In Proc. of the 1990 ACM Intl. Workshop on Timing issues in the Specification and Synthesis of Digital Systems, Vancouver, Canada, Aug 1990.

The Input Output Timed Automaton

A model for real-time parallel computation

Azer Bestavros*
Harvard University

Abstract

In this paper, we present the Input Output Timed Automaton (IOTA) model for the specification and verification of parallel real-time digital systems. Our model extends Nancy Lynch's IOA model [Lynch:88a] to allow for the inclusion of timing properties. We have used the IOTA model to study a range of real-time digital systems. In particular, we used it to specify and verify asynchronous circuit designs [Bestavros:89a] and to specify, analyze and simulate behaviors of autonomous creatures [Bestavros:90b]. In [Bestavros:90a], we developed a language and environment for the executable specification of parallel real-time interactive tasks based on the IOTA model.

A IOTA is an abstraction that encapsulates a real-time task. A system is modeled as a network of such IOTAs communicating with each other over channels. A IOTA might be the specification or the actual implementation of a software/hardware module. This allows the representation of both the external environment and the programmed system along with the available computational resources in a unique framework. IOTAs can be *composed* together to form higher level IOTAs. A IOTA *implements* another, if all *external behaviors* of the first (the implementation) are also external behaviors of the second (the specification.) We use timed possibility mappings to test for the implementation relationship. This is the primary tool that is used to verify that an implementation meets the required specification.

*This work was supported by DARPA N00039-88-C-0163

1 It's *Time* to make Real-Time Computing *Real !*

Real-time computing systems are and will continue to play an increasingly vital role in our world. Current real-time systems are expensive to build and their properties are usually verified with ad hoc techniques, or with expensive and extensive simulations [Stankovic:88a]. Different components of such systems are extremely difficult to integrate with each other which adds to the cost and complexity of these systems. Minor changes in any of these components result in another round of testing and fixing [Zave:82]. This brute force approach is not likely to scale-up with future systems. It has become clear that more work is needed in the area of real-time systems if we are to meet the needs and challenges of the future.

Viewed simply, any real-time system has two parts: an external environment and a programmed system. The *external environment* consists of a number of devices such as sensors and actuators that interact with the real-world. The *programmed system* collects information from the sensors and responds by producing actions to drive the actuators. The continual demands of the "unintelligent" external environment poses relatively rigid and urgent requirements on the performance of the programmed system. These requirements are usually stated in terms of real-time constraints. Examples of such systems include ballistic-missile-defense systems, nuclear reactors, robotics, process control plants, flight and space shuttle control, real-time databases, real-time communication networks *...etc.* [Zave:81], [Zave:82]. The complexity of these systems coupled with an obvious lack of powerful specification and verification techniques made it impossible to assess their performance or even to judge their correct operation.

In the past few years, different aspects of real-time systems have been studied, namely: specification, languages, models and semantics. However, the absence of a unified formal framework that addresses the afforementioned issues severely limited the usefulness of these studies [Stankovic:88a], [Joseph:88]. In this paper, we present a single framework for real-time systems which makes it possible to see the relation between specification, implementation, correctness, and performance issues. The framework we suggest is based on the *Input-Output Timed Automaton* (IOTA – read "yota") model.

We have used the IOTA framework to model robotics applications [Bestavros:90b]. The tasks involved in a robotic application are diverse (vision [BBN:86], motion control [Brockett:88], high-level planning and behavioral specification [Brooks:86], *...etc.*) and usually make use of very different resources (special purpose image processors, tailor made controllers and drivers, general purpose processors, *...etc.*) Furthermore, these tasks interact in a non-trivial way. Being able to specify and verify such complex systems in a single framework is both challenging and attractive.

2 The IOTA model: An introduction

In this section, we introduce the *Input-Output Timed Automaton* (IOTA) which is an extension of Nancy Lynch's *Input-Output Automaton* (IOA) introduced in [Lynch:88a] to study discrete event systems (*e.g.* distributed databases [Lynch:89b].) The IOTA model provides a formal semantics definition for ESPRIT [Bestavros:90a], a specification language for real-time systems. We used the IOTA model and the ESPRIT language to specify and analyze a number of real-time applications [Bestavros:90b].

2.1 Channels, Signals, Events and Actions

In our model, time is a measurable, continuous, infinitely divisible quantity. We represent any point in time by a nonnegative real $t \in \Re$. Time intervals are defined by specifying their endpoints which are drawn from the set of nonnegative rationals or nonnegative integers $\mathcal{D} \subset \Re$. Time intervals might be closed, open or semi-open. If t_1 and t_2 are two points in time, $t_1, t_2 \in \mathcal{D}$, then $[t_1, t_2], (t_1, t_2], [t_1, t_2), (t_1, t_2)$ are the possible closed, right-closed, left-closed, and open intervals, respectively. We use $\{t_1, t_2\}$ to denote any one of these cases.

A real-time system is viewed as a set of interacting IOTAs. IOTAs communicate with each other and with the external environment through *channels*. A channel is an abstraction for an *ideal* communication media. Among others, a channel might represent a simple physical wire that carries an electrical signal be it binary or analogue. It might represent a Unix-like socket or pipe. Or, it might represent a function invocation and a possible transfer of information through function calls and returns. The information that a channel carries is called a *signal*. A signal consists of a sequence of *events*. An event underscores the instantiation of an *action* at a specific point in time. Figure-1 illustrates the notions of channels, actions, events and signals.

Our concept of actions is similar to that used in the IOA model [Lynch:88a]. In particular, we view actions as *tokens* that are generated as outputs and consumed as inputs. More formally, let a_i be an action of a IOTA $\mathcal{A}$ that was signaled at time t_i and remained active[1] till time t_j, where $t_j > t_i$, when another action a_j was signaled. The tuples (a_i, t_i) and (a_j, t_j) are called events.[2] Furthermore, the sequence of events $(a_1, t_1), (a_2, t_2), (a_3, t_3), \ldots, (a_k, t_k), \ldots$ is called a signal.

As in [Lynch:88a], channels[3] are classified as *internal* or *external* and as *input* or *output*. Signals from external channels are observable from outside the IOTA. Signals from internal channels are only observable locally. Signals from input channels are uncontrollable since they are supplied by the environment or by other IOTAs. Unlike other models [Hoare:78], they cannot be blocked. Signals from output channels, on the other hand, are the way an IOTA reacts to the outside stimuli. Thus, a IOTA can be considered as performing a mapping from its external input signals to its external output signals. In a sense, it acts as an actor in a dataflow graph [Veen:86].

As was done in [Lynch:88a], we formally define the *signature* $sig(\mathcal{A})$ of a IOTA $\mathcal{A}$ to be a partition on the channels of $\mathcal{A}$ into three disjoint sets of input, output and internal channels. For a IOTA $\mathcal{A}$, we denote these sets by $in(sig(\mathcal{A}))$, $out(sig(\mathcal{A}))$ and $int(sig(\mathcal{A}))$, respectively. The set consisting of both input and output channels is the set of external channels observable from outside the IOTA. We denote this set by $ext(sig(\mathcal{A}))$, where:

$$ext(sig(\mathcal{A})) = in(sig(\mathcal{A})) \cup out(sig(\mathcal{A}))$$

The set consisting of both output and internal channels is the set of local channels. These are the locally controlled channels of $\mathcal{A}$. We denote this set by $loc(sig(\mathcal{A}))$, where:

$$loc(sig(\mathcal{A})) = out(sig(\mathcal{A})) \cup int(sig(\mathcal{A}))$$

We denote the set of all the channels of a IOTA $\mathcal{A}$ by $channels(\mathcal{A})$, where:

$$channels(\mathcal{A})) = in(sig(\mathcal{A})) \cup out(sig(\mathcal{A})) \cup int(sig(\mathcal{A}))$$

[1] By "remained active" we mean that no other actions were activated on the same channel during that period

[2] The interval for which the action a_i was active is $[t_i, t_j)$.

[3] and consequently signals and actions that they carry

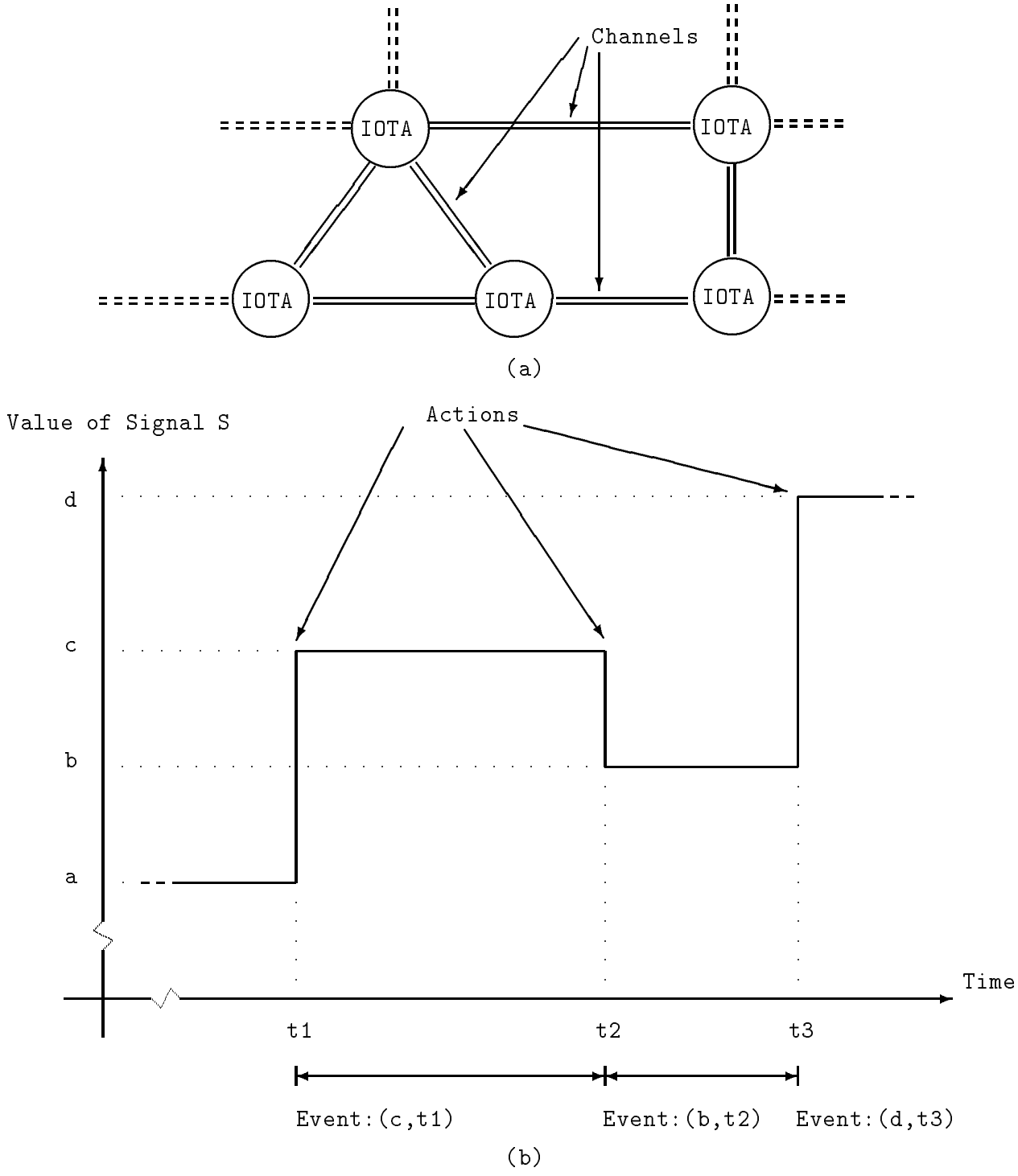


Figure 1: (a) IOTAs communicate via channels. (b) Actions, Events and signals.

Finally, as with channels, we classify actions, events and signals as being input/output, internal/external or local.

The *value* of a signal at any point in time t_r is defined to be the action that was active at that time. A special value, $\perp$ (null), denotes the *absence* of any actions. This is useful to model systems where actions are signaled on and off. In these situations, signaling off can be represented by signaling on a null action. Note that this allows the representation of both *continuous* and *discontinuous* signals. In particular, we call a signal *continuous* if it has a non-null value at *every* point in time. Otherwise, the signal is called *discontinuous*.

We formally define the *signaling range*, $range_{\mathcal{A}}()$, of a IOTA $\mathcal{A}$ to be a function that maps each channel of $\mathcal{A}$, to a (possibly infinite) range of non-null values that the channel might signal. These are the actions associated with that channel. For instance, if `MOVE` is a channel of $\mathcal{A}$ and $range_{\mathcal{A}}(\texttt{MOVE}) = \{\texttt{North}, \texttt{South}, \texttt{East}, \texttt{West}\}$ then the channel `MOVE` can only carry four actions: `MOVE(North)`, `MOVE(South)`, `MOVE(East)`, `MOVE(West)`, or the null action `MOVE`($\perp$). We denote the set of all possible actions of a IOTA $\mathcal{A}$ by $actions(\mathcal{A})$, where:

$$actions(\mathcal{A}) = \{c(a) : a \in range_{\mathcal{A}}(c), c \in channels(\mathcal{A})\}$$

A signal cannot convey more than one event simultaneously (*i.e.* at any point in time). This is guaranteed by requiring that for a signal $(a_1, t_1), (a_2, t_2), \ldots, (a_k, t_k), \ldots$: $t_{i+1} > t_i, i \geq 1$. As a matter of fact, we associate with every channel in the system a non-zero minimum *switching time* that determines the minimum time between successive events that can be signaled on that channel. This switching time is needed for physical as well as for formal reasons. Physically speaking, signaling an event requires energy and thus time. Imposing a positive switching time means that channels have *finite* capacities and thus cannot carry infinitely many events at the same time. Formally speaking, the switching time is necessary for conducting a finite state analysis on the system's possible behaviors [Lewis:89], [Lewis:90].

We define the *switching time* $switch_{\mathcal{A}}()$ of a IOTA $\mathcal{A}$ to be a function that maps the set of channels of the IOTA, to the set of positive rationals or positive integers $\mathcal{D}^+$. For instance, if c_i is a channel of $\mathcal{A}$, then $switch_{\mathcal{A}}(c_i) \in \mathcal{D}^+$ denotes the minimum switching time for the channel c_i.

The above definitions are somewhat different from those given in [Lynch:88a]. In particular, we have introduced the notion of signals and channels as useful abstractions to represent communicated information and communication media respectively. In our model, a number of actions[4] can be communicated through one channel. For example, consider the modeling of a motor controller [Bestavros:88] as a IOTA. A possible command that the controller might accept would be `ROTATE`(θ) in which case the motor is expected to rotate θ degrees. Using the IOA model a different input action would have been necessary for each possible value of θ. It is more natural, however, to represent the input to the controller with a single channel `ROTATE` which at different points in time carries different values for θ and thus different commands or actions.[5] Thus, it makes sense to use channels rather than individual actions in the definition of the signature of a IOTA and to explicitly specify the possible ranges of these channels.

Another deviation of the IOTA model from the original IOA model is with respect to the notion of *fairness*. In the IOA model [Lynch:88a] and its timed extension [Tuttle:88], the local

[4] possibly infinite

[5] It is possible for the channel to carry no information, in which case we say that the signal is absent (*i.e.* it has a null value).

actions of an automaton are partitioned into equivalence classes to capture the intuition of an automata being composed of a number of components. Each class of the partition is thought of as the set of actions under the local control of one of the system's components. The main reason for defining this partition is to insure *fair* executions, where each of the system's components gets infinitely many chances to perform output or internal actions. In the IOTA model we have chosen not to include this kind of partitioning explicitly. Instead, we decided that all the actions that can be signaled on a specific channel are in the same class. That is, every local channel defines an equivalence class by itself and, in an infinite execution, it is given the chance to change its value infinitely often.[6] In real-time systems, the major concern is *safe*[7] and not necessarily *fair* executions [Schneider:88]. Thus, in studying the behavior of IOTAs, we do not consider fairness to be a major concern.

Finally, in Lynch's model [Lynch:88a], [Lynch:88b] and its real-time extension [Tuttle:88], there is no notion of the channel *switching time*. As we have explained above, such a notion is necessary for a realistic modeling of real-time systems.

2.2 States, Computations and State transitions

At any point in time, a IOTA is in one of a possible set of *states*. We denote by $states(\mathcal{A})$ the set of possible states of a IOTA $\mathcal{A}$. A distinguished subset of this set is known as the set of start states of $\mathcal{A}$ and is denoted by $start(\mathcal{A})$. Neither $states(\mathcal{A})$ nor $start(\mathcal{A})$ need to be finite. The state of a IOTA is observable only locally and can be changed only by local *computations*. Computations (and thus state transitions) are triggered by actions and can be scheduled to meet specific timing constraints.

A channel changes its value by signaling (firing) a new action. Once this happens, the computation associated with that action is performed. This might result in a state transition. The firing of an action, along with the necessary computation and state transition, are assumed to be done *atomically*: that is *indivisibly* and *irreversibly*.

Input signals cannot be blocked. Thus, input channels are always enabled. Local channels, on the other hand, are enabled only when the IOTA is in one of some specific set of states. We, therefore, associate with each local channel of the IOTA a *pre-condition* that determines when that channel should be enabled to change its value. Note that the specific action that would fire, is state dependent. Thus, the state of a IOTA determines which channel(s) are enabled and which action(s) are fired.

To reflect the possible delays associated with computations, we associate lower and upper time bounds with each local channel to determine when its signal will change value (by firing an action) if the channel ever becomes enabled. For instance assume that τ_l^c and τ_u^c are the lower and upper time bounds associated with a local channel c. Furthermore, assume that c became enabled at time t_i, then c can fire at any time $t_j \in [t_i + \tau_l^c, t_i + \tau_u^c]$ provided that it remained enabled during the time interval $[t_i, t_j)$. In a sense, these time bounds define a timing constraint on the response of the IOTA to some conditions.

The pre-condition associated with a local channel c partitions the set of states of the IOTA into two disjoint subsets S_t^c and S_f^c. S_t^c is the set of states in which the pre-condition is true and thus the signal is enabled. S_f^c is the set of states in which the pre-condition is false and thus the

[6] if it becomes enabled infinitely often.

[7] That is executions where all timing constraints are satisfied.

signal is not enabled. Let $\{\tau_l^c, \tau_u^c\}$ represent the timing constraints associated with the channel c. These constraints implies that if the IOTA is in one of the states in S_t^c then in the time interval defined by $\{\tau_u^c, \tau_l^c\}$,[8] the IOTA will signal on c an action from the set $range_{\mathcal{A}}(c)$, or else, it will go to one of the states in S_f^c. In other words, an action is going to be taken on c "in time" unless disabled. Formally, this timing constraint is expressed as follows:

$$S_t^c \overset{\{\tau_l^c, \tau_u^c\}}{\rightsquigarrow} (S_f^c, range_{\mathcal{A}}(c))$$

For a IOTA $\mathcal{A}$, the set of all such timing constraints (one per local channel) is denoted by $delays(\mathcal{A})$.

2.3 Executions, Schedules, Behaviors and Projections

The computations and state transitions of a IOTA $\mathcal{A}$, can be formally described by a transition relation $steps(\mathcal{A}) \subseteq states(\mathcal{A}) \times actions(\mathcal{A}) \times states(\mathcal{A})$. Each element of this transition relation represents a possible step in the computation of the system that the IOTA models. Thus a computation step is a triplet $(s, \pi, s') \in steps(\mathcal{A})$ describing the transition from state s to the state s' through the occurrence of the action π. Note that in state s, the channel firing the action π is enabled. Furthermore, state s' captures the effect of firing π starting from s (*i.e.* atomically and irreversibly executing the computations associated with π starting from state s.)

We adopt the same definitions given in [Lynch:88a]. In particular, an execution fragment of the system that a IOTA models can be described with a possibly infinite string of alternating states and actions of the form $s_0, \pi_1, s_1, \pi_2, \ldots, \pi_n, s_n$. An *execution* is simply an execution fragments that begins with a start state (*i.e.* $s_0 \in start(\mathcal{A})$). The set of all possible executions of a IOTA $\mathcal{A}$ is denoted by $execs(\mathcal{A})$.

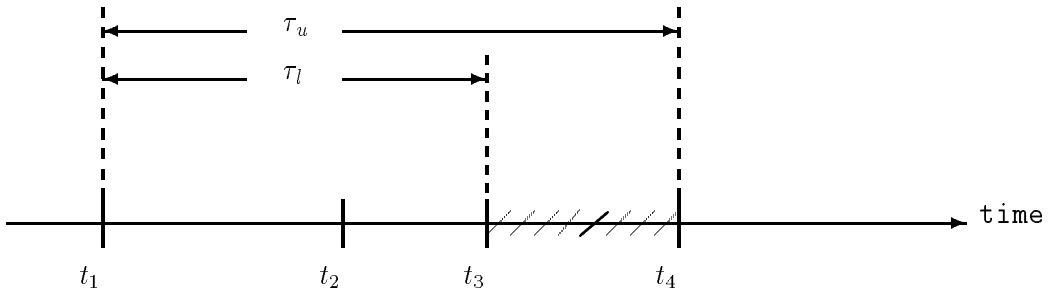
Since the states of a IOTA are unobservable (except locally), we will be often interested in sequences of actions rather than sequences of steps. The *schedule* α of an execution e is the sequence consisting of *all* actions appearing in e. Since internal actions are also unobservable, we further define a *behavior* β to be the sequence consisting of all the *external* actions appearing in α. The set of all possible schedules of a IOTA is denoted by $scheds(\mathcal{A})$ whereas the set of all possible behaviors of a IOTA is denoted by $behs(\mathcal{A})$. Obviously, $behs(\mathcal{A})$ describes all the possible interactions that a IOTA $\mathcal{A}$ might be engaged in and thus should conform with the specifications of the system that the IOTA models.

In looking at executions of IOTAs, we will be often interested in only a subset of the actions. For this purpose, we define the projection operation "|". If e is an execution of a IOTA $\mathcal{A}$ and Π is a subset of the actions of $\mathcal{A}$, then the projection $e|\Pi$ consists of the steps in e that have actions from Π. The projection operation is similarily defined for schedules and behaviors. In particular, a IOTA behavior is obtained from its corresponding schedule by projecting it on the set of external actions.

2.4 Status and Status Succession

The *state* of a IOTA at any arbitrary point in time is not sufficient to predict its *future behavior*. To explain why this is true consider the example shown in Figure-2 where a IOTA, $\mathcal{A}$, enters

[8] The interval can be open, closed, or semi-closed (left-open or right open)



State:	s	s'
Enabled:	a	a
Intention:	a@[t_3,t_4]	a@[t_3,t_4]

Figure 2: The notion of status = state + intentions

state s at time t_1. Assume that $a \in actions(\mathcal{A})$ is an action that is enabled in s. Furthermore, assume that the time delay associated with that action is $\{\tau_l, \tau_u\}$. Now, assume that $\mathcal{A}$ changes its state to s' at time t_2 ($t_2 < t_1 + \tau_l$) such that a remains enabled. Obviously, knowing that $\mathcal{A}$ entered s' at time t_2 is not sufficient to predict that action a might fire anytime in the interval $\{t_1 + \tau_l, t_1 + \tau_u\}$. What we really need in order to completely capture the possible future behavior of a IOTA is, in addition to its state, its scheduled actions. This is encapsulated in our notion of *status*.

Definition 1 *The* status *of a IOTA is formally defined to be a tuple* (s, I)*, where:*

- $s \in states(\mathcal{A})$ *is a state of the IOTA* $\mathcal{A}$*, and*
- I *is a vector of* r *time intervals,* $r = |channels(\mathcal{A})|$*, each associated with one of* $\mathcal{A}$*'s channels. The vector* I *is called the* intentions *vector.*

In the status (s, I), the vector I of time intervals describes our *expectations* concerning channels that might fire in the future. If channel c is not enabled in state s then we do not expect it to change its value in the near-future, and thus we associate an empty interval ε with it ($I(c) = \varepsilon$). On the other hand, if the channel c was enabled since time t', where $t' \leq t$, and is constrained to fire within an interval of $\{\tau_l, \tau_u\}$, then we do expect it to happen in the interval $\{max(t' + \tau_l, t), t' + \tau_u\}$, where t is the time in which the state s was entered. Therefore, we associate that interval with the channel c. That is ($I(c) = \{max(t' + \tau_l, t), t' + \tau_u\}$). Finally, if the channel c is enabled and unconstrained[9] then we do expect it to fire at any point in the future,

[9]For example if it is an input channel.

and thus we associate with it the interval $\{t, \infty\}$ ($I(a) = \{t, \infty\}$), where t is the time at which state s was entered.

Associated with every start state s_0 of a IOTA $\mathcal{A}$ we define an *initial intention*, I_0, as follows:

$\forall c_i \in sig(\mathcal{A})$:
  if $c_i \in in(sig(\mathcal{A}))$ then
    $I_0(c_i) = [switch_{\mathcal{A}}(c_i), \infty]$
  else
    if $s_0 \in S_t^{c_i}$ then
      $I_0(c_i) = \{t_l^{c_i}, t_u^{c_i}\}$
    else
      $I_0(c_i) = \varepsilon$

where $S_t^{c_i} \overset{\{t_l^{c_i}, t_u^{c_i}\}}{\rightsquigarrow} (S_f^{c_i}, range_{\mathcal{A}}(c_i))$ is the timing constraint associated with a local channel c_i. Notice that in the above definition, we have assumed that a IOTA starts at time 0. The tuple (s_0, I_0), where $s_0 \in start(\mathcal{A})$ and I_0 is defined as shown above, is called an initial status.

Now, we are prepared to define the notion of *status succession*. Assume that the status (s, I) of a IOTA, $\mathcal{A}$, was entered at time t (with $t = 0$ for initial status). Furthermore, assume that it remained in the same status until time t' ($t' \geq t$), where it entered another status (s', I') as a result of firing an action $\pi \in range_{\mathcal{A}}(c_j)$, where $c_j \in sig(\mathcal{A})$. We say that the status (s', I') is a legal *successor* of the status (s, I) if and only if the following conditions hold:

1. **Safety:** For any channel $c_i \in sig(\mathcal{A})$, either
   (a) $I(c_i) = \varepsilon$, or
   (b) $t' \in I(c_i)$.
2. **Legality:** For the fired action $\pi \in range_{\mathcal{A}}(c_j)$, where $c_j \in sig(\mathcal{A})$, we have:
   (a) $(s, \pi, s') \in steps(\mathcal{A})$, and
   (b) $t' \in I(c_j)$.
3. **Continuity:** For any schannel $c_i \in sig(\mathcal{A})$, we have:
   (a) If c_i is not enabled in s', then $I'(c_i) = \varepsilon$.
   (b) If $I(c_i) = \varepsilon$ and c_i is enabled in s', then $I'(c_i) = \{t' + \tau_l, t' + \tau_u\}$, where $\{\tau_l, \tau_u\}$ is the timing constraint associated with c_i.
   (c) If $I(c_i) = \{t_1, t_2\}$ and c_i is enabled in s' and $c_i \neq c_j$ then $I'(c_i) = \{max(t', t_1), t_2\}$
   (d) If c_j remains enabled in s' then $I'(c_j) = \{t' + max(\tau_l, switch_{\mathcal{A}}(c_j)), t' + \tau_u\}$, where $\{\tau_l, \tau_u\}$ is the timing constraint associated with c_j.

The first condition guarantees that scheduled actions eventually happen in time. An action does not occur if another action *should* be scheduled first. In other words, the execution is *safe*

and no deadlines are missed. The second condition guarantees that the state transition from s to s' is defined as a *legal* move of $\mathcal{A}$. In other words, a ready-to-fire action exists which would cause this transition. The third condition guarantees the *continuity* of intentions. This means that the right intervals are used for the new status. In particular, empty intervals are associated with disabled signals, fresh intervals are associated with newly enabled signals, updated intervals are associated with signals that continue to be enabled.

We use the notation $(s, I) \stackrel{(\pi,t')}{\longmapsto} (s', I')$ to denote direct status succession due to the event (π, t'). Moreover, we use the notation $(s, I) \stackrel{*}{\longmapsto} (s', I')$ to denote zero or more successions and the notation $(s, I) \stackrel{+}{\longmapsto} (s', I')$ to denote one or more successions.

2.5 Timed Executions, Histories, Schedules and Behaviors

Consider the sequence of status successions starting at time $t = 0$ with an initial status (s_0, I_0):

$$(s_0, I_0) \stackrel{(\pi_1,t_1)}{\longmapsto} (s_1, I_1) \stackrel{(\pi_2,t_2)}{\longmapsto} (s_2, I_2) \ldots (s_{i-1}, I_{i-1}) \stackrel{(\pi_i,t_i)}{\longmapsto} (s_i, t_i) \ldots$$

The sequence consisting of alternating statuses and events from a status succession is called an *execution history*. The execution history for the status succession given above is:

$$(s_0, I_0), (\pi_1, t_1), (s_1, I_1), (\pi_2, t_2), (s_2, I_2), \ldots (s_{i-1}, I_{i-1}), (\pi_i, t_i), (s_i, I_i), \ldots$$

Following the notation of [Tuttle:88], we will call the sequence $t_0, t_1, t_2, \ldots, t_i, \ldots$ a *timing* of the execution history given above. A timing is thus a nondecreasing mapping $\mathcal{T}$ from a non-empty prefix of nonnegative integers – namely the indices of the actions/states – to the set of nonnegative reals. For example, $t_i = \mathcal{T}(i)$ is the time at which the action π_i was taken and state s_i entered. Notice that the mapping $\mathcal{T}$ is generally a many-to-one mapping, thus, allowing several actions to be taken at the same point in time.[10] This is necessary to be able to describe parallel behaviors.

For an execution e, the tuple $(e, \mathcal{T})$ defines a *timed execution*. The timed execution corresponding to the execution history given above is:

$$s_0, (\pi_1, t_1), s_1, (\pi_2, t_2), s_2, \ldots s_{i-1}, (\pi_i, t_i), s_i, \ldots$$

Similarily, for a schedule h, the tuple $(h, \mathcal{T})$ defines a *timed schedule*. The timed schedule corresponding to the execution history given above is:

$$(\pi_1, t_1), (\pi_2, t_2), \ldots, (\pi_i, t_i), \ldots$$

Finally, for a behavior β, the tuple $(\beta, \mathcal{T})$ defines a *timed behavior*. A timed behavior can be obtained from a timed schedule using the projection:

$$((\pi_1, t_1), (\pi_2, t_2), \ldots, (\pi_i, t_i), \ldots) | ext(sig(\mathcal{A}))$$

We denote the set of all possible execution histories, timed executions, timed schedules and timed behaviors of a IOTA $\mathcal{A}$ by $hists(\mathcal{A})$, $t.execs(\mathcal{A})$, $t.scheds(\mathcal{A})$ and $t.behs(\mathcal{A})$, respectively.

[10] In which case their indices will map to the same value using $\mathcal{T}$.

3 The IOTA model: Definitions and Properties

We are now prepared to formally define a IOTA. A IOTA $\mathcal{A}$ is a septuple:

$$(sig(\mathcal{A}), range_{\mathcal{A}}(), switch_{\mathcal{A}}(), states(\mathcal{A}), start(\mathcal{A}), steps(\mathcal{A}), delays(\mathcal{A}))$$

where:

- $sig(\mathcal{A})$ is the signature of $\mathcal{A}$.
- $range_{\mathcal{A}}()$ maps every channel in $sig(\mathcal{A})$ to the possible set of actions it might signal.
- $switch_{\mathcal{A}}()$ maps every channel in $sig(\mathcal{A})$ to its minimum switching time.
- $states(\mathcal{A})$ is the set of states of $\mathcal{A}$.
- $start(\mathcal{A}) \subseteq states(\mathcal{A})$ is the set of starting states of $\mathcal{A}$.
- $steps(\mathcal{A})$ is the set of possible steps of $\mathcal{A}$.
- $delays(\mathcal{A})$ is the set of time constraints of $\mathcal{A}$.

3.1 Composing IOTAs

A basic aspect of the IOTA model is its capability to model a complex system by composing simpler system components together. We follow an approach similar to that proposed in [Lynch:88a] in that we require the signatures of the composed IOTAs to satisfy a *strong compatability condition.*

Definition 2 *A countable collection of IOTAs, $\{\mathcal{A}_i\}_{i \in I}$, is said to be strongly compatible if for all $i, j \in I, i \neq j$, we have:*

1. $out(sig(\mathcal{A}_i)) \cap out(sig(\mathcal{A}_j)) = \phi$,
2. $int(sig(\mathcal{A}_i)) \cap signals(\mathcal{A}_j) = \phi$,
3. *No channels appear in infinitely many signatures.*
4. *If $c \in out(sig(\mathcal{A}_i)) \cap in(sig(\mathcal{A}_i))$, then:*
   (a) $range_{\mathcal{A}_i}(c) \subseteq range_{\mathcal{A}_j}(c)$, *and*
   (b) $switch_{\mathcal{A}_i}(c) \geq switch_{\mathcal{A}_j}(c)$.

In the above definition, the first condition guarantess that no two output channels feed the same input channel. The second condition guarantees that internal channels are unobservable by the environment. The third condition restricts the fan-out of IOTAs to a finite number. These three conditions are identical to those used with the IOA model [Lynch:88a]. The last condition,

however, requires some explanation. Obviously, an output channel c from $\mathcal{A}_i$ can be used to feed an input channel of $\mathcal{A}_j$ only if the actions that might be signaled on c are within $\mathcal{A}_j$'s range of expectations. Moreover, it should be within $\mathcal{A}_i$'s capacity to handle the stream of events generated by $\mathcal{A}_i$.

Definition 3 *The composition $\mathcal{A} = \prod_{i \in I} \mathcal{A}_i$ of a strongly compatible collection of IOTAs, $\{\mathcal{A}_i\}_{i \in I}$, is the IOTA defined as follows:*

- *The signature, $sig(\mathcal{A})$, is given by:*
  - $in(sig(\mathcal{A})) = \bigcup_{i \in I} in(sig(\mathcal{A}_i)) - \bigcup_{i \in I} out(sig(\mathcal{A}_i))$.
  - $out(sig(\mathcal{A})) = \bigcup_{i \in I} out(sig(\mathcal{A}_i))$.
  - $int(sig(\mathcal{A})) = \bigcup_{i \in I} int(sig(\mathcal{A}_i))$.
- *The signaling range function, $range_{\mathcal{A}}()$, is defined as follows:*
  - *If $c \in in(sig(\mathcal{A}))$ then $range_{\mathcal{A}}(c) = \bigcap_{c \in in(sig(\mathcal{A}_i))} range_{\mathcal{A}_i}(c)$*
  - *If $c \in loc(sig(\mathcal{A}_i)), i \in I$ then $range_{\mathcal{A}}(c) = range_{\mathcal{A}_i}(c)$.*
- *The switching time function, $switch_{\mathcal{A}}()$, is defined as follows:*
  - *If $c \in in(sig(\mathcal{A}))$ then $switch_{\mathcal{A}}(c) = \max_{c \in in(sig(\mathcal{A}_i))} switch_{\mathcal{A}_i}(c)$*
  - *If $c \in loc(sig(\mathcal{A}_i)), i \in I$ then $switch_{\mathcal{A}}(c) = switch_{\mathcal{A}_i}(c)$.*
- $states(\mathcal{A}) = \prod_{i \in I} states(\mathcal{A}_i)$.
- $start(\mathcal{A}) = \prod_{i \in I} start(\mathcal{A}_i)$.
- $steps(\mathcal{A}) = \{(\vec{s_1}, \pi, \vec{s_2}) : \forall i \in I$, if $\pi \in actions(\mathcal{A}_i)$ then $(\vec{s_1}[i], \pi, \vec{s_2}[i]) \in steps(\mathcal{A}_i)$, and if $\pi \notin actions(\mathcal{A}_i)$ then $\vec{s_1}[i] = \vec{s_2}[i]\}$.
- $delays(\mathcal{A}) = \bigcup_{i \in I} \{\vec{S}_t^c \overset{\{\tau_l^c, \tau_u^c\}}{\rightsquigarrow} (\vec{S}_f^c, range_{\mathcal{A}_i}(c)) : S_f^c \overset{\{\tau_l^c, \tau_u^c\}}{\rightsquigarrow} (S_f^c, range_{\mathcal{A}_i}(c)) \in delays(\mathcal{A}_i)$;
  $\vec{S}_t^c = (\prod_{j<i} states(\mathcal{A}_j)) \times S_t^c \times (\prod_{j>i} states(\mathcal{A}_j))$;
  $\vec{S}_f^c = (\prod_{j<i} states(\mathcal{A}_j)) \times S_f^c \times (\prod_{j>i} states(\mathcal{A}_j))\}$

3.2 IOTAs as IOAs

In this section, we show how to build an equivalent IOA for a given IOTA. This construction allows us to use most of the proof techniques developed for the IOA model [Lynch:89c] with our IOTA model. The basic idea is to *encode* all the timing information of a IOTA into the states and steps of an equivalent IOA. We do so, by having the set of actions of the IOA be the set of all possible events of the IOTA; the set of states of the IOA be the set of all possible statuses of the IOTA; the set of start states of the IOA be the set of all possible start statuses of the IOTA; the set of steps of the IOA be the set of all possible status successions of the IOTA; and finally, the partition of the IOA be the trivial partition that puts all the actions of each channel in a different

class. We do so, formally, in the following definition.

Definition 4 *Given a IOTA $\mathcal{A}$, where:*

$$\mathcal{A} = (sig(\mathcal{A}), range_{\mathcal{A}}(), switch_{\mathcal{A}}(), states(\mathcal{A}), start(\mathcal{A}), steps(\mathcal{A}), delays(\mathcal{A}))$$

we define $\hat{\mathcal{A}}$, the IOA-equivalent of $\mathcal{A}$, to be:

$$\hat{\mathcal{A}} = (sig(\hat{\mathcal{A}}), states(\hat{\mathcal{A}}), start(\hat{\mathcal{A}}), steps(\hat{\mathcal{A}}), part(\hat{\mathcal{A}}))$$

where:

- $in(sig(\hat{\mathcal{A}})) = range_{\mathcal{A}}(in(sig(\mathcal{A}))) \times \Re$
- $out(sig(\hat{\mathcal{A}})) = range_{\mathcal{A}}(out(sig(\mathcal{A}))) \times \Re$
- $int(sig(\hat{\mathcal{A}})) = range_{\mathcal{A}}(int(sig(\mathcal{A}))) \times \Re$
- $states(\hat{\mathcal{A}}) = \{(s, I(c_i)) : s \in states(\mathcal{A}), I(c_i) \in \Re \times \Re, c_i \in sig(\mathcal{A})\}$
- $start(\hat{\mathcal{A}}) = \{(s_0, I_0) : s_0 \in start(\mathcal{A}); I_0$ is the initial intension associated with $s_0\}$
- $steps(\hat{\mathcal{A}}) = \{((s, I), (\pi, t'), (s', I')) : (s, I) \stackrel{(\pi,t')}{\longmapsto} (s', I')$ is a status succession of $\mathcal{A}\}$
- $part(\hat{\mathcal{A}}) = \{\{c_i\} : c_i \in sig(\mathcal{A})\}$

Theorem 1 *If $\hat{\mathcal{A}}$ is the IOA-equivalent of a IOTA $\mathcal{A}$, then:*

$$hists(\mathcal{A}) = execs(\hat{\mathcal{A}}), t.scheds(\mathcal{A}) = scheds(\hat{\mathcal{A}}), \text{ and } t.behs(\mathcal{A}) = behs(\hat{\mathcal{A}})$$

Proof: We need only to prove that $hists(\mathcal{A}) = execs(\hat{\mathcal{A}})$.[11] That is, we have to show that $e \in t.hists(\mathcal{A}) \Leftrightarrow e \in execs(\hat{\mathcal{A}})$. We do so by proving two lemmas. The first states that any prefix of an execution history of $\mathcal{A}$ exists as a prefix of an execution of $\hat{\mathcal{A}}$. The second, shows that if an execution history e of $\mathcal{A}$ is finite then it is an execution of $\hat{\mathcal{A}}$. These two results are sufficient to show that any finite or infinite execution history $e \in t.hists(\mathcal{A})$ is also an execution $e \in execs(\hat{\mathcal{A}})$. ■

Lemma 1 *If $\hat{\mathcal{A}}$ is the IOA-equivalent of a IOTA $\mathcal{A}$, then any prefix of an execution history $e \in hists(\mathcal{A})$ exists as a prefix of an execution in $execs(\hat{\mathcal{A}})$.*

Proof: The proof is by induction on l the length (number of events) of e.

- *Base* $(l = 0)$
  - Any prefix of length 0 in $hists(\mathcal{A})$ consists of just an initial status (s_0, I_0). By definition, $(s_0, I_0) \in start(\hat{\mathcal{A}})$. Hence, it is the prefix of at least an execution in $execs(\hat{\mathcal{A}})$.

[11] The proof for schedules and behaviors follows directly from the proof for executions.

- *Induction* ($l > 0$)
  - Assume that all prefixes of length $l-1$ in $hists(\mathcal{A})$ also exist in $execs(\hat{\mathcal{A}})$.
  - Consider any prefix e_l of length l in $hists(\mathcal{A})$. Obviously, $e_l = e_{l-1} \stackrel{(\pi_l, t_l)}{\longmapsto} (s_l, I_l)$, where e_{l-1} is some prefix of length $l-1$ in $hists(\mathcal{A})$.
  - From the induction assumption, we have $e_{l-1} \in execs(\hat{\mathcal{A}})$. (1)
  - Assume that (s_{l-1}, I_{l-1}) is the last status of e_{l-1}. It follows that the l^{th} status transition in e_l is $(s_{l-1}, I_{l-1}) \stackrel{(\pi_l, t_l)}{\longmapsto} (s_l, I_l)$. From the definition of $\hat{\mathcal{A}}$ above, we have $((s_{l-1}, I_{l-1}), (\pi_l, t_l), (s_l, I_l)) \in steps(\hat{\mathcal{A}})$. (2)
  - From (1) and (2), $e_l \in execs(\hat{\mathcal{A}})$. This proves the induction and the lemma.

■

Lemma 2 *If $\hat{\mathcal{A}}$ is the IOA-equivalent of a IOTA $\mathcal{A}$, then any finite execution history e of $\mathcal{A}$ is also a finite execution of $\hat{\mathcal{A}}$.*

Proof:

- Let e be a finite execution history of $\mathcal{A}$. From Lemma 1, we know that e exists as the prefix of some execution e' of $\hat{\mathcal{A}}$. All we need to show is that $e' = e$.
- Let (s, I) be the last status in e. In s, no channels of $\mathcal{A}$ are enabled[12]. From the definition of $\hat{\mathcal{A}}$, there can't be any actions out of (s, I). Thus, (s, I) is also the last status of e'. Thus $e = e'$.

■

3.3 IOTA implementation

A IOTA $\mathcal{A}$ is said to implement another IOTA $\mathcal{B}$, if and only if $t.behs(\mathcal{A}) \subseteq t.behs(\mathcal{B})$, that is, if any timed behavior of $\mathcal{A}$ can also be produced by $\mathcal{B}$. The reverse, however, is not true: there might exist timed behaviors of $\mathcal{B}$ that are not timed behaviors of $\mathcal{A}$. The notion of an IOTA implementing another will be used mainly in verification. The idea is to prove that a given IOTA implements a second, that the second implements the third, and so on until the final IOTA is shown to implement the required specifications. The transitivity of the implementation relation guarantees that the first IOTA indeed implements the specifications.

Lemma 3 *The implementation relation is transitive.*

Proof: For IOTAs $\mathcal{A}$, $\mathcal{B}$, and $\mathcal{C}$, assume that $\mathcal{A}$ implements $\mathcal{B}$ and $\mathcal{B}$ implements $\mathcal{C}$. Let U, V and W denote the set of all timed behaviors of $\mathcal{A}$, $\mathcal{B}$ and $\mathcal{C}$ respectively. Since $\mathcal{A}$ implements $\mathcal{B}$ and $\mathcal{B}$ implements $\mathcal{C}$, we have $U \subseteq V \subseteq W$. Thus $U \subseteq W$ and $\mathcal{A}$ implements $\mathcal{C}$. ■

[12] or otherwise s wouldn't have been the last state of e

Lemma 4 *A necessary condition for a IOTA $\mathcal{A}$ to weakly implement another IOTA $\mathcal{B}$ is that any external actions of $\mathcal{A}$ be also external actions of $\mathcal{B}$. That is $ext(sig(\mathcal{A})) \subseteq ext(sig(\mathcal{B}))$*

Proof: Immediate from the definition. ■

Note that the above definition allows a weak implementation not to have the same set of inputs as the IOTA it is implementing. That is, a weak implementation might ignore some of the inputs of the original IOTA. This, of course, leads to a trivial weak implementation of any IOTA which is the *nil* IOTA that doesn't have any input or output actions. Obviously the *nil* IOTA is a weak implementation of any possible IOTA. To disallow trivial implementations, we define strong implementations. A strong implementation (or just an implementation) of a IOTA does not ignore any of the external inputs of that IOTA.

Lemma 5 *A necessary condition for a IOTA $\mathcal{A}$ to strongly implement another IOTA $\mathcal{B}$ is that $ext(sig(\mathcal{A})) \subseteq ext(sig(\mathcal{B}))$, and $in(sig(\mathcal{B})) = in(sig(\mathcal{A}))$.*

Proof: Immediate from the definition. ■

Notice that the above conditions do not require the implementing IOTA to have all of the external output channels of the implemented IOTA. This is so because, in general, it might be the case that signaling on one (or more) of those channels can be avoided. That is, the implementing IOTA might elect to always produce behaviors that do not include actions signaled on those channels.[13] In most of the interesting cases, however, the implementing IOTA will not be able to discard any of the output actions and thus will have an external signature identical to that of the implemented IOTA.[14]

In the remaining of this section, we derive a set of sufficient conditions for strong implementation of a IOTA by another. The idea is to come up with a mapping between the statuses of the two IOTAs and show that any possible status succession in the implementing IOTA correspond to some possible succession in the implemented IOTA. This approach is similar to the possibility mapping proposed in [Lynch:88a], [Lynch:88b], except that it is complicated here by the need to preserve the timing constraints of the implemented IOTA. The following theorem establishes the required sufficient conditions.

Theorem 2 *A set of sufficient conditions for a IOTA $\mathcal{A}$ to strongly implement another IOTA $\mathcal{B}$ is that:*

1. *$ext(sig(\mathcal{A})) = ext(sig(\mathcal{B}))$, and*
2. *There exists a mapping Γ from $statuses(\mathcal{A})$ to the set $2^{statuses(\mathcal{B})}$ (the power set of $statuses(\mathcal{B})$), such that both of the following conditions hold:*
   (a) *For every initial status (s_0, I_0) of $\mathcal{A}$ there is a an initial status (r_0, J_0) of $\mathcal{B}$ such that $(r_0, J_0) \in \Gamma((s_0, I_0))$.*

[13] If such behaviors are allowed by the specification.

[14] In the remainder of this paper, we will assume that this is indeed the case.

(b) If (s, I) is a reachable status of $\mathcal{A}$, $(r, J) \in \Gamma((s, I))$ is a reachable status of $\mathcal{B}$, and $(s, I) \stackrel{(\pi,t')}{\longmapsto} (s', I')$ is a possible status succession of $\mathcal{A}$, then there exists a timed schedule (h, T) that takes $\mathcal{B}$ from status (r, J) to status (r', J'), where:

i. $(r', J') \in \Gamma((s', I'))$, and

ii. $(h, T)|ext(\mathcal{B}) = (\pi, t')|ext(\mathcal{A})$.

Proof: The first condition guarantees that the necessary conditions of Lemma5 are met. To prove the theorem, we need to show that the satisfaction of the second condition is sufficient to provide a strong implementation. Basically, we have to show that it guarantees that any possible external timed behavior of $\mathcal{A}$ can be generated by $\mathcal{B}$. We do this as follows:

- Let $\hat{\mathcal{A}}$ and $\hat{\mathcal{B}}$ be the IOA-equivalents of IOTAs $\mathcal{A}$ and $\mathcal{B}$ respectively.
- Since $ext(sig(\mathcal{A})) = ext(sig(\mathcal{B}))$, it follows from the construction of $\hat{\mathcal{A}}$ and $\hat{\mathcal{B}}$ that $ext(sig(\hat{\mathcal{A}})) = ext(sig(\hat{\mathcal{B}}))$. (1)
- The mapping Γ is a possibilities mapping from the set $states(\hat{\mathcal{A}})$ to the set $2^{states(\hat{\mathcal{B}})}$ (the power set of states of $\hat{\mathcal{B}}$). (2)
- From (1) and (2) above, and using Proposition-12 from [Lynch:88a], we get that $\hat{\mathcal{A}}$ implements $\hat{\mathcal{B}}$ implements. Thus, $execs(\hat{\mathcal{A}}) \subseteq execs(\hat{\mathcal{A}})$. (3)
- From Theorem 1, we get $execs(\hat{\mathcal{A}}) = hists(\mathcal{A})$ and $execs(\hat{\mathcal{B}}) = hists(\mathcal{B})$. (4)
- From (3) and (4) above, we have: $hists(\mathcal{A}) \subseteq hists(\mathcal{B})$. Thus, $\mathcal{A}$ implements $\mathcal{B}$.

■

3.4 IOTA equivalence

Two IOTAs are equivalent if there is no way of identifying one from the other just by comparing their behaviors.

Theorem 3 *A IOTA $\mathcal{A}$ is equivalent to another IOTA $\mathcal{B}$ if and only if: $\mathcal{A}$ implements $\mathcal{B}$, and $\mathcal{B}$ implements $\mathcal{A}$.*

Proof: Both the *if* and the *only if* parts can be proved by contradiction:

- Let $\mathcal{A}$ be equivalent to $\mathcal{B}$ and assume that $\mathcal{A}$ does not implement $\mathcal{B}$.[15] It follows that there exists at least one timed behavior of $\mathcal{A}$ that is not a timed behavior of $\mathcal{B}$. Using this behavior, the IOTAs $\mathcal{A}$ and $\mathcal{B}$ can be identified. Hence, they are not equivalent – a contradiction.

[15] The case in which $\mathcal{B}$ does not immplement $\mathcal{A}$ is symmetric.

- Let $\mathcal{A}$ be an implementation of $\mathcal{B}$ and $\mathcal{B}$ be an implementation of $\mathcal{A}$ and assume that $\mathcal{A}$ and $\mathcal{B}$ are not equivalent. It follows that there exists a timed behavior of $\mathcal{A}$ ($\mathcal{B}$) that is not a timed behavior of $\mathcal{B}$ ($\mathcal{A}$). Hence $\mathcal{A}$ ($\mathcal{B}$) does not implement $\mathcal{B}$ ($\mathcal{A}$) – a contradiction.

■

4 Conclusion

The current practice in building real-time embedded systems is not based on any sound scientific approach [Stankovic:88a]. In view of the increasing complexity, cost and criticality of these systems, it has become evident that a new methodology should be adopted in their design and implementation. In this paper, we proposed a unified framework for specifying and verifying real-time digital systems. The framework we suggest is based on the IOTA model which is an extension to the Input-Output Automata model proposed in [Lynch:88a] to study discrete event systems.

A IOTA is an abstraction that encapsulates a system task. An real-time digital system is viewed as a set of interacting IOTAs. IOTAs communicate with each other and with the external environment using *signals*. A signal carries a sequence of *events*, where an event represents an instantiation of an *action* at a specific point in time. Actions can be generated by either the environment or the IOTAs. Each IOTA has a *state*. The state of a IOTA is observable and can only be changed by local *computations*. Computations are triggered by actions and have to be scheduled to meet specific timing constraints. IOTAs can be *composed* together to form higher level IOTAs. A specification of a IOTA is a description of its behavior (*i.e.* how it reacts to stimuli from the environment). A IOTA is said to *implement* another IOTA, if it is impossible to differentiate between their external behaviors. This is the primary tool that is used to verify that an implementation meets the required specification.

In [Bestavros:90a], and based on the IOTA model, we proposed ESPRIT, a language and environment for the (E)xecutable (S)pecification of (P)arallel (R)eal-time (I)nteractive (T)asks. In ESPRIT, a system is specified as an interconnection of IOTA objects. Each IOTA object has a set of *state variables* and a set of *channels.* State transitions are triggered by firing *actions* on the IOTA's channels. Local time-constrained actions are scheduled autonomously, whereas input actions are enforced by the environment. When an action fires, the *method* associated with it is applied, resulting in a state transition. IOTA objects can be specified in a number of different ways, including inheritance, augmentation and composition. The instantiation (creation) of a IOTA artifact involves, determining its parameters, binding its channels, and initializing its state variables.

We have implemented a compiler for specifications written in the ESPRIT language. Once compiled, such specifications can be executed in simulated time. In [Bestavros:90b], the specification and simulated behavior of Buggy, an elaborate cockroach-like robot, were presented. The strength of the IOTA model (preserved in ESPRIT) was demonstrated by carrying out proofs on Buggy's specification (stability and timing properties) and by comparing it to other behavioral specification methodologies.

As we deepen our understanding of the issues involved in the specification and implementation of parallel real-time systems, we have come to realize the need for some extensions to our

work. As it stands, the IOTA model, as well as all similar models, allow time constraints to be specified for individual actions. It is often the case, however, that *structured* sequences of actions need to be timed. We are working now on a number of possible modifications of the IOTA model and ESPRIT language to support structured time constraints. Using the IOTA model, the only valid executions of a given system specification are those where time constraints are always satisfied. To obtain real implementations, enough "cycles" have to exist to guarantee that condition. Given a specific hardware configuration, an interesting question is whether the specifications are *realizable* or not. We have already started to use the IOTA model and the ESPRIT environment in the specification and implementation of robotics applications. It would be both interesting and challenging to apply our methodology to other applications. In this respect, we are interested in two quite different applications: real-time transaction management systems and massively parallel computations.

References

[BBN:86] BBN Laboratories Incorporated and Harvard Robotics Laboratory, "Robotic task control", *Proposal No. P88-CISD-109*, March 1988.

[Bestavros:90a] "ESPRIT: Executable Specification of Parallel Real-time Interactive Tasks." *Technical Report TR-06-90, Department of Computer Science, Harvard University*, March 1990.

[Bestavros:90b] Azer Bestavros, James Clark and Nicola Ferrier, "Management of sensori-motor activity in mobile robots." *Proceedings of the 1990 IEEE Robotics and Automation conference, Cincinati, Ohio*, Feb. 1990.

[Bestavros:89a] Azer Bestavros, "A new environment for developing real-time applications based on the IOTA model", *Internal Report – Department of Computer Science, Harvard University*, May 1989.

[Bestavros:89b] Azer Bestavros, "The Input Output Timed Automaton: A model for real-time parallel computation" *Technical Report, TR-12-89, Department of Computer Science, Harvard University*, May 1989.

[Bestavros:88] Azer Bestavros, "The Michael - Merlin Connection: Programming tools for the remote control of the American Cimflex robot", *Technical Report – Robotics Laboratory, Harvard University*, September 1988.

[Brockett:88] Roger Brockett, "On the Computer Control of Movement", *Proceedings of the 1988 IEEE Robotics and Automation Conference, Philadelphia.*

[Brooks:86] Rodney Brooks and Jonathan Connell, "Asynchronous Distributed Control System for a Mobile Robot", *SPIE Proceedings*, Vol. 727, October 1986.

[Clarke:83] E. Clarke, E. Emerson, and A. Sistla, "Automatic verification of finite-state concurrent systems using temporal logic specifications: A practical approach", *Tenth ACM Symposium on Principles of Programming Languages*, Austin, Texas, January 1983.

[Hoare:78] C. A. R. Hoare, "Communicating sequential processes", *Communication of the ACM*, Vol. 21, August 1978.

[Joseph:88] Mathai Joseph and Asis Goswami, "Formal Description of Real-Time Systems: A review", *Research Report 129 – Department of Computer Science, University of Warwick*, April 1988.

[Lewis:89] Harry Lewis, "Finite-state analysis of asynchronous circuits with bounded temporal uncertainty", *Technical report, TR-15-89, Department of computer science, Harvard University*, June 1989 (revised July 1989).

[Lewis:90] Harry Lewis, "A logic of concrete time intervals", *Proceedings of LICS'90, the fifth annual symposium on Logic In Computer Science, Philadelphia, PA*, June 1990.

[Lynch:88a] Nancy Lynch and Mark Tuttle, "An Introduction to Input/Output Automata", *Technical Report MIT/LCS/TM-373*, November 1988.

[Lynch:88b] Nancy Lynch, "The I/O Automata" *6.852 Distributed Algorithms Lecture Notes – Laboratory of Computer Science, MIT*, Fall 1988.

[Lynch:89a] Nancy Lynch and Haggit Attiya, "Time bounds for real-time process control in the presence of timing uncertainty", *Unpublished notes – LCS/MIT*, January 1989.

[Lynch:89c] Nancy Lynch and Haggit Attiya, "Assertional proofs for timing properties", *Preliminary report – LCS/MIT*, August 1989.

[Lynch:89b] N. Lynch, M. Merritt, W. Weihl, and A. Fekete, "Atomic Transaction", *In publication*, January 1989.

[Mishra:83] B. Mishra and E. Clarke, "Automatic and hierarchical verification of asynchronous circuits using temporal logic", *Technical Report CMU-CS-83-155*, September 1983.

[Schneider:88] Fred B. Schneider, "Critical (of) issues in real-time systems", *Technical Report 88-914 (Position Paper) – Department of Computer Science, Cornell University*, May 1988.

[Stankovic:88a] John A. Stankovic, "Real-Time computing systems: The next generation", *COINS Technical Report 88-06 – University of Massachusetts Amherst*, January 88.

[Stankovic:88a] John A. Stankovic, "Misconceptions about real-time computing", *IEEE Computer*, October 1988.

[Tewilliger:87b] Robert B. Tewilliger "PLEASE: Executable specification for incremental software development", *Report No. UIUCDCS-R-86-1295– Dept. of Computer Science, University of Illinois at Urbana-Champaign*, June 1987.

[Tewilliger:89b] Robert B. Tewilliger, Mark J. Maybee and Leon J. Osterweil, "An example of formal specification as an aid to design and development", *Proceedings of the fifth International Workshop on Software Specification and Design*, May 1989.

[Tuttle:88] Mark Tuttle, Michael Meritt and Francesmary Modugno, "Time Constrained Automata", *Unpublished Notes*, November 1988.

[Veen:86] A. Veen, "Dataflow Machine Architecture", *ACM Computing Surveys,* Vol. 18, No. 4, December 1986.

[Zave:81] Pamela Zave and Raymond Yeh, "Executable requirements for embedded systems", *Proceedings of the fifth conference on Software Engineering*, San Diego, California, March 1981.

[Zave:82] Pamela Zave, "An operational approach to requirements specification for embedded systems", *IEEE Transactions on Software Engineering*, Vol. 8, No. 3, May 1982.

[Zave:84a] Pamela Zave, "The operational versus the conventional approach to software development", *Communications of the ACM*, Vol. 27, No. 2, February 1984.

[Zave:86] Pamela Zave, "Salient features of an executable specification language and its environment", *IEEE Transactions on Software Engineering*, Vol. 12, No. 2, February 1986.