In Proceedings of RTSS'91: The 12th IEEE Real-time Systems Symposium, San Antonio, TX, pp. 244-253, Dec 1991.

Specification and Verification of Real-time Embedded Systems using Time-constrained Reactive Automata

Azer Bestavros
Department of Computer Science
Boston University
Boston, MA 02215

Abstract

The vital role that real-time embedded systems are playing and will continue to play in our world, coupled with their increasingly complex and critical nature, demand a rigorous and systematic treatment that recognizes their unique requirements. The Time-constrained Reactive Automaton (TRA) is a formal model of computation that admits these requirements. Among its salient features is a fundamental notion of space and time that restricts the expressiveness of the model in a way that allows the specification of only reactive, spontaneous, and causal computations. Using the TRA formalism, there is no conceptual distinction between a *system* and a *property*; both are specified as formal objects. This reduces the verification process to that of establishing correspondences – namely *preservation* and *implementation* relationships – between such objects. In this paper, we present the TRA model and briefly overview our experience in using it in the specification and verification of real-time embedded systems.

1 Introduction

A computing system is *embedded* if it is explicitly viewed as being a component of a larger system whose primary purpose is to monitor and control an environment. The leaping advances in computing technologies that the last few decades have witnessed has resulted in an explosion in the extent and variety of such systems. This trend is only likely to continue in the future.

Embedded systems are usually associated with critical applications, in which human lives or expensive machinery are at stake. Their missions are often long-lived and uninterruptible, making maintainance or reconfiguration difficult. Examples include command and control systems, nuclear reactors, process-control plants, robotics, avionics, switching circuits and telephony, data-acquisition systems, and real-time databases, just to name a few. The sustained demands of the external environments in which such systems operate pose relatively rigid and urgent requirements on their performance. These requirements are usually stated as constraints on the real-time behavior of these systems. Wirth [33] singled out this processing-time dependency as the one aspect that differentiates embedded systems from other sequential and parallel systems. This led to a body of research on *real-time computing*, which encompasses issues of specification techniques, analysis and validation, formal verification, programming languages, development tools, scheduling, and operating systems [31]. However, the absence of a unified suitable formal framework that addresses the afforementioned issues severely limited the usefulness of these studies [14].

Previous studies in modeling real-time systems have focussed on adding the notion of time to the formal modeling techniques of traditional systems, namely logic-based [26, 10, 1, 16], Petri-net-based [27, 24, 11, 17], state-based [12, 32, 18, 2, 5], and process-algebra-based [28, 13, 3]. In all of these studies very little emphasis, if any, was put on the physical nature of the modeled systems. Issues of spontaneity, causality, spacial locality, and reactivity are often disregarded, thus making real-time computing research physically unrealistic.

In [8], we proposed the Time-constrained Reactive Automata (TRA) model as a real-time computing formalism that recognizes the physical aspects of embedded systems. In that respect, the TRA model proved to be instrumental in the specification, verification, and simulation of asynchronous circuit designs, reactive control systems [9], and behavioral planning [7]. In this paper, we briefly overview the TRA model and its use in the specification and verification of real-time embedded systems.

2 The TRA Model

The TRA model has evolved from our earlier work in [5] extending Lynch's IOA model [21, 20] to suit embedded and time-constrained computation.

2.1 Novelties

The TRA model differs from others in that it does not allow the specification of systems that are not *reactive*. A system is reactive if it cannot block the occurence of events not under its control. A sufficient condition for reactiveness is the *input enabling* property proposed in [21]. The TRA model is input enabled. It distinguishes clearly between environment-controlled actions, which cannot be restricted or constrained, and locally-controlled actions, which can be scheduled and disabled. Communication is asynchronous and non-blocking.

An important aspect of the TRA model is its notion of space in relation to time. In particular, events occur at uniquely identifiable points in time and space. Events occuring at the same time and place are undistinguishable. The TRA model admits the causal nature of physical processes. A non-deterministic system is *causal* if given two inputs that are identical up to any given point in time, there exist outputs (for the respective inputs) that are also identical up to the same point in time. The TRA model enforces causality by requiring that any locally-controlled actions be produced only as a *result* of an earlier *trigger*. *Spontaneity* is a notion closely related to causality.[1] A system is *spontaneous* if its output actions at any given point in time t cannot depend on actions occuring at or after time t. In particular, if an output occurs simultaneously with (say) an input transition, the same output could have been produced without the simultaneous input transition [30]. Simultaneity is, thus, a mere coincidence; the output event could have occured spontaneously even if the input transition was delayed. The TRA model enforces spontaneity by requiring that simultaneously occuring events be independent; time has to *necessarily* advance to observe dependencies.

The TRA model distinguishes between two notions of time, namely *real* and *perceived*. Real time cannot be measured by any single process in a given system; it is only observable by the environment. Perceived time, on the other hand, can be specified using uncertain real time delays. The TRA model, therefore, does not provide for (or allow the specification of) any *global* or *perfect clocks*. As a consequence, the only measure of time available for system processes has to be relative to *imperfect, local clocks*. This distinction between real time and perceived time is important when dealing with embedded applications where specifications are usually given with respect to real time, but have to be implemented relying on perceived time.

[1] Actually both spontaneity and causality are directly related to the past and future light cones of an event in space-time [15].

2.2 Basic definitions

We adopt a continuous model of time similar to that used in [2, 19]. We represent any point in time by a nonnegative real $t \in \Re$. Time intervals are defined by specifying their end-points which are drawn from the set of nonnegative rationals $\mathcal{Q} \subset \Re$. A time interval is viewed as a traditional set over nonnegative real numbers. It can be an empty set, in which case it is denoted by ε, it can be a singleton set, in which case it is denoted by the $[t, t]$, $t \in \mathcal{Q}$, or else it can be an infinite set, in which case it is denoted by $[t_l, t_u]$, $(t_l, t_u]$, $[t_l, t_u)$, or (t_l, t_u) – the right-closed, left-closed, and open time intervals, respectively, where $t_l, t_u \in \mathcal{Q}$ and $t_l < t_u$. The set of all such infinite time intervals is denoted by $\mathcal{D}$.

A real-time system is viewed as a set of interacting mealy automata called TRAs. TRAs communicate with each other through *channels*. A channel is an abstraction for an *ideal* unidirectional communication. The information that a channel carries is called a *signal*, which consists of a sequence of *events*. An event underscores the occurence of an *action* at a specific point in time. An action is a *value* associated with a channel. For example, let `North`, `South`, `East`, and `West` be the possible values that can be signaled on some channel `MOVE` of a given TRA. `MOVE(East)` is, therefore, a possible action of the TRA. The instantiation of `MOVE(East)` at time t_1 denotes the occurence of an event $\langle \texttt{MOVE(East)} : t_1 \rangle$. The sequence of events $\langle \texttt{MOVE(East)} : t_1 \rangle \langle \texttt{MOVE(North)} : t_2 \rangle \langle \texttt{MOVE(South)} : t_3 \rangle$ ... *etc.* constitutes a signal. Signals are single valued; they cannot convey more than one event simultaneously. That is, for a signal $\langle a_0 : t_0 \rangle \langle a_1 : t_1 \rangle \ldots \langle a_k : t_k \rangle \ldots$ we require that $t_k < t_{k+1}, k \geq 0$.

At any point in time, a TRA is in a given *state*. The set of all such possible states defines the TRA's *state space*. The state of a TRA is visible and can only be changed by local *computations*. Computations (and thus state transitions) are triggered by actions and might be required to meet specific timing constraints.

2.3 TRA Objects

A TRA object is a sextuple $(\Sigma, \sigma_0, \Pi, \Theta, \Lambda, \Upsilon)$, where

- Σ, the TRA signature, is the set of all the channels of the TRA. It is partitioned into three disjoint sets of input, output, and internal channels. We denote these by Σ_{in}, Σ_{out}, and Σ_{int}, respectively. The set consisting of both input and output channels is the set of external channels (Σ_{ext}). These are the only channels visible from outside the TRA. The set consisting of both output and internal channels is the set of local channels (Σ_{loc}). These are the locally controlled channels of the TRA.

- $\sigma_0 \in \Sigma_{\text{in}}$ is the start channel.

- Π, the signaling range function, maps each channel in Σ to a possibly infinite set of actions that can be signaled on that channel. Action sets of different channels are disjoint. The function Π naturally generalizes to sets of channels in the following manner: $\Pi(\Sigma_i) = \bigcup_j \Pi(\sigma_{ij})$, where $\sigma_{ij} \in \Sigma_i$. In particular, the set of all the TRA actions is given by: $\Pi(\Sigma)$. The set of input, output, internal, external, and local actions are similarily given by $\Pi(\Sigma_{\text{in}})$, $\Pi(\Sigma_{\text{out}})$, $\Pi(\Sigma_{\text{int}})$, $\Pi(\Sigma_{\text{ext}})$, and $\Pi(\Sigma_{\text{loc}})$, respectively.

- Θ is a possibly infinite set of states of the TRA.

- $\Lambda \subseteq \Theta \times \Pi(\Sigma) \times \Theta$ is a set of possible computational steps of the TRA. TRAs are input enabled which means that for every $\pi \in \Pi(\Sigma_{\mathtt{in}})$, and for every $\theta \in \Theta$, there exists at least one step $(\theta, \pi, \theta') \in \Lambda$, for some $\theta' \in \Theta$.

- $\Upsilon \subseteq \Sigma \times \Sigma_{\text{loc}} \times \mathcal{D} \times 2^{\Theta}$ is a set of time constraints on the operation of the TRA. A time constraint $\upsilon_i \in \Upsilon$ is a quadruple $(\sigma_i, \sigma_i', \delta_i, \Theta_i)$ whose interpretation is that: if an action is signaled at time $t \in \Re$ on the channel σ_i, then an action should be fired on the channel σ_i' at time t', where $t' - t \in \delta_i$, provided that the TRA does not enter any of the states in Θ_i for the open interval (t, t'). The channel $\sigma_i \in \Sigma$ is called the *trigger* of the time constraint, whereas $\sigma_i' \in \Sigma_{\text{loc}}$ is called the *constrained* channel. $\Theta_i \subseteq \Theta$ defines the set of states that disable the time constraint; once triggered a time constraint becomes and remains *active* until *satisfied* or *disabled.* A time constraint is satisfied by the firing of an action on the channel σ_i within the imposed time bounds; it is disabled if the TRA enters in one of the disabling states in Θ_i before it is satisfied. The interval δ_i specifies upper and lower bounds on the delay between the triggering and satisfaction (or disabling) of the time constraint υ_i.

As an example of a TRA specification, consider the the up/down counter whose state diagram is shown in Figure-1. The counter accepts commands issued on the input channel `cmd` to count up or down and signals the value of the current count on the output channel `cnt`. The counter starts executing once an action is fired on the `init` channel. The value of the `init` signal determines the starting state of the counter. Thereafter, the counter is constrained to produce a count every at least 1.9 and at most 2.1 time units. Figure-2 shows the TRA specification of such a counter.

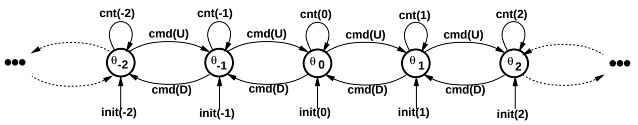

Figure 1: State diagram of up/down counter.

$\bullet \Sigma = \Sigma_{\text{in}} \cup \Sigma_{\text{out}} \cup \Sigma_{\text{int}}$, where:
$\Sigma_{\text{in}} = \{\mathtt{cmd}, \mathtt{init}\}$, $\Sigma_{\text{out}} = \{\mathtt{cnt}\}$, and $\Sigma_{\text{int}} = \phi$.
$\bullet$`init` $\in \Sigma_{\text{in}}$ is the start channel.
$\bullet \Pi$, the signaling range function, is defined as follows:
$\Pi(\mathtt{init}) = \mathcal{Z}$, $\Pi(\mathtt{cmd}) = \{\mathtt{Up}, \mathtt{Down}\}$, and $\Pi(\mathtt{cnt}) = \mathcal{Z}$.
$\bullet \Theta$, the set of states is given by: $\{\theta_i : i \in \mathcal{Z}\}$.
$\bullet \Lambda$, the set of computational steps is given by:

$$\begin{aligned}\Lambda = (&\textstyle\bigcup_{i,j \in \mathcal{Z}} \{(\theta_i, \mathtt{init}(j), \theta_j)\}) \cup \\ &(\textstyle\bigcup_{i \in \mathcal{Z}} \{(\theta_i, \mathtt{cmd(UP)}, \theta_{i+1})\}) \cup \\ &(\textstyle\bigcup_{i \in \mathcal{Z}} \{(\theta_i, \mathtt{cmd(Down)}, \theta_{i-1})\}) \cup \\ &(\textstyle\bigcup_{i \in \mathcal{Z}} \{(\theta_i, \mathtt{cnt}(i), \theta_i)\}).\end{aligned}$$

$\bullet \Upsilon$, the set of time constraints is given by:
$\Upsilon = \{(\mathtt{init}, \mathtt{cnt}, [1.9, 2.1], \phi), (\mathtt{cnt}, \mathtt{cnt}, [1.9, 2.1], \phi)\}$.

Figure 2: TRA-specification of up/down counter.

2.4 Space and Time aspects of TRAs

The behavior of a TRA is generally non-deterministic. Three sources of non-determinism can be singled out. In a given state there might be a number of choices concerning the action to be fired. Each one of these choices results in a different computational step, and thus in a different execution. This gives rise to *control* non-determinism. The TRA timing constraints specify lower and upper bounds on the delay between causes and effects, thus leaving the TRA with a potentially infinite number of choices concerning the exact delay to be exhibited. Each one of these choices results in a different event, and thus in a different execution. This gives rise to *timing* nondeterminism. Finally, the computation associated with specific actions might be non-deterministic. In this case, firing the same action from the same state might result in different next states, and thus in different executions. This gives rise to *computation* non-determinism. Considered separately, each one of the above forms of non-determinism is benign. A combination thereof, however, deserves a closer attention. In particular, the interplay between control non-determinism and timing non-determinism is interesting because it is related to the notions of space and time. Control non-determinism refers to uncertainties about the identity of the channel that will be fired; it refers to a *spacial uncertainty.* As such, and to abide by the spontaneity principle, it must reduce the range of possible *timing uncertainty.*

Two actions of a TRA *conflict* if the occurence of one of them *enables*, *disables*, or *affects* the occurence of the other. We extend our notion of conflict to channels as follows. Two channels σ_1 and σ_2 conflict if at least one action from $\Pi(\sigma_1)$ and one action from $\Pi(\sigma_2)$ conflict. The formal definitions for "enables", "disables", and "affects" were given in [8]. Obviously, each one of these three relationships depicts one form of computational dependency that emerges due to sharing information about state. For two local actions to conflict, their respective channels must be under the control of a single *component* of the TRA. The conflict relationship, therefore, defines a partition on the locally-controlled channels of a given TRA. Two local channels σ_1 and σ_2 belong to the same component (class) if they conflict.

The partition of the TRA's locally-controlled channels into classes captures some of the structure of the system the automaton is modeling or the set of requirements it is specifying. In particular, each class of channels is intended to represent the set of channels locally-controlled by *some* system component. This partitioning retains the basic control structure of the system's primitive components and provides a concrete notion of spacial locality.

The actions on the input channels of a given TRA are not under its control; they can fire at any time. To preserve the non-blocking (input-enabled) nature of the TRA model, it is, therefore, necessary to insure that input actions on different channels do not conflict. A TRA $\mathcal{A}$ is improper if at least two of its input channels conflict, otherwise it is proper. For the remainder of this paper, it will be assumed that any TRA is *proper* unless otherwise stated.

The notion of system components we are presenting here is novel and entirely different from that used in untimed models to express fairness [21] by requiring that, in an infinite execution, each of the system's components gets infinitely many chances to perform its locally-controlled actions. In timed systems, the major concern is *safe* and not necessarily *fair* executions [29]. Even if required, fairness can be enforced by treating it as a safety property; liveness properties can be handled in infinite execution by requiring time to grow unboundedly.[2]. This led to the abandoning of the idea of partitioning a system into components in our earlier model proposed in [5]. Lynch and Vaandrager [22] followed suit in their recent modification of the model proposed in [32]. In the TRA model we use system components to represent what can be termed as *spacial locality.* Different actions can be signaled at the same "time" only if they are not signaled from the same "place"; they can be produced at the same "place" only if they do not occur at the same "time". This intuition is inspired from physical systems, where events are characterized and distinguishable by their time-space coordinates [15].

2.5 TRA Executions and Behaviors

In standard automata theory, there is no distinction between choosing a transition and firing it; both of them occur instantaneously. In the TRA model, a distinction is made whereby choosing (scheduling) a transition and executing (commiting) that transition are not necessarily instantaneous activities. They are "distinct" in that they may be separated in time. As a matter of fact, a scheduled transition does not necessarily have to be committed; it can be abandoned due to unforeseeable conditions. The distinction between the two activities is also pronounced in the way the TRA model differentiates between input and local events. Input events are uncontrollable; they are not scheduled. Local events are.

The *state* of a TRA at an arbitrary point in time is not sufficient to construct its *future behavior.* To explain why this is true consider the example shown in Figure-3, where a TRA is known to be in some state s at time t_1. Assume that, due to a triggering event at some earlier time t_0, an action is scheduled to fire at some point in a future interval given by $[t_0 + t_{\text{lo}}, t_0 + t_{\text{hi}}]$. Knowing only the state of the TRA at time t_1 is obviously not sufficient to predict future behaviors. In addition to the state, the intervals of time where scheduled transitions might fire have to be recorded. We encapsulate this knowledge in our notion of *intentions.*

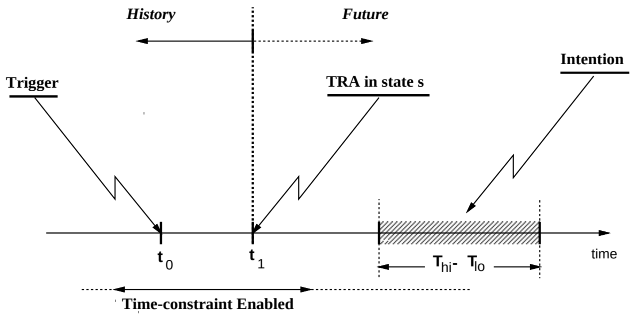

Figure 3: The notion of a TRA status.

Consider the time constraint $v_i = (\sigma_i, \sigma_i', \delta_i, \Theta_i) \in \Upsilon$. As we explained before, v_i identifies a time-constrained causal relationship between the events signaled on σ_i and those signaled on σ_i'. In particular, the occurence of a triggering event on σ_i results in an intention to perform an action on σ_i' within the time frame imposed by δ_i. The commitment or abandonment of such an intention in due time is conditional on the states assumed by the TRA from when the intention is posted until when it is committed or abandoned. At any given point in time, a TRA might have several outstanding intentions.

For a given TRA, we define the *intention vector* $I = \vec{\Delta}$ to be a vector of r sets of intentions, where $r = |\Upsilon|$. Each entry in I is associated with one of the TRA's time constraints. In particular, if $v_i = (\sigma_i, \sigma_i', \delta_i, \Theta_i) \in \Upsilon$ is one of the TRA's time constraints, then $I[v_i] = \{\delta_{i1}, \delta_{i2}, \ldots, \delta_{ik}, \ldots \delta_{im}\}$ denotes a set of m time intervals during which actions on the channel σ_i' are intended to be fired as a result of earlier triggers on σ_i. Each one of the intervals in Δ_i can be thought of as an independent *activation* of the time constraint v_i. An empty intentions set, $I[v_i] = \phi$, indicates the absence of any activations of v_i. The empty intention vector, I_ϕ, consists of r such empty sets.

At any point in time, the intention vector of a TRA can be thought of as an extension of the TRA's state.

[2] Such executions were called *admissible* in [22]

We define the status of a TRA at any point in time $t \in \Re$ to be the tuple (θ, I), where θ and I are the TRA's state and intention vector at time t, respectively.

A TRA changes its status only as a response to the occurence of an input or an intended local event. In other words, the change in a TRA's status is necessarily a causal *reaction* to an input event or to an earlier triggering event. Assume that the status (θ, I) of a TRA was entered at time t as a result of an event $\langle \pi : t \rangle$, where $\pi \in \Pi(\sigma_j), \sigma'_j \in \Sigma$. Furthermore, assume that at time t' ($t' \geq t$), an action $\pi' \in \Pi(\sigma'_j)$ is fired, where $\sigma'_j \in \Sigma$. As a result, the TRA will assume a new status (θ', I'). The status (θ', I') is called a *successor* of the status (θ, I) due to the event $\langle \pi' : t' \rangle$. Five conditions – namely, legality, spontaneity, safety, causality, and consistency – have to be met for such a succession to occur. These conditions are formally defined in [8].

The *legality* condition insures that the computational step that changes the state of the TRA from θ to θ' as a result of action π' is defined. The *spontaneity* condition allows the occurence of simultaneous events only if they are independent; time has to ellapse for dependencies to be manifested. In particular, two simultaneous input events can occur only if they are on different channels, two simultaneous local events can occur only if they are fired on channels belonging to different components, and a local event cannot depend on an input event fired simultaneously. The *safety* condition guarantees that no active time constraints expire. In other words, outstanding intentions are either committed or abandoned in due time. The *causality* condition necessitates that local events be causal; they are signaled only if intended due to an earlier trigger. The *consistency* condition requires that the intentions in I continue to exist in I' unless otherwise dictated by the occurence of the event $\langle \pi' : t' \rangle$.

We use the notation $(\theta, I) \overset{\langle \pi' : t' \rangle}{\longmapsto} (\theta', I')$ to denote the *direct status succession* from (θ, I) to (θ', I') due to the firing of the event $\langle \pi' : t' \rangle$. Furthermore, we use the notation $(\theta, I) \overset{\alpha}{\longmapsto} (\theta', I')$ to denote the *extended status succession* from (θ, I) to (θ', I') due to the firing of the sequence of events α.

A TRA is said to have reached a *stable status* $(\hat{\theta}, \hat{I})$, if all entries of the intention vector are empty sets. That is $\hat{I} = I_\phi$. Obviously, a TRA will remain in a stable status until it is excited by an external input event. This follows directly from the causality requirement for a status succession.

To start executing, a TRA $(\Sigma, \sigma_0, \Pi, \Theta, \Lambda, \Upsilon)$ is put in a stable status (θ_0, I_0), where $I_0 = I_\phi$ and $\theta_0 \in \Theta$. The status (θ_0, I_0) is called an *initial status.* The execution is initiated at time t_0 with the firing of an action π_0 on the start channel σ_0, where $\pi_0 \in \Pi(\sigma_0)$. The event $\langle \pi_0 : t_0 \rangle$ is called the *initiating event.*

An execution e of a TRA is a possibly infinite string of alternating statuses and events, which starts with an initial status followed by an initiating event, and which contains an infinite number of status successions (infinite execution), or terminates in a stable status (finite execution). Since statuses and internal events are invisible from outside a TRA, we will often be interested only in external events. We follow an approach similar to that adopted in [21] by defining β to be a *behavior* of a TRA $\mathcal{A}$, if it consists of all the *external* events appearing in some execution e of $\mathcal{A}$. We denote the set of all the possible behaviors of a TRA $\mathcal{A}$ by $behs(\mathcal{A})$. Obviously, $behs(\mathcal{A})$ describes all the possible interactions that the TRA $\mathcal{A}$ might be engaged in, and, therefore, constitutes a complete specification of the system that $\mathcal{A}$ models.

A TRA $\mathcal{A}$ is said to *implement* another TRA $\mathcal{B}$ if $\mathcal{A}$ does not produce any behavior that $\mathcal{B}$ could have produced. In other words, all of $\mathcal{A}$'s behaviors (the implementation) are possible behaviors of $\mathcal{B}$ (the specification). The reverse, however, is not true. There might exist behaviors of $\mathcal{B}$ that cannot be generated by $\mathcal{A}$. The notion of a TRA implementing another will be used mainly in verification.

2.6 TRA Composition

A basic aspect of the TRA model is its capability to model a complex system by operating on simpler system components. In this section we examine such an operation, namely composition. Other operations (for example hiding and renaming) were presented in [8].

The composition of a countable collection of *compatible* TRAs, $\{\mathcal{A}_i : i \in \mathcal{I}\}$, is a new TRA $\mathcal{A} = \mathcal{A}_0 \times \mathcal{A}_1 \times \ldots \times \mathcal{A}_i \times \ldots = \Pi_{i \in \mathcal{I}} \mathcal{A}_i$. The execution of $\mathcal{A}$ involves the execution of all its components $\mathcal{A}_{i \in \mathcal{I}}$, each starting from an initial status and observing every external event signaled by either the environment (input) or by any TRA in the collection $\{\mathcal{A}_i : i \in \mathcal{I}\}$. The *compatibility* condition for composition insures that, for each channel in the composition, there is at most one writer, a finite number of readers, and that the signaling ranges of readers and writers are compatible.

The input signature of the composed TRA consists of those channels that are inputs to one or more of the component TRAs, and which are not outputs of any of the component TRAs. The output signature of the composed TRA consists of all the outputs of all the component TRAs. Similarily, the internal signature of the composed TRA consists of all the internal channels of all the component TRAs. The start channel of the composed TRA is the start channel of one or more of its component TRAs.[3] The signaling range function of the composed TRA is defined so as to preserve its input-enabled property. In particular, the signaling range of an input channel consists of only those actions that can accepted by all readers of that channel. A computational step of the composed TRA is necessarily a step of one of its

[3] Without loss of generality, we assume that TRA to be $\mathcal{A}_0$.

components. Similarily the time-constrained causal relationships of the composed TRA are exactly those of the component TRAs.

In [8], the formal construction of the sextuple representation of a composition is given. Also, the relationships between the behaviors and spacial properties of the composed TRA and those of its constituent TRAs are established. In particular, we prove that the sets of proper, spontaneous, and causal TRAs are closed under composition.

The TRA composition operation is more general than those reported in [21, 32, 5] in that it allows the specification of both *parallel* and *sequential* composition. In particular, the introduction of the *start channel* permits the execution of two TRAs to be concurrent if they share the same start channel, or to be serialized if the start channel of one (child) is an output of the other (parent). Through appropriate composition, our model is capable of representing all of the composition operations in [23].

3 TRA-based Verification

Verification is the process of establishing the correctness of a system by proving that it preserves certain desired properties. In this section, we overview three verification techniques based on modular, functional, and hierarchical system decomposition.

3.1 Modular Decomposition

One common methodology for verifying properties of a complex system is *modular decomposition*, in which one reasons about each property of the entire system separately.

The input enabling property of the TRA model forbids a system specification from controlling or constraining inputs it receives from its environment. As a result, such a specification can only guarantee properties that are independent from the behavior of the environment. In computer embedded applications, this seems to be the only safe and realistic approach to be adopted. In many circumstances, however, the correct operation of a system is only expected under certain restrictions on its inputs. These restrictions may be guaranteed in the context of a known *installation*, where the behavior of other parts of the environment is a priori certified, or may be assumed by a problem statement, whereby the correct behavior of a solution is required only under a set of specific conditions.

For example, consider the up/down counter $\mathcal{C}$ of Figure-2 and assume that, as a safety condition, $\mathcal{C}$ is required to produce at least one output action in the interval between any two inputs. Obviously, such a requirement cannot be guaranteed without restricting the behavior of the environment feeding the `cmd` signal to the counter. In particular, it can be shown that an unsafe behavior will result if two or more `Up` (or `Down`) actions are fired on the `cmd` channel within less than 1.9 units of time. Now, assume that in a given installation, $\mathcal{C}$ is composed with a subsystem $\mathcal{X}$ that generates `cmd` actions at a slower rate, or a subsystem $\mathcal{Y}$ that issues a new counting request only after it receives the response of $\mathcal{C}$ to its previous request. Obviously, in these restricted environments, the aforementioned safety condition can be indeed certified.

A useful notion for discussing the aforementioned restrictions is that of a TRA *preserving* a property. This notion was introduced in [21] to study fair behaviors of discrete event systems using the IOA model. In this section we generalize this notion to suit the TRA model.

A property $\mathcal{P}$ defines a possibly infinite set of sequences over a given alphabet (or signature). Properties can be defined by specifying them as TRAs, or, alternately, by describing the set of behaviors they allow.[4]

A TRA $\mathcal{P}$ is said to define a *property* for a TRA $\mathcal{A}$ if and only if $\sigma_0^{\mathcal{A}} \in \Sigma_{\text{ext}}^{\mathcal{P}}$. A TRA $\mathcal{A}$ is said to preserve a property $\mathcal{P}$ if it is not the "first" to violate it. Once the property $\mathcal{P}$ is violated, $\mathcal{A}$ is under no obligation to behave in any specific way. That is, the TRA $\mathcal{A}$ behaves according to the property $\mathcal{P}$ until the environment, or possibly another TRA composed with $\mathcal{A}$, violates that property. In [8], this notion of *property preservation* is formally defined by imposing restrictions on the set of behaviors in $behs(\mathcal{A})$ based on the set of behaviors in $behs(\mathcal{P})$.

The following Lemma establishes that property preservation is closed under composition. Thus, if a property is preserved by the constituent components of a composition, then it is preserved by the composition.

Lemma 1 *Let $\{\mathcal{A}_i : i \in \mathcal{I}\}$ be a collection of compatible* TRA*s. If $\mathcal{A}_i$ preserves property $\mathcal{P}$ for all $i \in \mathcal{I}$, then the composition $\prod_{i\in\mathcal{I}} \mathcal{A}_i$ preserves $\mathcal{P}$.*

The notion of a system preserving a property is a weak version of the implementation relationship between TRAs. In particular, a TRA $\mathcal{A}$ *implements* a property $\mathcal{P}$ if $\mathcal{A}$ preserves $\mathcal{P}$ in *all* possible behaviors – independently from the environment's behavior. The following lemma establishes sufficient conditions for a composition of TRAs to implement a property.

Lemma 2 *A set of sufficient conditions for the composition $\mathcal{A} = \prod_{i\in\mathcal{I}} \mathcal{A}_i$ to implement the property $\mathcal{P}$ is that:*

1. $\Sigma_{\text{in}}^{\mathcal{A}} \subseteq \Sigma_{\text{in}}^{\mathcal{P}}$.
2. *$\mathcal{A}_i$ preserves $\mathcal{P}$, for all $i \in \mathcal{I}$.*

[4] Defining a property by specifying it as a TRA has been termed in [34] as the *functional* specification approach, as opposed to the *conventional* black-box approach.

A special case of particular interest occurs when the composition in Lemma-2 is *closed.* A TRA is closed if it has no input channels except the start channel. A closed TRA can be thought of as specifying a system that is *environment independent.* In particular, if $\mathcal{S}$ is the TRA representing an embedded system, and $\mathcal{E}$ is a compatible TRA, where: $\Sigma_{\text{in}}^{\mathcal{S}} = \Sigma_{\text{out}}^{\mathcal{E}} \cup \{\sigma_0^{\mathcal{S}}\}$, $\Sigma_{\text{in}}^{\mathcal{E}} = \Sigma_{\text{out}}^{\mathcal{S}} \cup \{\sigma_0^{\mathcal{E}}\}$, then the composition $\mathcal{S} \times \mathcal{E}$ is closed, and the TRA $\mathcal{E}$ is said to define an *installation* for $\mathcal{S}$.

In the counting example presented in Figure-2, let $\mathcal{P}$ be the property depicting the requirement that any two events on `cmd` will be separated by at least one event on `cnt`. Figure-4 shows a possible sextuple specification of $\mathcal{P}$.

- $\Sigma_{\text{in}} = \{\texttt{init}\}$, $\Sigma_{\text{out}} = \{\texttt{cmd}, \texttt{cnt}\}$, and $\Sigma_{\text{int}} = \phi$.
- $\sigma_0 = \texttt{init} \in \Sigma$.
- $\Pi(\texttt{init}) = \mathcal{Z}$, $\Pi(\texttt{cmd}) = \{\texttt{Up}, \texttt{Down}\}$, and $\Pi(\texttt{cnt}) = \mathcal{Z}$.
- $\Theta = \{\texttt{Go}, \texttt{Wait}\}$.
- $\Lambda = \{(\theta, \texttt{init}(), \theta), (\texttt{Go}, \texttt{cmd}(c), \texttt{Wait}), (\theta, \texttt{cnt}(i), \texttt{Go})\}$, where $\theta \in \Theta$, $c \in \Pi(\texttt{cmd})$, and $i \in \Pi(\texttt{cnt})$.
- $\Upsilon = \{(\texttt{init}, \texttt{cnt}, [0, \infty], \phi), (\texttt{init}, \texttt{cmd}, [0, \infty], \phi),$ $(\texttt{cnt}, \texttt{cnt}, [0, \infty], \phi), (\texttt{cnt}, \texttt{cmd}, [0, \infty], \{\texttt{Wait}\})\}$.

Figure 4: TRA-specification of the property $\mathcal{P}$.

It can be easily shown that both the counter $\mathcal{C}$ and the installation $\mathcal{Y}$ preserve the property $\mathcal{P}$. From Lemma-2, it follows that the closed system resulting from embedding $\mathcal{C}$ in $\mathcal{Y}$, namely the composition $\mathcal{Y} \times \mathcal{C}$, implements $\mathcal{P}$. The same conclusion, although correct, cannot be reached for the composition $\mathcal{X} \times \mathcal{C}$ since, in general, $\mathcal{X}$ does not preserve $\mathcal{P}$. This stems from the fact that the conditions in Lemma-2 are sufficient and not necessary conditions. In particular, Lemma-2 cannot be used for properties that *emerge* from a composition (emergent properties).

3.2 Hierarchical Decomposition

Another methodology for the verification of complex systems is *hierarchical decomposition*, in which one reasons about the entire system at varying levels of abstractions and details. This verification approach is analogous to the *stepwise refinement* implementation approach.

The idea behind hierarchical decomposition is to prove that a given TRA implements a second, that the second implements the third, and so on until the final TRA is shown to implement the required specifications. The transitivity of the implementation relation guarantees that the first TRA, indeed, implements the specifications.

In the remainder of this section, we derive a set of sufficient conditions for the (strong) implementation of a TRA by another. The idea is to come up with a mapping Ψ between the states and intentions of the two TRAs and show that any possible status succession in the implementing TRA corresponds to some possible succession in the specification TRA. Figure-5 illustrates that correspondence.

Our approach in establishing a mapping between a specification and its implementation is similar to the possibilities mappings proposed in [21, 20] and the prophecy mappings proposed in [25], except that it is complicated here by the need to preserve the timing constraints of the specification TRA. The following lemma establishes the required sufficient conditions.

Lemma 3 *A set of sufficient conditions for a TRA $\mathcal{A}$ to implement another TRA $\mathcal{B}$ is that both of the following conditions are satisfied:*

1. – $\Sigma_{\text{in}}^{\mathcal{A}} = \Sigma_{\text{in}}^{\mathcal{B}} = \Sigma_{\text{in}}$, *and*
   – $\forall \sigma_i \in \Sigma_{\text{in}} : \Pi^{\mathcal{A}}(\sigma_i) = \Pi^{\mathcal{B}}(\sigma_i)$.
2. *There exist two mappings: $\Psi_\Theta : \Theta^{\mathcal{A}} \rightarrow 2^{\Theta^{\mathcal{B}}}$, from the set of states of $\mathcal{A}$ to the power set of states of $\mathcal{B}$, and $\Psi_I : I^{\mathcal{A}} \rightarrow 2^{I^{\mathcal{B}}}$, from the set of intentions of $\mathcal{A}$ to the power set of intentions of $\mathcal{B}$, such that the following conditions hold:*
   a. $\Psi_I(I_\phi^{\mathcal{A}}) = \{I_\phi^{\mathcal{B}}\}$,
   b. *Let (θ_i, I_i) be a reachable status of the TRA $\mathcal{A}$, and let (θ_i', I_i') be a reachable status of the TRA $\mathcal{B}$, where $\theta_i' \in \Psi_\Theta(\theta_i)$, and $I_i' \in \Psi_I(I_i)$. If $(\theta_i, I_i) \xrightarrow{\langle \pi : t \rangle} (\theta_j, I_j)$ is a possible status succession of $\mathcal{A}$, then there exists an extended status succession*[5] *for $\mathcal{B}$ of the form $(\theta_i', I_i') \xrightarrow{\alpha} (\theta_j', I_j')$, such that:*
      i. $\alpha | \Sigma_{\text{ext}}^{\mathcal{B}} = \langle \pi : t \rangle | \Sigma_{\text{ext}}^{\mathcal{A}}$.
      ii. $\theta_j' \in \Psi_\Theta(\theta_j)$, *and* $I_j' \in \Psi_I(I_j)$.

As an example, let us focus once more on the counting example of Figure-2 and the installation $\mathcal{X}$ that we described before. Recall that, using modular decomposition, we were unable to verify that the composition $\mathcal{X} \times \mathcal{C}$ preserves the property $\mathcal{P}$ stating that any two events on `cmd` are separated by at least one event on `cnt`. Using the sufficient conditions of Lemma-3, such a proof can be constructed.

In particular, consider the TRA sextuples for the installation $\mathcal{X}$ and the closed system $\mathcal{X} \times \mathcal{C}$ shown in Figure-6 and Figure-7, respectively.

Now, consider the mappings $\Psi_\Theta : \Theta^{\mathcal{X} \times \mathcal{C}} \rightarrow 2^{\Theta^{\mathcal{P}}}$ and $\Psi_I : I^{\mathcal{X} \times \mathcal{C}} \rightarrow 2^{I^{\mathcal{P}}}$ given below:

◇ $\Psi_\Theta((Q, \theta_i)) = \{\texttt{Go}, \texttt{Wait}\}$, where $i \in \mathcal{Z}$.

[5] See Figure-5 for an illustration.

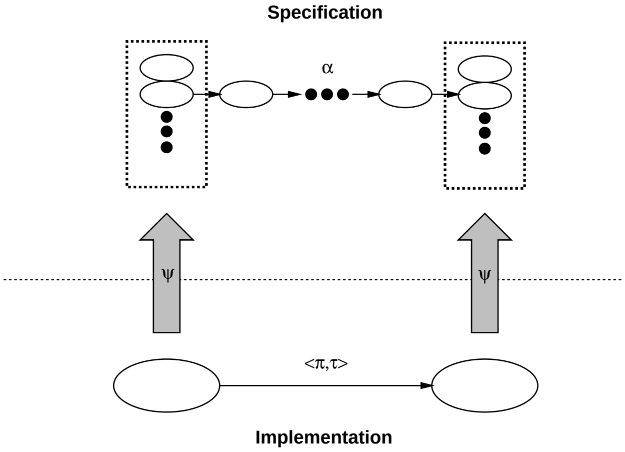

Figure 5: Ψ-mapping.

$\bullet \Sigma_{\rm in} = \{\texttt{init}\}$, $\Sigma_{\rm out} = \{\texttt{cmd}\}$, and $\Sigma_{\rm int} = \phi$.
$\bullet \sigma_0 = \texttt{init}$.
$\bullet \Pi(\texttt{init}) = \mathcal{Z}$, $\Pi(\texttt{cmd}) = \{\texttt{Up}, \texttt{Down}\}$.
$\bullet \Theta = \{Q\}$.
$\bullet \Lambda = \{(Q, \texttt{init}(), Q), (Q, \texttt{cmd}(c), Q)\}$,
where $c \in \Pi(\texttt{cmd})$.
$\bullet \Upsilon = \{(\texttt{init}, \texttt{cmd}, [4.9, 5.1], \phi), (\texttt{cmd}, \texttt{cmd}, [4.9, 5.1], \phi)\}$.

Figure 6: TRA-specification of the installation $\mathcal{X}$.

$\diamond$ $\Psi_I([\{\delta_0\}, \{\delta_1\}, \{[t_l, t_h]\}, \{[t'_l, t'_h]\}]) =$
$\{[\{\delta_0\}, \{\delta_1\}, \{[t_l, t_h]\}, \varepsilon], [\{\delta_0\}, \{\delta_1\}, \{[t_l, t_h]\}, \{[t'_l, t'_h]\}]\}$,
if $t'_l > t_h$,
$\{[\{\delta_0\}, \{\delta_1\}, \{\delta_2\}, \{[t_l, t_h]\}]\}$,
otherwise.

The mapping Ψ_Θ reflects the fact that there is no direct correspondence between the states of $\mathcal{X} \times \mathcal{C}$ and those of $\mathcal{P}$; any state of $\mathcal{X} \times \mathcal{C}$ maps to any state of $\mathcal{P}$. The mapping Ψ_I, however, reflects how the combined time constraints of $\mathcal{X}$ and $\mathcal{C}$ interrelate to guarantee the satisfy the emergent property $\mathcal{P}$. In particular, the mapping Ψ_I implies that in the composition $\mathcal{X} \times \mathcal{C}$, an intention to fire an action on the `cmd` channel necessitates a similar intention in $\mathcal{P}$ only if `cnt` cannot be guaranteed to fire before `cmd`. The proof is established by checking that the mappings Ψ_Θ and Ψ_I satisfy the conditions of Lemma-3.

$\bullet \Sigma_{\rm in} = \{\texttt{init}\}$, $\Sigma_{\rm out} = \{\texttt{cmd}, \texttt{cnt}\}$, and $\Sigma_{\rm int} = \phi$.
$\bullet \sigma_0 = \texttt{init}$.
$\bullet \Pi(\texttt{init}) = \mathcal{Z}$, $\Pi(\texttt{cmd}) = \{\texttt{Up}, \texttt{Down}\}$, and $\Pi(\texttt{cnt}) = \mathcal{Z}$.
$\bullet \Theta = \{(Q, \theta_i) : i \in \mathcal{Z}\}$.
$\bullet \Lambda = (\bigcup_{i,j \in \mathcal{Z}} \{((Q, \theta_i), \texttt{init}(j), (Q, \theta_j))\}) \cup$
$(\bigcup_{i \in \mathcal{Z}} \{((Q, \theta_i), \texttt{cmd(Up)}, (Q, \theta_{i+1}))\}) \cup$
$(\bigcup_{i \in \mathcal{Z}} \{((Q, \theta_i), \texttt{cmd(Down)}, (Q, \theta_{i-1}))\}) \cup$
$(\bigcup_{i \in \mathcal{Z}} \{((Q, \theta_i), \texttt{cnt}(i), (Q, \theta_i))\})$.
$\bullet \Upsilon = \{(\texttt{init}, \texttt{cnt}, [1.9, 2.1], \phi), (\texttt{cnt}, \texttt{cnt}, [1.9, 2.1], \phi),$
$(\texttt{init}, \texttt{cmd}, [4.9, 5.1], \phi), (\texttt{cmd}, \texttt{cmd}, [4.9, 5.1], \phi)\}$.

Figure 7: TRA-specification of the composition $\mathcal{X} \times \mathcal{C}$.

3.3 Functional Decomposition

System verification using functional decomposition is strongly tied to system implementation using the divide-and-conquer approach. In particular, one way of verifying an implementation is by dividing the problem to be solved into independent sub-problems, and verifying the implementation (or preservation) of each sub-problem separately. The following Lemma formalizes this concept.

Lemma 4 *Let $\{\mathcal{A}_i : i \in \mathcal{I}\}$ and $\{\mathcal{P}_i : i \in \mathcal{I}\}$ be two collections of compatible* TRA*s. If $\mathcal{A}_i$ implements (preserves) $\mathcal{P}_i$, for every $i \in \mathcal{I}$, then the composition $\mathcal{A} = \prod_{i \in \mathcal{I}} \mathcal{A}_i$ implements (preserves) the composition $\mathcal{P} = \prod_{i \in \mathcal{I}} \mathcal{P}_i$.*

The TRA formalism admits only one combinator for TRAs, namely the composition operation. While such a combinator is adequate for the specification of systems out of their components, other combinators, like conjunction and disjunction, might prove useful for the description of properties. More work needs to be done to formally define such combinators.

4 TRA-based System Design

The rational behind proposing the TRA formalism is that it can serve as the backbone of a development methodology for embedded real-time applications. To achieve that goal, we have developed $\mathcal{CLEOPATRA}$[6] – a convenient language for the specification of embedded systems under the TRA formalism. $\mathcal{CLEOPATRA}$ specifications can be compiled and executed efficiently in simulated time. Also, they can be transformed, mechanically and unambiguously, into formal TRA objects for verification purposes.

In $\mathcal{CLEOPATRA}$, systems are specified as interconnections of objects which are instantiations of parametrized TRA-*classes*. For example, Figure-8 shows the $\mathcal{CLEOPATRA}$ specification of three TRA-classes, `integrate`, `constant`, and `ramp`.

The *header* of a TRA-class determines its external signature and its signaling range function. For example, any TRA from the class `integrate` has a signature consisting of an input channel `in` and an output channel

[6] A $\mathcal{C}$-based $\mathcal{L}$anguage for the $\mathcal{E}$vent-driven $\mathcal{O}$bject-oriented $\mathcal{P}$rototyping of $\mathcal{A}$synchronous $\mathcal{T}$ime-constrained $\mathcal{R}$eactive $\mathcal{A}$utomata.

`out`. Both `in` and `out` carry actions whose values are drawn from the set of reals. The body of a TRA class determines the behavior of objects from that class. Such a behavior can be either *basic* or *composite*.

The description of a basic behavior involves the specification of a state space in the `state:` section, the specification of an initialization of that space in the `init:` section, and the specification of a set of causal, time-constrained reactions in the `act:` section. For example, the behavior of an object belonging to the TRA-class `integrate` is basic. It consists of two reactions. The first specifies the reaction of the integrator to events on the input channel `in()`. The second specifies a periodic signaling on the output channel `out()`. Composite behaviors are specified by composing previously defined, simpler TRA-classes together in the `include:` section. For example, in Figure-8, the class `ramp` is defined by composing the `integrate` and `constant` classes together.

```
TRA-class integrate(double TICK, TICK_ERR)
     in(double) -> out(double)
{
 state:
  double x0 = 0, x1 = 0, y = 0;
 act:
  in(x1) -> :
     ;
  init(),out() -> out(y):
    within [TICK-TICK_ERR~TICK+TICK_ERR]
      commit { y=y+TICK*(x0+x1)/2; x0=x1; }
}


TRA-class constant(double VAL,TICK,TICK_ERR)
  -> out(double)
{
  act:
    init(), out() -> out(VAL):
      within [TICK-TICK_ERR~TICK+TICK_ERR] ;
}


TRA-class ramp
  -> y(double)
{
  internal:
    x(double) -> ;
  include:
    constant -> x() ;
    integrate x() -> y() ;
}
```

Figure 8: Integrations using trapezoidal approximation.

We have used *CLEOPATRA* to study a range of real-time digital systems. In particular, we used it to specify and verify asynchronous circuits [4] and to specify, analyze and simulate behaviors of autonomous creatures [6, 7].

To close up the gap between formality and practicality, the development cycle of embedded applications has to be supported in its entirety. This requires that systems implementation – and not only specification, validation and verification – be addressed. We are currently developing a compiler that would make of *CLEOPATRA* a real-time programming language. The testbed for our methodology is a robotic experiment, which involves the coordination of motor requests to perform manipulative tasks using directed-vision feedback. An initial report on that experiment can be found in [9].

5 Conclusion

Previous studies in modeling real-time computing have focussed on adding the notion of time to formal modeling techniques of traditional systems without regard to the physical realities of the modeled systems. In this paper, we propose the TRA model as an attempt at addressing some of the issues involved therein.

The TRA model is a physically sound formalism for real-time embedded computations. Among its salient features is a fundamental notion of space and time that restricts the expressiveness of the model in a way that allows the specification of only reactive, spontaneous, and causal computations. Using the TRA model, an embedded system is viewed as a set of asynchronously interacting automata (TRAs), each representing an autonomous system entity. TRAs are input enabled; they communicate by signaling events on their output channels and by reacting to events signaled on their input channels. The behavior of a TRA is governed by time-constrained causal relationships between computation-triggering events. The TRA model is compositional and allows benign time, control, and computation nondeterminism.

We envision the TRA model as the backbone of a methodology for the development of real-time embedded systems. In addition to specification and verification, such a methodology would support validation, prototyping, and implementation. To that end, we have developed *CLEOPATRA*, a TRA-based specification language. *CLEOPATRA* specifications can be compiled and executed in simulated time for validation purposes. We are currently working on a compiler that would allow the real-time execution of realizable *CLEOPATRA* specifications, thus making of it a programming language for embedded real-time systems. The testbed for our experiments is a robotic application, in which the real-time management of sensori-motor activities of an industrial robot arm will be developed using *CLEOPATRA*.

Acknowledgment: This research was partially conducted while the author was at Harvard University and was partially supported by DARPA N00039-88-C-0163.

References

[1] J. Allen. Towards a general theory of action and time. *Artificial Intelligence*, 23:123–154, 1984.

[2] Rajeev Alur, Costas Courcoubetis, and David Dill. Model-checking for real-time systems. In *Proceedings of the 5th annual IEEE Symposium on Logic in Computer Science*, Philadelphia, Pensylvania, June 1990. IEEE Computer Society Press.

[3] J. Baeten and J. Bergstra. Real time process algebra. *Formal Aspects of Computing*, 3(2):142–188, 1991.

[4] Azer Bestavros. A new environment for developing real-time embedded systems. Internal Report – Department of Computer Science, Harvard University, April 1989.

[5] Azer Bestavros. The IOTA: A model for real-time parallel computation. In *Proceedings of TAU'90: The 1990 ACM International Workshop on Timing issues in the Specification and Synthesis of Digital Systems*, Vancouver, Canada, August 1990.

[6] Azer Bestavros. TRA-based real-time executable specification using CLEOPATRA. In *Proceedings of the 10th Annual Rochester Forth Conference on Embedded Systems*, Rochester, NY, June 1990. (revised May 1991).

[7] Azer Bestavros. Planning for embedded systems: A real-time prospective. In *Proceedings of AIRTC-91: The 3rd IFAC Workshop on Artificial Intelligence in Real Time Control*, Napa/Sonoma Region, CA, September 1991.

[8] Azer Bestavros. *Time-constrained Reactive Automata: A novel development methodology for embedded real-time systems.* PhD thesis, Harvard University, Division of Applied Sciences (Department of Computer Science), Cambridge, Massachusetts, September 1991.

[9] Azer Bestavros, James Clark, and Nicola Ferrier. Management of sensori-motor activity in mobile robots. In *Proceedings of the 1990 IEEE International Conference on Robotics & Automation*, Cincinati, Ohio, May 1990. IEEE Computer Society Press.

[10] Gregor Bochmann. Hardware specification with temporal logic: An example. *IEEE transactions on Computers*, C-31(3), March 1982.

[11] J. E. Coolahan and N. Roussopoulos. Timing requirements for time-driven systems using augmented petri nets. *IEEE Transactions on Software Engineering*, SE-9:603–616, September 1983.

[12] B. Dasarathy. Timing constraints of real-time systems: Control for expressing them, Method for validating them. *IEEE Transactions on Software Engineering*, 11(1), January 1985.

[13] R. Gerber, I. Lee, and A. Zwarico. A complete axiomatization of real-time processes. CIS, University of Pennsylvania – Submitted for publication, February 1989.

[14] Asis Goswami and Mathai Joseph. What's "real" about real-time systems? Research Report 123, Department of Computer Science, University of Warwick, April 1988.

[15] Stephen W. Hawking. *A brief history of Time: From the Big Bang to Black Holes.* Bantam Books, April 1988.

[16] Farnam Jahanian and Aloysius Mok. Safety analysis of timing properties in real-time systems. *IEEE Transaction on Software Engineering*, 12(9):890–904, 1986.

[17] Nancy Leveson and Janice Stolzy. Safety analysis using Petri Nets. *IEEE Transactions on Software Engineering*, 13(3):386–97, March 1987.

[18] Harry Lewis. Finite-state analysis of asynchronous circuits with bounded temporal uncertainty. Technical Report TR-15-89, Department of computer science, Harvard University, Cambridge, MA, June 1989.

[19] Harry Lewis. A logic of concrete time intervals. In *Proceedings of the 5th annual IEEE Symposium on Logic in Computer Science*, Philadelphia, PA, June 1990. IEEE Computer Society Press.

[20] Nancy Lynch and Kenneth Goldman. 6.852 distributed algorithms lecture notes: The I/O Automata. Technical report, Laboratory of Computer Science, MIT, Cambridge, MA, Fall 1988.

[21] Nancy Lynch and Mark Tuttle. An introduction to Input/Output Automata. Technical Report MIT/LCS/TM-373, MIT, Cambridge, Massachusetts, November 1988.

[22] Nancy Lynch and Frits Vaandrager. Forward and backward simulations for timing-based systems. Unpublished notes, Massachusetts Institute of Technology Laboratory for Computer Science, August 1991.

[23] Damian Lyons and Michael Arbib. A formal model of computation for sensory-based robotics. *IEEE Transactions on Robotics and Automation*, 5(3):280–293, 1989.

[24] P. M. Merlin and D. J. Faber. Recoverability of communication protocols: Implications of a theoretical study. *IEEE Transactions on Communication*, COM-24:1036–1043, September 1976.

[25] Michael Merritt. Completness theorems for automata, May 1989. In *REX Workshop*.

[26] Amir Pnueli. The temporal logic of programs. In *Proccedings of the IEEE Annual Symposium on Foundations of Computer Science*, November 1977.

[27] C. Ramchandani. *Analysis of asynchronous concurrent systems by timed Petri nets.* PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, 1974. Project MAC Report MAC-TR-120.

[28] G. M. Reed and A. W. Roscoe. A timed model for Ccommunicating Sequential Processes. *Theoretical Computer Science*, 58:249–261, 1988.

[29] Fred Schneider. Critical (of) issues in real-time systems: A position paper. Technical Report 88-914, Department of Computer Science, Cornell University, Ithaca, NY, May 1988.

[30] Ramavarapu Sreenivas. *Towards a system theory for interconnected Condition/Event systems.* PhD thesis, Carnegie Mellon University, Pittsburgh, PA, September 1990.

[31] John Stankovic and Krithi Ramamritham, editors. *Hard Real-Time Systems.* IEEE Computer Society Press, 1988.

[32] Mark Tuttle, Michael Meritt, and Francesmary Modugno. Time constrained automata. MIT/LCS, November 1988.

[33] Niklaus Wirth. Toward a discipline of real-time programming. *Communications of the ACM*, 20(8), August 1977.

[34] Pamela Zave. An operational approach to requirements specification for embedded systems. *IEEE Transactions on Software Engineering*, 8(3), May 1982.