Madey, "Functional documentation for computer systems," *Science of Computer Programming*, vol. 25, pp. 41–61, Oct. 1995.

[5] T. A. Alspaugh, S. R. Faulk, K. H. Britton, R. A. Parker, D. L. Parnas, and J. E. Shore, "Software requirements for the A-7E aircraft," Tech. Rep. NRL/FR/5546-92-9194, Naval Research Lab., Washington DC, 1992.

[6] C. L. Heitmeyer, R. D. Jeffords, and B. G. Labaw, "Automated consistency checking of requirements specifications," *ACM Trans. Software Eng. and Methodology*, vol. 5, pp. 231–261, April-June 1996.

[7] D. K. Peters, *Deriving Real-Time Monitors from System Requirements Documentation.* PhD thesis, McMaster University, Hamilton ON, to appear 1999.

[8] K. L. Heninger, "Specifying software requirements for complex systems: New techniques and their application," *IEEE Trans. Software Engineering*, vol. SE-6, pp. 2–13, Jan. 1980.

[9] M. von Mohrenschildt and D. K. Peters, "The draw-bot: A project for teaching software engineering," in *Proc. Frontiers in Education Conf. (FIE '98)*, pp. 1022–1027, American Society for Engineering Education, Nov. 1998.