# Requirements-based Monitors for Real-Time Systems

Dennis K. Peters
Electrical and Computer Engineering
Faculty of Engineering and Applied Science
Memorial University of Newfoundland
St. John's, Newfoundland
Canada
dpeters@engr.mun.ca
http://www.engr.mun.ca/~dpeters

David L. Parnas
Department of Computing and Software
Faculty of Engineering
McMaster University
Hamilton, Ontario
Canada

## ABSTRACT

Before designing safety- or mission-critical real-time systems, a specification of the required behaviour of the system should be produced and reviewed by domain experts. After the system has been implemented, it should be thoroughly tested to ensure that it behaves correctly. This is best done using a monitor, a system that observes the behaviour of a target system and reports if that behaviour is consistent with the requirements. Such a monitor can be used both as an oracle during testing and as a supervisor during operation. Monitors should be based on the documented requirements of the system.

If the target system is required to monitor or control real-valued quantities, then the requirements, which are expressed in terms of the monitored and controlled quantities, will allow a range of behaviours to account for errors and imprecision in observation and control of these quantities. Even if the controlled variables are discrete valued, the requirements must specify the timing tolerance. Because of the limitations of the devices used by the monitor to observe the environmental quantities, there is unavoidable potential for false reports, both negative and positive.

This paper discusses design of monitors for real-time systems, and examines the conditions under which a monitor will produce false reports. We describe the conclusions that can be drawn when using a monitor to observe system behaviour.

## Categories and Subject Descriptors

D.2.5 [**Software Engineering**]: Testing and Debugging—*Monitors*; C.3 [**Computer Systems Organization**]: Special-Purpose and Application-Based Systems—*Real-time and embedded systems*

## Keywords

Automated testing, Test oracle, Real-time system, Supervisor

## 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. Research suggests that such reviews are more effective if the system behavioural requirements documentation:

- expresses the required behaviour in terms of the quantities from the environment in which the system operates (i.e., external to the system),
- uses terminology and notation that is familiar to, or can be learned by, the domain experts, and
- is structured to permit independent review and application of small parts of the document.[3]

After the system has been implemented, it should be tested to ensure that its behaviour satisfies the requirements. In safety-critical applications the system should be monitored by an independent safety system to ensure continued correct behaviour. To achieve these goals there must be a means of quickly determining if the observed behaviour is acceptable or not; this can be quite difficult for complex real-time systems. Several authors (e.g., [13]) have suggested that a practical approach to analysing the behaviour of a real-time system is to use a *monitor*: a system that observes and analyses the behaviour of another system (the *target system*). Such a monitor could be used either as an 'oracle'[16] during system testing, or as a 'supervisor'[11] to detect and report system failure during operation.

This paper examines the relationship between the target system and the monitor, in particular with respect to the means by which the monitor observes the system behaviour, and the impact of this on the usefulness of the monitor. It gives some necessary conditions for monitor feasibility. In related work [7] we have developed techniques for automatically generating monitor software from a System Requirements Document (SRD) written in a notation that is based on the method presented in [14], which developed from the A-7E project at the US Naval Research Laboratory.[4]

This work focuses on monitors for computer-based systems that are intended to observe and/or control some quantities external to the computer. Such quantities are often best represented by continuous, rather than discrete, valued functions. In particular, the requirements for any real-time systems will include time, which we model as a continuous variable.

The remainder of this section defines the notation and terminology used in this paper. Section 2 briefly presents the "Four Variable Model", which relates system and software requirements in terms of the behaviour of the input and output devices. Section 3 formally defines a monitor and its accuracy, and discusses possible monitor configurations. Section 4 discusses the impact of realistic monitor input devices on the conclusions that can be drawn from using the monitor, and gives some necessary conditions that must be satisfied in order for a monitor to be useful. The final section draws some conclusions and suggests future work.

## 1.1 Notation and Terminology

There are at least two systems of interest in any application of this work:

- The *target system* is the system to be monitored. Its required behaviour is specified in the System Requirements Document (SRD).
- The *monitor system* is the system that observes the behaviour of the target system and reports whether or not it conforms to the SRD.

As discussed further in Section 3.2, in some configurations these two systems may share components.

To help simplify the text, font faces and notational conventions are used. These are illustrated in Table 1. The symbol "$\overset{\text{df}}{=}$" is used to represent "is defined as", so, for example "$\mathrm{f}(x) \overset{\text{df}}{=} x + 5$" defines the function f. The common bracketing notation for describing an open or closed range of real numbers is used:

$$\begin{aligned}
[x,y] &= \{z \in \mathbf{Real} \mid x \leq z \leq y\} \\
(x,y) &= \{z \in \mathbf{Real} \mid x < z < y\} \\
(x,y] &= \{z \in \mathbf{Real} \mid x < z \leq y\} \\
[x,y) &= \{z \in \mathbf{Real} \mid x \leq z < y\}
\end{aligned}$$

This bracketing notation is extended to ranges of fixed-precision numbers by replacing "," with "...", so, for example, $[0.0 \ldots 0.7] = \{0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7\}$.

### 1.1.1 Predicates and Relations

For a set, $\mathbf{R}$, the *characteristic predicate*, R, is the predicate such that $\mathrm{R}(x)$ is $\underline{true}$ if and only if $x \in \mathbf{R}$, i.e., $\mathrm{R}(x) \Leftrightarrow x \in \mathbf{R}$. We say that the predicate R *characterizes* the set $\mathbf{R}$. Table 2 gives notation used for standard operations on binary relations.

## 2. THE FOUR VARIABLE REQUIREMENTS MODEL

As pointed out, e.g. in [14, 15, 17], 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. The "Four Variable Model", introduced in [5, 14, 15], addresses this issue and is adopted here as a framework for describing requirements. According to this model, 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. (The requirements for the *software* alone can be expressed in terms of variables internal to the system, see Section 2.4.) These quantities 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 states of controlled devices.

It is widely accepted (e.g., see [4, 5, 9, 15]) that environmental quantities can be modelled by functions of time. Given the environmental quantities relevant to a particular system, $q_1, q_2, \ldots, q_n$, of types $\mathbf{Q_1}, \mathbf{Q_2}, \ldots \mathbf{Q_n}$, respectively, we can represent the behaviour of the system in its environment by an *environmental state function*, $\mathrm{S} : \mathbf{Real} \to \mathbf{Q_1} \times \mathbf{Q_2} \times \ldots \times \mathbf{Q_n}$, defined on all intervals of system operation. For convenience we define $\mathbf{St} \overset{\text{df}}{=} \mathbf{Q_1} \times \mathbf{Q_2} \times \ldots \times \mathbf{Q_n}$ (i.e., $\mathbf{St}$ is the set of possible environmental states).

The environmental quantities of interest 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.[1] Assuming that the monitored quantities are $q_1, q_2, \ldots, q_i$, the *monitored state function*, $\underline{m}^t : \mathbf{Real} \to \mathbf{Q_1} \times \mathbf{Q_2} \times \ldots \times \mathbf{Q_i}$, is derived from the environmental state function by including only the monitored quantities. Similarly, if the controlled quantities are $q_j, q_{j+1}, \ldots, q_n$, the *controlled state function*, $\underline{c}^t : \mathbf{Real} \to \mathbf{Q_j} \times \mathbf{Q_{j+1}} \times \ldots \times \mathbf{Q_n}$, is derived. In this paper, a pair of functions $(\underline{m}^t, \underline{c}^t)$ will denote an environmental state function. With respect to a particular target system, $\mathbf{M}$ denotes the set of functions of type $\mathbf{Real} \to \mathbf{Q_1} \times \mathbf{Q_2} \times \ldots \times \mathbf{Q_i}$, (type correct for a monitored state function) and $\mathbf{C}$ denotes the set of functions of type $\mathbf{Real} \to \mathbf{Q_j} \times \mathbf{Q_{j+1}} \times \ldots \times \mathbf{Q_n}$ (type correct for a controlled state function).

We usually are only interested in the environmental state function during the periods when the system is operating (i.e., it is turned on). An environmental state function defined on the (possibly infinite) time interval of a single execution of the system is known as a *behaviour* of the system. A behaviour is *acceptable* if it describes a situation in which the system is operating in a manner that is consistent with the requirements.

[1]There may also be environmental quantities that are neither monitored nor controlled but are relevant to the design or analysis of the system. These quantities are not relevant to the the four-variable model and so are not considered in this presentation.

Table 1: Fonts and Notational Conventions

| Item | Notation | Example |
|---|---|---|
| function or predicate | math roman | func |
| standard function | italicised | *kfunc* |
| sequence | bold | **seq** |
| set | math bold, capital | $\mathbf{R}$ |
| characteristic predicate of $\mathbf{R}$ | set name as predicate | $\mathrm{R}(x)$ |
| tuple | bracketed | (x, y) |
| vector function of time | underlined, superscript $t$. | $\underline{m}^t$ |

Table 2: Operations on Binary Relations

| Operation | Definition |
|---|---|
| Domain | $domain(\mathbf{R}) \overset{\mathrm{df}}{=} \{x \mid \exists y, \mathrm{R}(x,y)\}$ |
| Range | $range(\mathbf{R}) \overset{\mathrm{df}}{=} \{y \mid \exists x, \mathrm{R}(x,y)\}$ |
| Inverse | $\mathbf{R}^{-1} \overset{\mathrm{df}}{=} \{(x,y) \mid \mathrm{R}(y,x)\}$ |
| Composition | $\mathbf{R_1} \circ \mathbf{R_2} \overset{\mathrm{df}}{=} \{(x,y) \mid \exists z, \mathrm{R_1}(x,z) \wedge \mathrm{R_2}(z,y)\}$ |

## 2.1 System Requirements

The *system behavioural requirements* (or, where the meaning is clear from context, *system 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, $\mathbf{REQ} \subseteq \mathbf{M} \times \mathbf{C}$. A behaviour is acceptable if and only if $\mathrm{REQ}(\underline{m}^t, \underline{c}^t)$ is $\underline{true}$. 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 $\underline{m}^t$.

In many cases **REQ** will be independent of the actual date and time, and will depend only on the time elapsed since some event (e.g., the system being turned on). In these cases, equivalence classes of behaviours can be represented by the behaviour formed by translation along the time axis such that time = 0 corresponds to that event. In cases where aspects of the date or time (e.g., day of month, hour of day) are significant, time = 0 will need to be chosen to correspond to an appropriate clock time and appropriate functions defined to determine the needed quantities.

## 2.2 Environmental Constraints

The possible values of the environmental state function are constrained by physical laws independent of the system to be built. For example,

- the rate of change (or higher order derivatives) of a quantity may be constrained by some natural laws,
- some quantities may be related to each other (e.g., pressure and temperature in a closed container),
- values may be only able to change in certain ways (e.g., positions of selector switches), or
- certain events may not be able to occur simultaneously.

These laws are described by the relation $\mathbf{NAT} \subseteq \mathbf{M} \times \mathbf{C}$, which contains all values of $(\underline{m}^t, \underline{c}^t)$ that are possible in the environment.

## 2.3 System Design

The environmental quantities cannot usually be directly observed or manipulated by the system software, but must be measured or controlled by some devices (e.g., sensors, actuators, relays, buttons), which communicate with the software through the computer's input or output registers, represented by program variables. The *input* quantities are those program variables that are available to the software and provide information about the monitored quantities. For input quantities, $i_1, i_2, \ldots, i_n$, of types $\mathbf{I_1}, \mathbf{I_2}, \ldots, \mathbf{I_n}$, respectively, an *input state function* is a function, $\underline{i}^t : \mathbf{Real} \to \mathbf{I_1} \times \mathbf{I_2} \times \ldots \times \mathbf{I_n}$, representing the values of the input quantities during system operation. Similarly, the *output* quantities are those program variables through which the software can change the value of the controlled quantities. For output quantities, $o_1, o_2, \ldots, o_m$, of types $\mathbf{O_1}, \mathbf{O_2}, \ldots, \mathbf{O_m}$, respectively, an *output state function* is a function, $\underline{o}^t : \mathbf{Real} \to \mathbf{O_1} \times \mathbf{O_2} \times \ldots \times \mathbf{O_m}$, representing the values of the output quantities during system operation. For convenience, with respect to a particular system being specified, the set of functions of type $\mathbf{Real} \to \mathbf{I_1} \times \mathbf{I_2} \times \ldots \times \mathbf{I_n}$ is denoted $\mathbf{I}$, and the set of functions of type $\mathbf{Real} \to \mathbf{O_1} \times \mathbf{O_2} \times \ldots \times \mathbf{O_m}$ is denoted $\mathbf{O}$. The behaviour of the interface between the environment and the software is described by the input relation, $\mathbf{IN} \subseteq \mathbf{M} \times \mathbf{I}$, which characterizes the possible values of $\underline{i}^t$ for any instance of $\underline{m}^t$, and the output relation, $\mathbf{OUT} \subseteq \mathbf{O} \times \mathbf{C}$, which characterizes the possible values of $\underline{c}^t$ for any instance of $\underline{o}^t$. This is illustrated in Figure 1.

## 2.4 Software Requirements

In [5] the actual software behaviour is described by the *software behaviour relation*, **SOF**, and an expression is given for software acceptability. In this work, as in [2], we are interested in characterizing all acceptable software, so we use the *software requirements relation*, **SOFREQ**, which characterizes the set of acceptable behaviours of the software—those pairs, $(\underline{i}^t, \underline{o}^t)$, such that any possible environmental state function, with respect to the given input and output devices and environmental constraints, are acceptable. This is fully determined by **REQ**, **IN**, **OUT** and **NAT**, as fol-

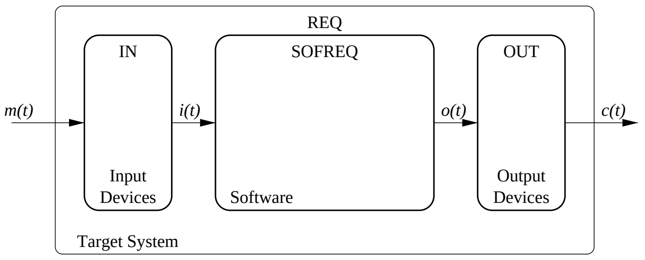

**Figure 1: System Design**

lows

$$\textbf{SOFREQ} \overset{\text{df}}{=} \left\{ \left(\underline{i}^t, \underline{o}^t\right) \mid \left(\forall \underline{m}^t, \underline{c}^t, \left(\text{IN}\left(\underline{m}^t, \underline{i}^t\right) \wedge \text{OUT}\left(\underline{o}^t, \underline{c}^t\right) \wedge \text{NAT}\left(\underline{m}^t, \underline{c}^t\right)\right) \Rightarrow \text{REQ}\left(\underline{m}^t, \underline{c}^t\right)\right)\right\} \quad (1)$$

Note that in many cases $\text{REQ}\left(\underline{m}^t, \underline{c}^t\right) \Rightarrow \text{NAT}\left(\underline{m}^t, \underline{c}^t\right)$. Further, any observed behaviour must be in **NAT**, since, by definition, behaviours not in **NAT** are not possible.

## 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. 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 would report the value of $\text{REQ}\left(\underline{m}^t, \underline{c}^t\right)$.

### 3.1 Using Monitors

A monitor can be used to check the behaviour of a target system either concurrently with the target system or post-facto, using a recording of the behaviour. In either case, the monitor should report if *all* behaviours exhibited by the target system are acceptable (i.e., in **REQ**). For a given behaviour $\left(\underline{m}^t, \underline{c}^t\right)$ on some interval $[t_i, t_f]$ and any $t_0 \in (t_i, t_f]$, the prefix behaviour, $\left(\hat{\underline{m}^t}, \hat{\underline{c}^t}\right)$, formed by considering $\left(\underline{m}^t, \underline{c}^t\right)$ on $[t_i, t_0]$ only, i.e.,

$$\left(\hat{\underline{m}^t}, \hat{\underline{c}^t}\right)(t) \overset{\text{df}}{=} \begin{cases} \left(\underline{m}^t, \underline{c}^t\right)(t) & \text{for } t \in [t_i, t_0] \\ \text{undefined} & \text{otherwise} \end{cases}$$

is also a behaviour of the system. Thus, if $\left(\hat{\underline{m}^t}, \hat{\underline{c}^t}\right) \notin$ **REQ** the system has behaved unacceptably and the monitor should report a failure.

This interpretation restricts these techniques to what [1] calls *safety properties*—if a behaviour is unacceptable then no extension of that behaviour is acceptable. Once a monitor has detected a failure, no further analysis of that behaviour will give a different result. In applications such as supervision[11], where continued analysis of the behaviour is needed following detection of a failure, the monitor, and presumably the target system, will need to be initialized (i.e., a new behaviour begins).

#### 3.1.1 Non-Testable Requirements

In [1], safety properties are distinguished from *liveness properties*—those requirements such that, for a given requirement and any finite duration behaviour, the behaviour can always be extended such that it satisfies the requirement. These include the common notions of liveness (the system must respond eventually) and fairness (if requested often enough eventually a given response will occur) as well as statistical properties on the behaviour (e.g., the average response time must be less than $T$). No monitor can determine that a target system does not satisfy such a requirement, since that can only be determined using infinite behaviours (e.g., a pending request could be serviced in the future). For real systems, however, liveness requirements are rarely strong enough to specify the true requirements, and should be converted into requirements that can be checked for finite duration behaviours (e.g., the system must respond to requests within a fixed time limit), which can be checked by a monitor.

### 3.2 Monitor Configuration

In this work, the monitor is assumed to consist of some software running on a computer system. The monitor software cannot, in general, observe the environmental state function, $\left(\underline{m}^t, \underline{c}^t\right)$, directly, but must do so through some input devices that communicate the values of the environmental quantities to input registers known as the *monitor software inputs.* For monitor software inputs, $s_1, s_2, \ldots, s_n$, of types $\mathbf{S_1}, \mathbf{S_2}, \ldots \mathbf{S_n}$, respectively, a *monitor input state function* is a function, $\underline{s}^t : \mathbf{Real} \rightarrow \mathbf{S_1} \times \mathbf{S_2} \times \ldots \mathbf{S_n}$, representing the value of the monitor software inputs for the periods of monitor operation. With respect to a particular monitor system, the set of all functions of type $\mathbf{Real} \rightarrow \mathbf{S_1} \times \mathbf{S_2} \times \ldots \mathbf{S_n}$ is denoted $\mathbf{S}$. The behaviour of the monitor input devices is characterized by the monitor input relation, $\mathbf{IN_{mon}} \subseteq (\mathbf{M} \times \mathbf{C}) \times \mathbf{S}$. An environmental state function–input state function pair is in the monitor input relation, $\left(\left(\underline{m}^t, \underline{c}^t\right), \underline{s}^t\right) \in \mathbf{IN_{mon}}$, if and only if $\underline{s}^t$ is a possible monitor input state function for the environmental state function represented by $\left(\underline{m}^t, \underline{c}^t\right)$. Since the monitor must observe all acceptable behaviours, it is required that $domain(\mathbf{IN_{mon}}) \supseteq \mathbf{REQ} \cap \mathbf{NAT}$.

The design of the monitor will determine, for each monitored or controlled quantity, whether it is observed independently of the target system (i.e., using different devices) or observed directly from the target system software. This results in two basic monitor configurations, in addition to the obvious mixtures of these approaches:

**Software Monitor** A *software monitor* is a monitor that directly observes the target system software input and output variables, i.e., $\underline{s}^t = \left(\underline{i}^t, \underline{o}^t\right)$, as illustrated in Figure 2. In this case $\mathbf{IN_{mon}}$ is related to **IN** and **OUT** as follows.

$$\mathbf{IN_{mon}} = \left\{ \left(\left(\underline{m}^t, \underline{c}^t\right), \left(\underline{i}^t, \underline{o}^t\right)\right) \middle| \text{IN}\left(\underline{m}^t, \underline{i}^t\right) \wedge \text{OUT}\left(\underline{o}^t, \underline{c}^t\right)\right\} \quad (2)$$

Software monitors include all of the monitor "architectures" discussed in [12].

**System Monitor** A *system monitor* is a monitor that ob-

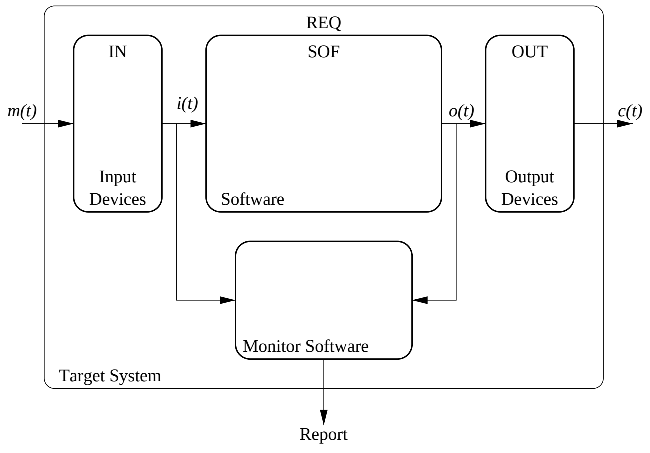

**Figure 2: Software Monitor**

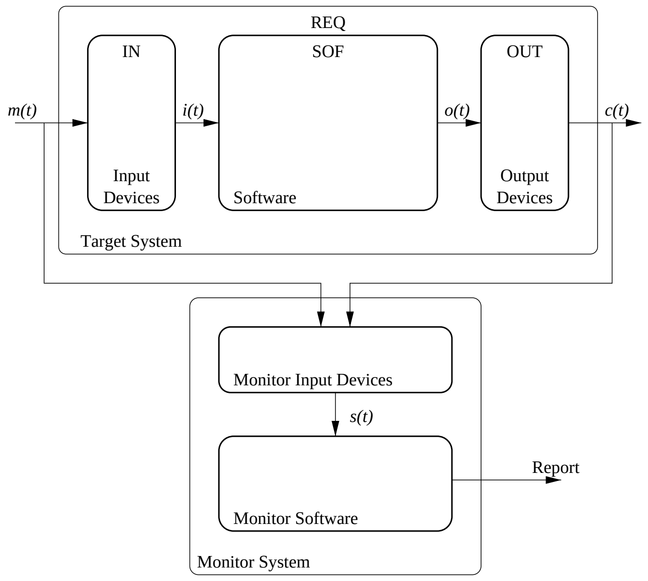

**Figure 3: System Monitor**

serves $\left(\underline{m}^t, \underline{c}^t\right)$ using its own input devices as illustrated in Figure 3.

The monitor software determines if the target system behaviour is consistent with **REQ** under the assumption that the monitor system's input devices are functioning correctly, as described in $\mathbf{IN_{mon}}$. The software must take into account the fact that $\mathbf{IN_{mon}}$ is usually non-functional, which we characterize by the two extreme approaches of a pessimistic or an optimistic monitor. A pessimistic monitor requires that *all* behaviours that could have resulted in a particular observation of the target system behaviour, $\underline{s}^t$, be in **REQ**, so the monitor software determines if $\underline{s}^t$ is in the pessimistic monitor relation, $\mathbf{MON_{pe}}$, which is defined as

$$\mathbf{MON_{pe}} \overset{\text{df}}{=} \Big\{ \underline{s}^t \in range(\mathbf{IN_{mon}}) \Big| \Big( \forall \left(\underline{m}^t, \underline{c}^t\right) \in \mathbf{M} \times \mathbf{C}, \left(\mathrm{IN_{mon}}\left(\left(\underline{m}^t, \underline{c}^t\right), \underline{s}^t\right) \wedge \mathrm{NAT}\left(\underline{m}^t, \underline{c}^t\right)\right) \Rightarrow \mathrm{REQ}\left(\underline{m}^t, \underline{c}^t\right)\Big)\Big\} \quad (3)$$

If $\mathrm{MON_{pe}}(\underline{s}^t)$ is $\underline{true}$ then the behaviour is certainly acceptable, i.e., $(\mathrm{MON_{pe}}(\underline{s}^t) \wedge \mathrm{IN_{mon}}\left(\left(\underline{m}^t, \underline{c}^t\right), \underline{s}^t\right)) \Rightarrow \mathrm{REQ}\left(\underline{m}^t, \underline{c}^t\right)$. A more optimistic view is to check if *any* behaviour that could have resulted in $\underline{s}^t$ is in **REQ**. The optimistic monitor relation, $\mathbf{MON_{op}}$, is defined as

$$\mathbf{MON_{op}} \overset{\text{df}}{=} \Big\{ \underline{s}^t \in range(\mathbf{IN_{mon}}) \Big| \Big( \exists \left(\underline{m}^t, \underline{c}^t\right) \in \mathbf{M} \times \mathbf{C}, \mathrm{IN_{mon}}\left(\left(\underline{m}^t, \underline{c}^t\right), \underline{s}^t\right) \wedge \mathrm{NAT}\left(\underline{m}^t, \underline{c}^t\right) \wedge \mathrm{REQ}\left(\underline{m}^t, \underline{c}^t\right)\Big)\Big\} \quad (4)$$

and includes those observations that may, but do not necessarily, represent acceptable behaviour. A monitor that evaluates $\mathbf{MON_{op}}$ will not give false negative results—reports that an acceptable behaviour is unacceptable—but is not appropriate for safety-critical systems since it may give false positive results—unacceptable behaviour reported as acceptable. The difference between $\mathbf{MON_{pe}}$ and $\mathbf{MON_{op}}$, or, more specifically their inverse image under $\mathbf{IN_{mon}}$, is indicative of the appropriateness of the monitor input devices as reflected in $\mathbf{IN_{mon}}$. A realistic monitor may combine these approaches, for example being pessimistic with regard to some quantities, and optimistic with regard to some others.

In the case of the software monitor configuration, and neglecting impossible behaviours (i.e., $\left(\underline{m}^t, \underline{c}^t\right) \notin \mathbf{NAT}$), $\mathbf{MON_{pe}} = \mathbf{SOFREQ}$—a software monitor determines if the target software is behaving in an acceptable manner.

From the above definitions we can see that $\mathrm{MON_{op}}(\underline{s}^t) \not\Rightarrow \mathrm{MON_{pe}}(\underline{s}^t)$—the pessimistic monitor will reject some behaviours that are accepted by the optimistic one. Also, if we assume that the input devices are working, i.e., for any observed monitor input state function, $\underline{s}^t$, there is a possible corresponding environmental state function: $\exists \left(\underline{m}^t, \underline{c}^t\right) \in \mathbf{M} \times \mathbf{C}, \left(\mathrm{IN_{mon}}\left(\left(\underline{m}^t, \underline{c}^t\right), \underline{s}^t\right) \wedge \mathrm{NAT}\left(\underline{m}^t, \underline{c}^t\right)\right)$, then the reverse implication holds, i.e., $\mathrm{MON_{pe}}(\underline{s}^t) \Rightarrow \mathrm{MON_{op}}(\underline{s}^t)$—if a behaviour is acceptable using the pessimistic approach, then it is acceptable using an optimistic approach. Both of these are consistent with our intuition.

In cases where $\mathbf{IN_{mon}}$ and $\mathbf{IN_{mon}}^{-1}$ are both functions (i.e., each $\left(\underline{m}^t, \underline{c}^t\right)$ maps to only one $\underline{s}^t$, which can be uniquely mapped back to $\left(\underline{m}^t, \underline{c}^t\right)$), and again assuming that any observed monitor input state function results from a possible environmental state function, $\mathbf{MON_{pe}} = \mathbf{MON_{op}}$. As discussed in Section 4, for real input devices and discrete time systems, $\mathbf{IN_{mon}}$ and $\mathbf{IN_{mon}}^{-1}$ are both non-functional relations in practice.

## 3.3 Accuracy

The accuracy of a monitor is determined by the set of possible false negatives, denoted **FN**, which is the intersection of **REQ** with the set of actual behaviours that the monitor may report as being unacceptable. The behaviours that may be reported as unacceptable are those in the image of

$\overline{\mathbf{MON_{pe}}}$ under $\mathbf{IN_{mon}}^{-1}$, as follows.

$$\mathbf{NEG} \overset{\mathrm{df}}{=} \Big\{ \left(\underline{m}^t, \underline{c}^t\right) \in \mathbf{M} \times \mathbf{C} \mid \left(\exists \underline{s}^t \in \mathbf{S}, \mathrm{IN_{mon}}\left(\left(\underline{m}^t, \underline{c}^t\right), \underline{s}^t\right) \wedge \neg \mathrm{MON_{pe}}(\underline{s}^t)\right)\Big\}$$
$$= \left\{ \left(\underline{m}^t, \underline{c}^t\right) \in \mathbf{M} \times \mathbf{C} \mid \left(\mathbf{IM}\left(\underline{m}^t, \underline{c}^t\right) \cap \overline{\mathbf{REQ}}\right) \neq \emptyset \right\} \tag{5}$$

where $\mathbf{IM}\left(\underline{m}^t, \underline{c}^t\right)$ is the image of $\left(\underline{m}^t, \underline{c}^t\right)$ under $\mathbf{IN_{mon}} \circ \mathbf{IN_{mon}}^{-1}$:

$$\mathbf{IM}\left(\underline{m}^t, \underline{c}^t\right) \overset{\mathrm{df}}{=} \Big\{ \left(\underline{\hat{m}}^t, \underline{\hat{c}}^t\right) \mid \left(\left(\underline{m}^t, \underline{c}^t\right), \left(\underline{\hat{m}}^t, \underline{\hat{c}}^t\right)\right) \in \left(\mathbf{IN_{mon}} \circ \mathbf{IN_{mon}}^{-1}\right)\Big\} \tag{6}$$

and thus, **FN** is

$$\mathbf{FN} \overset{\mathrm{df}}{=} \mathbf{REQ} \cap \mathbf{NEG}$$
$$= \Big\{ \left(\underline{m}^t, \underline{c}^t\right) \in \mathbf{M} \times \mathbf{C} \Big| \mathrm{REQ}\left(\underline{m}^t, \underline{c}^t\right) \wedge \left(\mathbf{IM}\left(\underline{m}^t, \underline{c}^t\right) \cap \overline{\mathbf{REQ}}\right) \neq \emptyset \Big\} \tag{7}$$

Consider **FN** under the best and worst case scenarios with respect to $\mathbf{IN_{mon}}$. In the best case $\mathbf{IN_{mon}}$ is the identity relation, perfectly relaying the values of $\left(\underline{m}^t, \underline{c}^t\right)$ to the monitor software. In this case, $\mathbf{IM}\left(\underline{m}^t, \underline{c}^t\right) = \left\{\left(\underline{m}^t, \underline{c}^t\right)\right\}$, $\mathbf{NEG} = \overline{\mathbf{REQ}}$ and $\mathbf{FN} = \emptyset$—the monitor software can detect exactly if the behaviour is acceptable or not. In the worst case $\mathbf{IN_{mon}}$ is a constant function, mapping all values of $\left(\underline{m}^t, \underline{c}^t\right)$ to the same value. In this case $\mathbf{IM}\left(\underline{m}^t, \underline{c}^t\right) = \mathit{domain}(\mathbf{IN_{mon}})$, $\mathbf{NEG} = \mathit{domain}(\mathbf{IN_{mon}})$ and $\mathbf{FN} = \mathbf{REQ}$—under no circumstances can the monitor be sure that the behaviour is acceptable. In this case the monitor will be *infeasible.*

DEFINITION 1. *A monitor is said to be* infeasible *with respect to a monitor input relation,* $\mathbf{IN_{mon}}$*, system requirements relation,* **REQ***, and environmental constraints,* **NAT***, if and only if* $\mathbf{MON_{pe}} = \emptyset$.

Note that infeasibility is an extreme case: it indicates that the monitor input devices are such that no behaviours will be accepted. The size of **FN** relative to the operational domain is a more precise measure of monitor usefulness.

Clearly if $\mathbf{IN_{mon}}$ is the identity relation, then $\mathbf{MON_{pe}} = \mathbf{REQ}$ and so, assuming a non-empty **REQ**, the monitor is feasible.

For an alternative view of accuracy, consider the set of false positives that may be reported by a monitor using the optimistic approach defined in Eq. (4), as follows.

$$\mathbf{POS} \overset{\mathrm{df}}{=} \Big\{ \left(\underline{m}^t, \underline{c}^t\right) \in \mathbf{M} \times \mathbf{C} \mid \left(\exists \underline{s}^t \in \mathbf{S}, \mathrm{IN_{mon}}\left(\left(\underline{m}^t, \underline{c}^t\right), \underline{s}^t\right) \wedge \mathrm{MON_{op}}(\underline{s}^t)\right)\Big\}$$
$$= \left\{ \left(\underline{m}^t, \underline{c}^t\right) \mid \left(\mathbf{IM}\left(\underline{m}^t, \underline{c}^t\right) \cap \mathbf{REQ}\right) \neq \emptyset \right\} \tag{8}$$

where **IM** is as defined in Eq. (6). The false positive set, **FP**, is thus

$$\mathbf{FP} \overset{\mathrm{df}}{=} \overline{\mathbf{REQ}} \cap \mathbf{POS}$$
$$= \Big\{ \left(\underline{m}^t, \underline{c}^t\right) \in \mathbf{M} \times \mathbf{C} \Big| \neg \mathrm{REQ}\left(\underline{m}^t, \underline{c}^t\right) \wedge \left(\mathbf{IM}\left(\underline{m}^t, \underline{c}^t\right) \cap \mathbf{REQ}\right) \neq \emptyset \Big\} \tag{9}$$

Considering the $\mathbf{IN_{mon}}$ scenarios from above, in the best case $\mathbf{FP} = \emptyset$ and in the worst case $\mathbf{FP} = \mathit{domain}(\mathbf{IN_{mon}})$—the monitor will report all observations as acceptable behaviour.

For realistic cases **FN** and **FP** will be non-empty and should be used during monitor system design to determine if the monitor is accurate enough for the particular application. This is discussed further in the next section.

## 4. PRACTICAL MONITORS

Practical monitors are likely to be implemented using either general- or special-purpose digital computers. This technology implies certain characteristics of the monitor input relation, and monitor behaviour, which influence the conclusions that can be drawn from the monitor output. This section discusses these characteristics, and states some conditions which must hold in order for the monitor to produce meaningful results.

### 4.1 Observation Errors

The choice of devices and/or software used by the monitor to observe the environmental quantities is a major design decision with respect to the monitor system. Design of a general mechanism for observing target system behaviour in a non-intrusive manner is beyond the scope of this work—readers interested in that topic are referred to [10] for a survey of the relevant literature. The following are some factors that should be taken into consideration in choosing monitor input devices.

Assuming that the monitor is a discrete-time system, there are two basic approaches to observing behaviour:

- Sample (i.e., observe the instantaneous value of) the relevant quantities at intervals.
- Modify the behaviour of the target system, and/or the systems that interact with it, to have them notify the monitor system of the values of relevant quantities ($\underline{m}^t, \underline{c}^t, \underline{i}^t$ or $\underline{o}^t$) as they read or change them. Such notification is assumed to include a *timestamp* indicating the time at which the reported value was observed by the target system.

#### 4.1.1 Discrete Time

Regardless of whether sampling or notification is used, time can only be measured at discrete points: if sampling is used then the sampling period determines the smallest relevant clock increment, whereas if notification is used it is determined by the precision of the notification timestamp. If we assume that using the notification approach the monitor receives notifications for all relevant changes, then this approach is not significantly different from the sampling approach in which the sampling period is the precision of the notification timestamp and uninteresting samples discarded. Thus, the results from sampling theory (e.g., see [8]) can be applied here to show that, for infinite duration signals

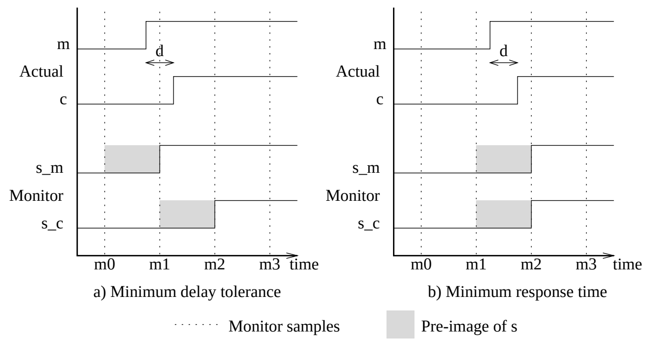

**Figure 4: Time Accuracy**

(behaviours), it is sufficient to sample at twice the maximum frequency of change in the environmental quantities. However, the monitor is typically concerned with what has happened between the most recent two samples, and so the discrete clock will introduce some error in the perceived time of events, which is referred to as the *time error.* For real-time systems, errors in measuring time are particularly important.

Consider the behaviours illustrated in Figure 4, in which the values of $m$ and $c$ represent that a condition of, respectively, a monitored and controlled quantity is either $\underline{false}$ (low) or $\underline{true}$ (high). Similarly, the values of $s\_m$ and $s\_c$ represent the values as they appear to the monitor software and the shaded regions represent the image of these changes under $\mathbf{IN_{mon}}^{-1}$. Let $\delta_{mon}$ represent the monitor sampling interval (i.e., $m_i - m_{i-1}$) and $d$ be the elapsed time between the change in $m$ and $c$, as illustrated. Assuming that the change in $c$ is a correct target system response to the change in $m$, consider the two cases illustrated.

**a)** The monitor sees distinct changes. The monitor can determine only that $0 < d < 2\delta_{mon}$. This behaviour will be rejected (considered unacceptable) if the specified maximum delay for that change is less than $2\delta_{mon}$. This results in Condition 1, below.

CONDITION 1. *The maximum time error introduced by the monitor input devices must be less than* $\frac{1}{2}min(\mathbf{Delay})$*, where* **Delay** *is the set of maximum delay tolerances for the dependent*[2] *quantities given in the SRD.*

**b)** The monitor sees simultaneous changes. Here the monitor can determine that $-\delta_{mon} < d < \delta_{mon}$ (i.e., $c$ could change before $m$); hence this behaviour will be rejected if $c$ is only permitted to change following $m$. The implication is that $\delta_{mon}$ must be less than the minimum response time of the target system. This constraint can be weakened, however, by noting that, in order for the target system to have responded to the change in $m$, it must have observed its value between the changes in $m$ and $c$, so this case can be avoided by ensuring that the monitor samples in that interval as well. Thus we have Condition 2, below, which can be satisfied by ensuring that sampling by the target and monitor systems is synchronized to within the minimum target system response time. If event notification from the target system is used, the monitor and target systems are assured to be synchronized.

[2]A quantity $c$ is dependent on $m$ if the value of $c$ may be required to change as a result of a change in the value of $m$.

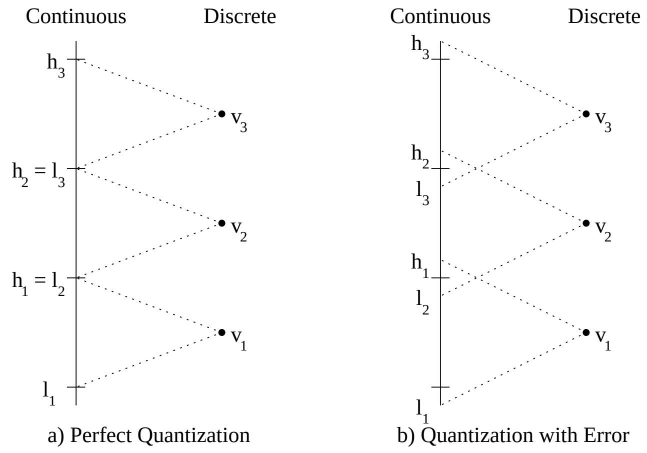

**Figure 5: Quantization and Error**

CONDITION 2. *The maximum difference between the time error in the target system and the time error in the monitor system for the same event must be less than the minimum time in which the target system might respond to that event.*

A monitor system that does not satisfy Condition 1 will be infeasible. A system that does not satisfy Condition 2 may give false negative results for target systems responding too quickly.

### *4.1.2 Quantization and Measurement Error*

As with time, other values observed by the monitor software must be of finite precision, so **Real** valued environmental quantities must be quantized, such that, for example, discrete value $v_i$ represents all continuous values, $x$, such that $l_i < x \leq h_i$. Whereas time is continuously increasing, so we know something about the error, other quantities do not necessarily have this property. As illustrated in Figure 5, if the quantization is perfect, i.e., $h_i = l_{i+1}$, the worst case error is half the quantization step size, $h_i - l_i$, and no non-determinism is introduced. Practical devices will exhibit some measurement error in addition to quantization, so the actual error will be larger, and $\mathbf{IN_{mon}}$ will be non-functional.

For a monitor to be feasible, there must be some monitor input state functions, $\underline{s}^t$, for which all images under $\mathbf{IN_{mon}}^{-1}$ are acceptable. Because of the variety of ways that quantities may be used in the SRD, we cannot state generally applicable conditions on $\mathbf{IN_{mon}}$ that will ensure that a monitor is feasible. Condition 3 is a necessary, but not sufficient condition for feasibility.

CONDITION 3. *The maximum error in observing a particular controlled quantity must be less than the difference between the maximum and minimum values of that quantity permitted by* **REQ**.

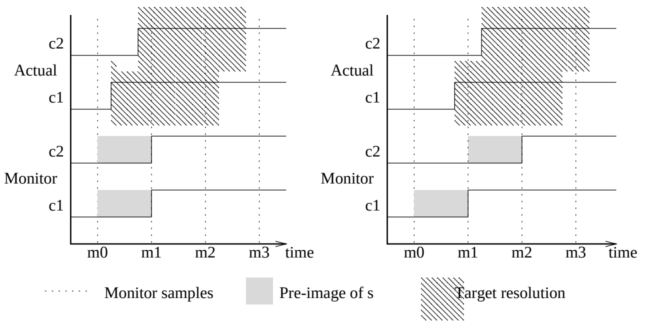

**Figure 6: Event Resolution**

## 4.2 Non-determinism

As mentioned in Section 2.1, practical requirements documents will be non-functional to allow for unpredictable delays or errors in calculation or measurement. In particular, if the target system is to be implemented using a discrete-time system, then, for some small time, $r$, **REQ** must allow events that occur within $r$ of each other to be treated as either a single event (i.e., simultaneous) or distinct events (i.e., non-simultaneous). The time $r$ is known as the *time resolution* for the target system.[3] The monitor system must take this non-determinism into account when evaluating behaviour.

Consider the behaviour illustrated in Figure 6, and the target time resolution as indicated. The requirements must allow the changes in C1 and C2 to be treated as either simultaneous or not in both cases illustrated. Assuming that the monitor system samples at the indicated times, it will observe the changes either simultaneously or not, but can certainly tell that they occurred within $2\delta_{mon}$ of each other. If $\delta_{mon}$ is less than half the time resolution required for the target, which is required to satisfy Condition 1, then in both cases all images of $\underline{s}^t$ under $\mathbf{IN_{mon}}^{-1}$ allow the changes to be interpreted as happening in either order or simultaneously, so $\mathbf{MON_{pe}}$ accepts a behaviour in which the target system interprets them in either way. The monitor software must take this non-determinism into account.

In the case of the software monitor configuration, as illustrated in Figure 2, the monitor software and the target system software are assured to see the same values (i.e., $\underline{s}^t = (\underline{i}^t, \underline{o}^t)$), so the monitor implementation can require deterministic behaviour.

## 4.3 Response Time

Clearly the delay introduced by the monitor input devices will impose a lower limit on the monitor response time—the maximum time between a failure occurring and the monitor reporting it—since a monitor cannot report a failure before it is evident in $\underline{s}^t$. The choice of input devices can also affect the amount of processing required by the monitor software, which will also affect response time, although less predictably so. For example, input devices may be available that can directly report the value of relevant conditions (e.g., sensors to detect if a robot has touched a wall) whereas a different choice of input devices would require that the monitor software perform some, possibly expensive, calculations (e.g., search a list of wall locations to determine if the robot is touching any).

[3]Readers are referred to [7] for a further discussion of time resolution.

## 4.4 Computational Resources

Using any notation that is expressive enough to describe realistic target system requirements, it is certainly possible to express requirements such that $\mathrm{MON_{pe}}(\underline{s}^t)$ is either not computable, or is computable only using an impractical amount of computational resources. Some possible causes of this are:

- $\mathrm{REQ}\left(\underline{m}^t, \underline{c}^t\right)$ or $\mathrm{NAT}\left(\underline{m}^t, \underline{c}^t\right)$ may not be practically computable. As in [6], this may result from specification errors such as infinite recursions in function or predicate definitions, or from computation of $\mathrm{MON_{pe}}(\underline{s}^t)$ requiring quantification over large sets. Specification authors must take care to avoid these situations, if possible.
- $\mathrm{IN_{mon}}$ may be such that the pre-image of $\underline{s}^t$ is not easily computed. Since real-valued monitored and controlled quantities are permitted, the pre-image of $\underline{s}^t$ will often be infinite, but, for most practical input devices, will be easily described by simple predicates, characterizing a range of possible values, for example. If this is not the case, however, it may be impractical to determine if all elements of the pre-image are acceptable.

Careful review of the SRD and judicious choice of monitor input devices may help to avoid these situations.

## 5. CONCLUSIONS

This paper presents a precise definition of a monitor for a real-time system, and identifies some necessary conditions for a monitor to be feasible and useful. These conditions can be used to help determine if a particular monitor design is sufficient for the target system.

Monitors, such as described in this work, are well suited to automated testing of systems, where they function as an oracle, reporting if the behaviour is acceptable or not. This application offers significant improvement over non-automated testing since test cases can be evaluated quickly and errors in behaviour are quickly and reliably detected.

In a similar way, monitors can be used as supervisors to observe the behaviour of the target system in operation and report failures as they occur. Such a supervisor could be used as a redundant safety system to initiate corrective or preventative action when a failure is detected.

## 5.1 Future Work

We have validated this work using a few software monitors that were automatically generated from system requirements documentation.[7] Further study, using different target systems and using the system monitor configuration discussed in Section 3.2 would undoubtedly lead to new insight.

Further work is also needed to enhance techniques for specifying the behaviour of input and output devices, and to develop analysis techniques that will permit designers to easily determine if a particular set of monitor input devices is sufficient for the monitoring task at hand.

## Acknowledgements

Many friends and colleagues at McMaster University have helped to improve this work through informal discussions and helpful comments offered on earlier versions. In particular Drs. Ryszard Janicki, Jan Madey, Martin von Mohrenschildt, Emil Sekerinski and Jeffery Zucker have each offered helpful and constructive comments. Comments from the anonymous referees have also helped to improve this paper.

Constance Heitmeyer and her group at the US Naval Research Laboratory, Center for High Assurance Computer Systems, were helpful in the initial formulation of the problem statement.

The financial support received from the Natural Sciences and Engineering Research Council (NSERC), Communications and Information Technology Ontario, (CITO), the Telecommunications Research Institute of Ontario (TRIO), McMaster University and Memorial University of Newfoundland is gratefully acknowledged.

## 6. REFERENCES

[1] B. Alpern and F. B. Schneider. Defining liveness. *Inf. Process. Lett.*, 21:181–185, Oct. 1985.

[2] C. L. Heitmeyer, A. Bull, C. Gasarch, and B. G. Labaw. SCR*: A toolset for specifying and analyzing requirements. In *Proc. Conf. Computer Assurance (COMPASS)*, pages 109–122, Gaithersburg, MD, June 1995. National Institute of Standards and Technology.

[3] K. L. Heninger. Specifying software requirements for complex systems: New techniques and their application. *IEEE Trans. Softw. Eng.*, SE-6(1):2–13, Jan. 1980.

[4] K. L. Heninger, D. L. Parnas, J. E. Shore, and J. Kallander. Software requirements for the A-7E aircraft. Technical Report MR 3876, Naval Research Laboratory, 1978.

[5] D. L. Parnas and J. Madey. Functional documentation for computer systems. *Science of Computer Programming*, 25(1):41–61, Oct. 1995.

[6] D. K. Peters. Generating a test oracle from program documentation. M. Eng. thesis, McMaster University, Dept. of Electrical and Computer Engineering, Hamilton, ON, Apr. 1995.

[7] D. K. Peters. *Deriving Real-Time Monitors from System Requirements Documentation.* PhD thesis, McMaster University, Hamilton ON, Jan. 2000.

[8] J. G. Proakis and D. G. Manolakis. *Digital Signal Processing Principles, Algorithms and Applications.* Maxwell Macmillan, second edition, 1992.

[9] A. P. Ravn, H. Rischel, and K. M. Hansen. Specifying and verifying requirements of real-time systems. *IEEE Trans. Softw. Eng.*, 19(1):41–55, Jan. 1993.

[10] U. Schmid. Monitoring distributed real-time systems. *Real-Time Systems*, 7(1):33–56, July 1994.

[11] D. Simser and R. E. Seviora. Supervision of real-time systems using optimistic path prediction and rollbacks. In *Proc. Int'l Symp. Software Reliability Eng. (ISSRE)*, pages 340–349, Oct. 1996.

[12] J. J. Tsai, Y. Bi, S. J. H. Yang, and R. A. W. Smith, editors. *Distributed Real-Time Systems : Monitoring, Visualization, Debugging, and Analysis.* John Wiley & Sons, 1996.

[13] J. J. Tsai and S. J. Yang, editors. *Monitoring and Debugging of Distributed Real-Time Systems.* IEEE Computer Society Press, 1995.

[14] 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. also printed as CRL Report No. 242, Telecommunications Research Institute of Ontario (TRIO).

[15] A. J. van Schouwen, D. L. Parnas, and J. Madey. Documentation of requirements for computer systems. In *Proc. Int'l Symp. Requirements Eng. (RE '93)*, pages 198–207. IEEE, Jan. 1993.

[16] E. J. Weyuker. On testing non-testable programs. *The Computer Journal*, 25(4):465–470, 1982.

[17] P. Zave and M. Jackson. Four dark corners of requirements engineering. *ACM Trans. Software Eng. and Methodology*, 6(1):1–30, Jan. 1997.