In Proceedings of AIRTC'91: The 3rd IFAC Workshop on Artificial Intelligence in Real Time Control, Sonoma, CA, September 1991

Planning for Embedded Systems: A Real-Time prospective

Azer Bestavros

Department of Computer Science
Boston University
bestavros@cs.bu.edu

September 1991

Abstract: We investigate the problem of planning for embedded systems where issues of safety, liveness, and responsiveness are much more important than intelligence. We argue that in such systems, a planning agent should produce a *behavioral specification* that, when *superimposed* on running behaviors, preserves the properties critical to the mission of the system. In this respect, we propose the Time-constrained Reactive Automata (TRA) formalism [Best91b] for plan generation and verification.

Keywords: Embedded systems; Artificial intelligence; Automata theory; Modeling; Specification languages; Planning; TRA.

1 Introduction

Embedded computing systems are characterized by the rigidity of their performance and reliability requirements, which are dictated by the critical nature of their missions and the demanding and often hostile environments with which they interact.

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 "unintelligent" environment. Examples of such systems include ballistic-missile-defense systems, command and control systems, nuclear reactors, industrial process-control plants, robotics, space shuttle and aircraft avionics, aircraft collision avoidance systems, automotive control, switching circuits and telephony systems, data-acquisition systems, and real-time databases, just to name a few. The leaping advances in computing technologies witnessed in the last few decades have resulted in an explosion in the extent and variety of such systems. This trend is only likely to continue in the future.

Viewed simply, any embedded system has two parts: an *external interface* and a *programmed system*. The external interface consists of a number of devices such as sensors and actuators that interact with the environment. The programmed system collects information from the sensors and responds by producing actions to drive the actuators. The sustained demands of the external environment pose relatively rigid and urgent requirements on the performance of the programmed system. These requirements are usually stated as constraints on the real-time behavior of the system. Wirth [Wirt77] singled out this processing-time dependency as the one aspect that differentiates embedded systems from other sequential and parallel systems.

The complexity of embedded systems coupled with the critical nature of their missions makes the importance of their safety, liveness and responsiveness unquestionable. On the other hand, the dynamic, unpredictable, and often hostile nature of their external environments dictates the incorporation of some level of intelligence to cope (*react*) with environmental changes or even to adopt new behaviors (*plan*) to insure missions success. Research in the real-time aspects of embedded systems has focussed on issues of specification, verification, analysis, languages, models and semantics [Stan88], whereas research in the intelligence aspects of these systems was geared towards planning, learning, knowledge representation and logic.

With research in real-time and intelligence aspects of embedded systems drifting apart, it has become evident that some kind of "common ground" is needed to make embedded systems both *intelligent* and *reallistic*. Recently several attempts were made to achieve that goal. Most of these attempts were initiated by the AI research community in an effort to make planning applicable to embedded systems. Among these we cite: Brook's subsumption architecture [Broo86], Maes' situated agents [Maes90], Nilsson's action networks [Nils88], Lyons' $\mathcal{RS}$ process algebra [Lyon90c, Lyon90a], Allens' formal temporal logic [Alle86, Pela88], and Rosenschein's situated-automata [Rose85]. One problem with most of these attempts is the way planning for embedded systems is perceived. Planners are considered an integral part of the tight (continuous) loop of sensing and acting on the environment. This led to the notion of reactive planning [Maes89], [Lyon91a] where plans are described as a selection algorithm between precompiled (or hardwired) behaviors based on the state of the environment. Although closer to reality than traditional planners, reactive planners fail in one or both of the following aspects. First, they do not allow for *derived* (creative or learned) behaviors. Sec-

ond, they deal with time as an afterthought making the verification of safety conditions (in the form of timing properties) much harder. As we mentioned earlier, this latter deficiency is unacceptable. In a way, customers would prefer an unintelligent safe system over an intelligent potentially unsafe one!

2 Levels of Reactive Control

We single out four levels of real-time control *Servo*, *Selective*, *Teleo-selective*, and *Intelligent*. One common aspect shared between these types of control is that, at any given point in time, a unique *reactive* behavior – certified to preserve the safety, liveness and responsiveness of the embedded system – is followed. This notion of continuous reactiveness is crucial to embedded systems [Nils90].

A *servo control* system consists usually of a *unique behavior* that uses feedback to guarantee a tight coupling between its input and output signals. Designers of servo systems are often concerned with questions of stability, transient and steady state responses[1], compensation, ... *etc.* Power steering is an example of a servo control system. It has one behavior, to keep the steering wheel (input) and the front wheels (output) tightly coupled.

Using *selective control*, a system's behavior is selected from amongst a fixed number of competing behaviors based on stimuli from the external environment (world) in order to achieve a *unique goal*. Temperature control is an example of a reactive control system where either a cooling or heating behavior is selected depending on the ambient temperature. The cooling/heating behaviors might themselves be servo controlled. Examples of reactive control systems include Brooks' subsumption architecture and Brockett's Motion Description Language (MDL) [Broc88].

Using *teleo-selective* control, a system's behavior is selected from amongst a fixed number of competing behaviors based on stimuli from the environment and motivations to achieve one of a *set of pre-determined goals*. Examples of teleo-selective systems include Nilsson's action networks [Nils88], Maes' situated agents [Maes90], and our earlier work using Input Output Timed Automata [Best90a, Best90c, Best90b].

In *intelligent* control, a system's behavior is selected from amongst a number of fixed and superimposed *synthesized behaviors* based on stimuli from the environment and motivations to achieve a run-time goal. The process of synthesizing a behavior is carried out by a planning agent *outside* the sensing/acting loop based on a *perceived* model of the world and a set of goals to be achieved. Figure-1 shows the architecture of an intelligent control system.

The success of the planning process, which entails a safe progress towards achieving the planner's goals, depends heavily on the existence of an accurate description of the behavior of the world. This can be achieved in two different ways. On the one hand, a cognition agent might be able, through observation and learning, to provide the planning agent with a *perceived* behavioral specification of the world. The potential inaccuracy of this process – due, for example, to a rapidly changing environment – dictates that perceived world specifications be used only in conjunction with the plan synthesis process. On the other hand, the verification process can rely only on invariants about the world's behaviors, which are asserted and guaranteed by the system designers. We call these assertions *hardcore specifications*.[2]

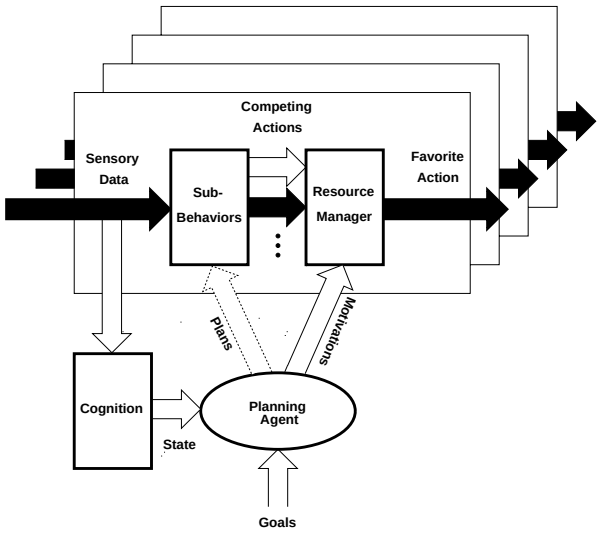

Figure 1: Intelligent control system architecture.

Most of the current research in real-time planning falls in the first three levels of control systems discussed earlier, where plans are pre-compiled (hardwired) into the sensing/acting loop. During run-time, the system is executing a pre-compiled plan. Recent attempts at building intelligent embedded systems [Kael86, Lyon91b, Lyon91a] are mostly concerned with the dynamic, unpredictable, and often hostile nature of the environment. The consideration of safety issues expressed as timing constraints has been minimal. This is primarily due to the lack of a computational model powerful enough to capture timing properties relevant to real-time specification and verification, and expressive enough to serve as a framework for planning. In the next section, we propose such a framework, namely the TRA model, to be used as a vehicle for planning in real-time systems. In particular, it is suitable as a formalism for the representation of actions as well as for verification purposes.

3 The TRA Model

For a complete treatment of the model, we refer the reader to [Best91a].

3.1 Novelties

The TRA model differs from others (like Hoare's

[1] For example overshoot, settling time, steady state errors.

[2] In embedded systems, it is usually assumed [Leve91] that an external mechanism is responsible for securing the safety of the system. Examples of such mechanisms include human intervention, using limit switches, ... *etc.*

CSP model [Hoar85], Reed and Roscoe's timed CSP model [Reed88], Arbib and Lyons' $\mathcal{RS}$ process-based model [Lyon89], and Baeten and Bergestra's real-time process algebra [Baet91]) 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. This property is crucial for accurate and realistic modeling of embedded and real-time systems. A sufficient condition to ensure that a model is reactive is the *input enabling* property proposed in [Lync88]. 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.

The TRA model is unique in that it admits the causal nature of physical processes. A 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 *cause*.

Spontaneity is a notion closely related to causality. 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 [Sree90]. 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.

3.2 Basic Definitions

An embedded system is viewed as a set of interacting automata called TRAs (Time-constrained Reactive Automata). TRAs communicate with each other through *channels* (see Figure-2). A channel is an abstraction for an *ideal* unidirectional communication. The information carried on a is called a *signal*, which consists of a sequence of *events*. An event underscores the instantiation of an *action* at a specific point in time. TRA channels have finite capacities, and the signals they carry are single valued; they cannot convey more than one event simultaneously.

Similar to [Alur90, Lewi90], we adopt a continuous model of time. 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 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 closed, 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 $\mathcal{D}$.

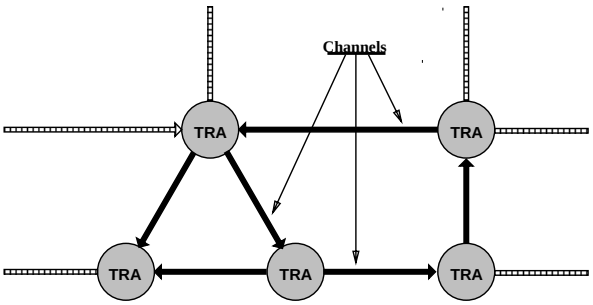

Figure 2: TRA objects and channels.

Figure-3 illustrates the notions of actions, events, and signals of a channel `MOVE` of some TRA. Let `North`, `South`, `East`, and `West` be the possible values that can be signaled on `MOVE`. `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 $\langle$`MOVE(East)` $: t_1\rangle$.The sequence $\langle$`MOVE(East)` $: t_1\rangle\langle$`MOVE(North)` $: t_2\rangle\langle$`MOVE(South)` $: t_3\rangle$ *... etc.* constitutes a signal.

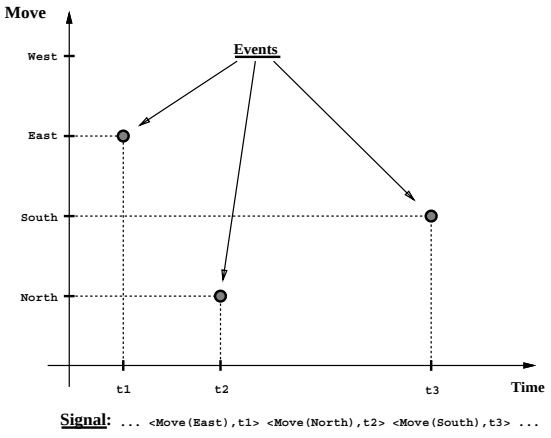

Figure 3: Signals, events, and actions.

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.

3.3 The TRA Object

Formally, a TRA object (see Figure-4) 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.

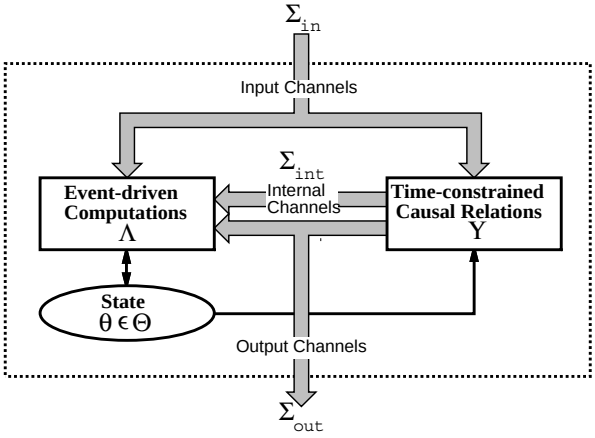

Figure 4: Basic components of a TRA object.

- $\sigma_0 \in \Sigma_{\text{in}}$ 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 $\pi \in \Pi(\Sigma_{\text{in}})$, 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 (see Figure-5). 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.

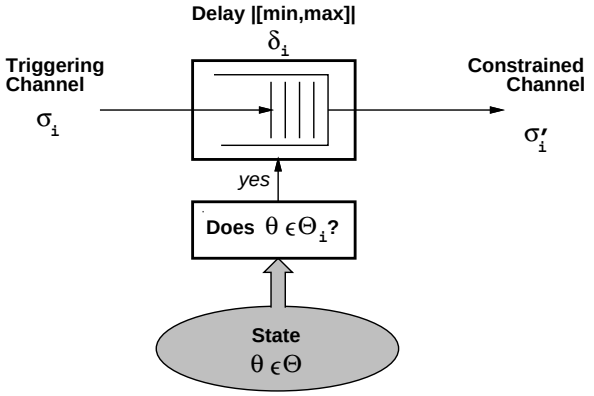

Figure 5: Time-constrained causal relationship.

As an example of a TRA specification, consider the the up/down counter whose state diagram is shown in Figure-6. The counter accepts commands issued on the input channel `cmd` to count up or down and signals the value of the current count (state) on the output channel `cnt`. The counter starts its operation once an action is fired on the `init` channel. The value of the `init` signal determines the starting state of the counter. The counter is constrained to produce a count every at least 1.9 and at most 2.1 units of time, once it starts execution. The formal TRA specification of the counter is shown in Figure-7.

3.4 The TRA Operational Semantics

In this section, we describe the rules governing the reaction of a TRA object to the events occuring on its input channels. Once these rules are specified, the possible behaviors of a given TRA can be determined.

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 unforeseable 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

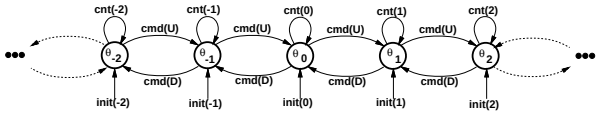

Figure 6: State diagram of up/down counter.

- $\Sigma = \Sigma_{\text{in}} \cup \Sigma_{\text{out}} \cup \Sigma_{\text{int}}$ is the signature of the counter, where $\Sigma_{\text{in}} = \{\texttt{cmd}, \texttt{init}\}$, $\Sigma_{\text{out}} = \{\texttt{cnt}\}$, and $\Sigma_{\text{int}} = \phi$.
- $\texttt{init} \in \Sigma_{\text{in}}$ is the start channel.
- The signaling range function is given by: $\Pi(\texttt{init}) = \mathcal{N}$, $\Pi(\texttt{cmd}) = \{\texttt{UP}, \texttt{DOWN}\}$, and $\Pi(\texttt{cnt}) = \mathcal{N}$.
- $\Theta = \{\theta_i : i \in \mathcal{N}\}$ is the set of states of the counter.
- The set of computational steps is: $\Lambda = (\bigcup_{i,j \in \mathcal{N}} \{(\theta_i, \texttt{init}(j), \theta_j)\}) \cup (\bigcup_{i \in \mathcal{N}} \{(\theta_i, \texttt{cmd}(\texttt{UP}), \theta_{i+1})\}) \cup (\bigcup_{i \in \mathcal{N}} \{(\theta_i, \texttt{cmd}(\texttt{DOWN}), \theta_{i-1})\}) \cup (\bigcup_{i \in \mathcal{N}} \{(\theta_i, \texttt{cnt}(i), \theta_i)\})$.
- The set of time constraints is: $\Upsilon = \{(\texttt{init}, \texttt{cnt}, [1.9, 2.1], \phi), (\texttt{cnt}, \texttt{cnt}, [1.9, 2.1], \phi)\}$.

Figure 7: TRA-specification of up/down counter.

explain why this is true consider the example shown in Figure-8, 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*.

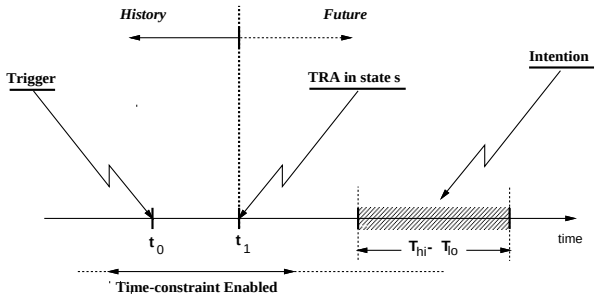

Figure 8: The notion of a TRA status.

Let $\upsilon_i = (\sigma_i, \sigma_i', \delta_i, \Theta_i) \in \Upsilon$ be a time constraint. As we explained before, υ_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. At any given point in time, a TRA might have several such outstanding intentions. 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 $\upsilon_i = (\sigma_i, \sigma_i', \delta_i, \Theta_i) \in \Upsilon$ is one of the TRA's time constraints, then $I[\upsilon_i] = \{\delta_{i1}, \delta_{i2}, \ldots, \delta_{ik}, \ldots \delta_{im}\}$ 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 υ_i. An empty intentions set, $I[\upsilon_i] = \phi$, indicates the absence of any activations of υ_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 extending the TRA's state. This is captured in our notion of a TRA status.

Definition 1 *We define the status of a* TRA $(\Sigma, \sigma_0, \Pi, \Theta, \Lambda, \Upsilon)$ *at any point in time* $t \in \Re$ *to be the tuple* (θ, I)*, where* $\theta \in \Theta$ *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. Five conditions have to be met for a status transition to take place. We informally present these conditions in the following definition. For a formal treatment refer to [Best91b].

Definition 2 *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* $\langle \pi' : t' \rangle$*, if and only if the following conditions hold:*

1. *Legality:*
   $(\theta, \pi', \theta') \in \Lambda$.
2. *Spontaneity:*
   $t' = t$ *only if* π *and* π' *are independent and occur in different components.*[3]
3. *Safety:*
   None of the intentions in I *expired at time* t'.
4. *Causality:*
   If $\sigma_j' \in \Sigma_{\text{loc}}$*, then there exists an intention in* I *to perform an action on* σ_j' *and* t' *is a possible time to commit that intention.*
5. *Consistency:*
   The intentions in I *continue to exist in* I' *unless otherwise dictated by the occurence of the event* $\langle \pi' : t' \rangle$.

[3] TRA components represent a weak notion of spacial locality. For a formal treatment, we refer the reader to [Best91b].

We use the notation $(\theta, I) \stackrel{\langle \pi' : t' \rangle}{\longmapsto} (\theta', I')$ 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 $(\theta, I) \stackrel{\alpha}{\longmapsto} (\theta', I')$ 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* of a TRA is a possibly infinite string of alternating statuses and events that starts with an initial status and an initiating event. A *schedule* α of an execution e is the sequence consisting of *all* the events appearing in e. Since internal events are invisible from outside a TRA, we will often be interested only in external events. We define β to be a *behavior* of a TRA $\mathcal{A}$, if it consists of all the *external* events appearing in some schedule α of $\mathcal{A}$. We denote the set of all possible behaviors of $\mathcal{A}$ by $behs(\mathcal{A})$. Obviously, $behs(\mathcal{A})$ describes all the possible interactions that $\mathcal{A}$ might be engaged in, and, therefore, constitutes a 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 [Lync88]. 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 is used mainly in verification [Best91a].

3.5 TRA Composition Operation

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 to preserve the input enabled property.

The input signature of the composition $\mathcal{A}$ 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.[4] 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 components. Similarily the time-constrained causal relationships of the composed TRA are exactly those of the component TRAs. The formal definition for the composition operation can be found in [Best91b].

The TRA composition operation is more general than those reported in [Lync88, Tutt88, Best90a] 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 [Lyon89, Lyon90b].

3.6 *CLEOPATRA*

CLEOPATRA[5] is a convenient language for the specification of embedded systems under the TRA formalism. *CLEOPATRA* specifications are executable and can be transformed, mechanically and unambiguously, into formal TRA objects for verification purposes [Best91b]. Throughout this paper, we will use *CLEOPATRA* to specify reactive behaviors.

In *CLEOPATRA*, systems are specified as interconnections of TRA objects. Each TRA object has a set of *state variables* and a set of *channels*. Time-constrained causal relationships between events occuring on the different channels, and the computations (state transitions) that they trigger, are specified using *Time-constrained Event-driven Transactions*. The behavior of a TRA object is described using TETs. TRA objects can be composed together to specify more complex TRAs.

In *CLEOPATRA*, TRAs are defined in *classes*. For example, Figure-9 shows the *CLEOPATRA* specification of the class of integrators that use trapezoidal approximation.

An integrator from the `integrate` TRA-class has two parameters, namely `TICK` and `TICK_ERROR`. Its signature consists of an input channel `in`, and an output channel `out`. Both `in` and `out` carry actions whose values are drawn from the set of reals. The body of `integrate` specifies a state space consisting of three real state variables `x0`, `x1` and `y`. It specifies two TETs. The first specifies that the response to an action on the input channel `in` is to store its value in state variable `x1`. The second specifies a transaction that is triggered initially by the `init` signal, and subsequently with every firing of `out`. After an amount of time bounded by a delay of `TICK` $\pm$ `TICK_ERROR` from when the transaction is triggered, an action of value `y` is signaled on `out`, and a state transition that

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

[5] A *C*-based *L*anguage for the *E*vent-driven *O*bject-oriented *P*rototyping of *A*synchronous *T*ime-constrained *R*eactive *A*utomata.

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

Figure 9: Specification of the `integrate` class.

updates the value of the state variable `y` and `x0` is committed.

```
typedef enum{0,1,X} tristate;

TRA-class subsume(double DELAY)
  dominant(tristate),inferior(tristate)
  -> behavior(tristate)
{
  state:
   tristate d_val = X, i_val = X ;
  act:
   dominant(d_val) -> behavior(d_val):
     before DELAY
       unless(d_val == X && i_val != X)
         commit { i_val = X ; }
   inferior(i_val) -> behavior(i_val):
     before DELAY
       unless(d_val != X)
         ;
}
```

Figure 10: The subsumption TRA in *CLEOPATRA*

4 TRA-based Specification

The TRA model and the *CLEOPATRA* language are ideal for specifying reactive behaviors of embedded systems. In this section we overview our experience in specifying basic, subsuming, and competing reactive behaviors, which correpsond respectively to the servo, selective, teleo-selective classes of control systems that we discussed earlier.

In modeling embedded systems, it is important that basic resources (e.g. actuators and sensors) be included in the system specification. The TRA framework has proven to be appropriate for representing the behavior of such low level controls. In particular, we used it to specify and simulate position and velocity feedback linear and non-linear control systems, as well as complex asynchronous digital circuits and systems [Best91b].

For reactive behaviors at a higher level, Rodney Brooks [Broo87] proposes the *subsumption architecture* as a methodology for specifying and building complex control systems. This architecture suggests the use of a vertical decomposition of the control system into a number of parallel independent task-achieving behaviors organized in a hierarchy of *dominant* and *inferior* behaviors. The subsumption architecture can be supported easily and effectively using the TRA model. In [Best91b], we showed that the TRA model is more general than the subsumption architecture. In particular, using the `subsume` TRA-class shown in Figure-10, dominant and inferior behaviors can be patched together to describe a complex behavior.

The subsumption architecture is suitable for the specification of task-achieving behaviors that can be *statically* organized as a hierarchy of dominant and inferior behaviors. It cannot deal with applications with *dynamically* changing priorities. In particular, if the priority of a behavior depends on the task (or goal) to be achieved, and if such a goal is dynamically changing, then this behaviour can be dominant in some situations and inferior in others. Rather than dominant and inferior behaviors, such systems are described in terms of *competing behaviors*.

The TRA framework is ideal for the specification of systems with competing behaviors. Examples of TRA behavioral specification of such systems were given in [Best90a]. The use of the TRA model in the specification and simulation of these systems is similar to Nilsson's action networks [Nils88], and Maes' situated agents [Maes90]. TRA specifications, however, allow (potentially automated) analysis to be performed on behaviors. For instance, given a finite-state TRA description, it is possible to obtain a finite-state description of all of its possible behaviors, and thus, proving assertions about these behaviors. This can be done using techniques similar to those suggested in [Lewi89, Alur90]. In addition, the TRA model provides a vehicle for efficient simulation and implementation using *CLEOPATRA*. Silicon compilation of *CLEOPATRA* specifications for simple behaviors is also a possiblity [Frie91].

As an example of TRA-specification of competing behaviors, consider the specification of *Buggy*, a bug-like autonomous creature. Buggy has two actuators to move in 2-D and two noisy sensors to locate predators, and detect floor cracks in a limited neighborhood. Buggy has two potentially competing behaviors, searching for food along cracks, and keeping itself away from predators (or obstacles). Buggy has only one goal: to survive (i.e. avoid starvation and predators). Buggy's urge to find food increases as time elapses and no food is found. Its fear from predators increases as its distance from them decreases. Dynamically, the behavior that is more important to Buggy's survival subsumes the other.

Buggy's competing behaviors were specified using *CLEOPATRA*. Figure-11 shows one of Buggy's simulated behaviors in a room with two cracks and a predator going in a circle. In this behavior one can identify some of Buggy's basic behaviors. In particular, when Buggy's sensors fail to detect any cracks or obstacles in its immediate neighborhood, Buggy's behavior is to *wander* randomly until the sensors return some readings. The pace of this wandering behavior (speed and rate of direction change) depends on the state of Buggy – its hunger and fear levels. Other basic behaviors of Buggy include approaching a crack, following a crack, and running away from ob-

stacles. In addition to the basic behaviors of Buggy, one can also identify a number of *emergent behaviors*. An emergent behavior is not specified explicitly; it emerges from the composition of other basic behaviors. For example, in Figure-11, two behavioral patterns can be easily singled out. The first is a hesitant behavior, in which, driven by hunger and fear, Buggy switches back and forth between approaching a crack to find food and running away from it to escape from the nearby rotating predator. The second is a routine behavior, in which Buggy reaches a limit-cycle of approaching a crack, following it, and running away from it.

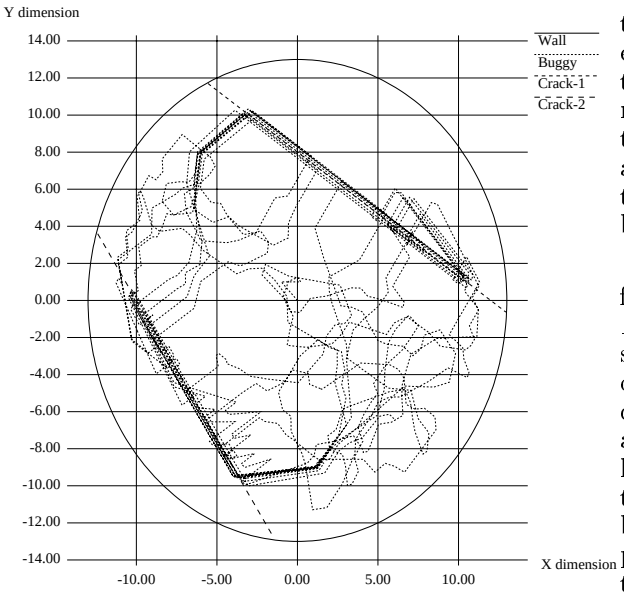

Figure 11: Basic and emergent behaviors of Buggy.

5 TRA-based Planning

As we argued earlier, planning in embedded systems entails synthesizing reactive behaviors that, when composed with existing behaviors (including the perceived behavior of the world), would result in the state of the world *potentially* progressing towards the satisfaction of a given set of goals – while *always* preserving system safety. We emphasize the word "potentially" because, synthesized behaviors are generated based on a perceived state of the world that might be inaccurate or dynamically changing. Also, we emphasize the word "always" because, as we explained earlier, the safety of an embedded system is far more important than its intelligence.

5.1 A simple planning example

Consider the robot *Sneaky*, who is assigned the task of going into a high security room. A security alarm sounds if the room's door is left open for more than a given amount of time. Also, an alarm will be signaled if a password is not correctly input within a given time frame after walking into the room. Figure-12 shows a partial CLEOPATRA specification of the perceived environment and the basic behaviors of the robot. The goal of the planner is to synthesize a behavior that when composed with the world's specification would result in Sneaky being in the room. Figure-13 shows a partial specification of such a behavior. The safety of the system requires that the security alarm does not go off at any time (because of Sneaky). This can be formaly proved by composing the TRAs representing Sneaky's behavior and the environment's specification, and showing that starting from an initially safe state, the composition cannot reach a state where the alarm goes off.

5.2 Planning through learning

The architecture we proposed in figure 1 decouples the planning agent from the tight loop between the embedded system and its environment. In a way, this architecture allows the agent to perform explicit reasoning at execution time without interfering with the system's reactive and timely behavior. Another advantage of such an architecture is that it allows the planning agent to modify a synthesized behavior based on its performance.

To exemplify this notion of planning by learning from experience, consider the simulated behavior of *Buggy* shown in Figure-11. A planning agent observing such a behavior will notice the reoccurence of a limit cycle in which Buggy discovers food on a crack, eats it while moving along the crack and, after a while, realizing it became very close to an obstacle leaves that path to get away from the wall. Eventually, however, it becomes hungry again, and goes back to the crack repeating almost exactly the same path. On other occasions, however, Buggy looses the crack and starts a random wandering phase away from the crack where there is no food, and thus, becomes threatened with starvation. A planning agent can benefit from observing such a behavior in a number of different ways. For example, by realizing that Buggy never starves when locked in a limit cycle, it might develop a subsuming *routine behavior* that makes it go blindly in that triangular limit cycle, once it finds one.

Similar potentials for learning from reactive behaviors were explored in [Maes89] where an autonomous mobile robot learned how to coordinate the use of its actuators to move forward. This work is limited in that the notion of "what to learn" is defined and static throughout the life-span of the creature – namely, coordination. In other words, rather than learning a "new" behavior, the creature is merely perfecting an existing behavior.

6 Conclusion

In this paper, instead of trying to adapt planning to real-time systems, we followed the alternative approach of redefining planning in a way acceptable to the real-time research community. In other words, instead of making intelligent systems real, we try to fit intelligence in real systems. This seems to be

```
GrabKnob -> :
  within [2~5]
    commit { Grabbing = TRUE ; }
ReleaseKnob -> :
  within [2~5]
    commit { Grabbing = FALSE ; }
Turn -> OpenDoor:
  unless (!Grabbing)
    within [40~50]
      commit { DoorOpen = TRUE ; }
OpenDoor -> Alarm:
  unless (ValidPasswd)
    within [40~50]
      commit { AlarmSet = TRUE ; }
OpenDoor -> Alarm:
  unless (!DoorOpen)
    within [100~110]
      commit { AlarmSet = TRUE ; }
MoveIn -> :
  unless (Grabbing || !DoorOpen)
    within [20~30]
      commit { RobotIn = TRUE ; }
EnterPasswd -> :
  unless (!RobotIn)
    within [5~8]
      commit { ValidPasswd = TRUE ; }
PressButton -> CloseDoor:
  unless (!RobotIn)
    within [40~50]
      commit { DoorOpen = FALSE ; }
```

Figure 12: Partial world model for Sneaky.

```
Init -> GrabKnob:
  within [1~2] ;
GrabKnob -> Turn:
  within [6~7] ;
Turn -> ReleaseKnob:
  within [60~60] ;
ReleaseKnob -> MoveIn:
  within [6~7] ;
MoveIn -> EnterPasswd:
  within [35~40] ;
MoveIn -> PressButton:
  within [35~80] ;
```

Figure 13: Partial synthesized plan for Sneaky.

the only viable approach especially in critical applications where lives and expensive machinery are at stake. We believe that for such systems, planning agents should produce *behavioral specifications* that, when *superimposed* on running behaviors, preserve the properties critical to the mission of the system. In this respect, we propose the TRA model as a framework for real-time plan generation and verification.

The TRA model is ideal for specifying reactive behaviors for embedded systems. On the one hand, its input enabled nature allows the specification of task-achieving behaviors in a realistic manner. On the other hand, its formal capabilities and compositional nature make possible the analysis and verification of safety conditions within a given environment. We have developed a specification language *CLEOPATRA* based on the TRA model. Behavioral specifications written in *CLEOPATRA* can be compiled and executed efficiently for simulation purposes.

Our experimentation with the TRA model as a backbone for building *intelligent control systems* is ongoing. In particular, we are working on an experiment that involves the management of sensorimotor activities for a robotics application that includes manipulative and active vision tasks executed in a dynamic environment. Our experiment involves research in three different areas: real-time systems, software engineering, and artificial intelligence. The use of the TRA framework provides the necessary link between all three areas. It offers formal verification and analysis capabilities to guarantee real-time properties. It offers an expressive executable specification language to validate customers needs, and defines clear interfaces between the different system components. It offers a formalism suitable for the specification and generation of reactive tasks.

References

[Alle86] J. Allen and R. Pelavin. "A formal logic of plans in temporally rich domains." *IEEE Special Issue on Knowledge Representation*, October 1986.

[Alur90] 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.

[Baet91] J. Baeten and J. Bergstra. "Real time process algebra." *Formal Aspects of Computing*, 3(2):142–188, 1991.

[Best90a] 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.

[Best90b] Azer Bestavros. "TRA-based real-time executable specification using CLEOPA-TRA." In *Proceedings of the 10th Annual Rochester Forth Conference on Embedded Systems*, Rochester, NY, June 1990. (revised May 1991).

[Best90c] 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.

[Best91a] Azer Bestavros. "Specification and verification or real-time embedded systems using the Time-constrained Reactive Automata." In *Proceedings of RTSS'91: The 12th IEEE Real-time Systems Symposium*, pages 244–253, San Antonio, Texas, December 1991.

[Best91b] 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.

[Broc88] Roger Brockett. "On the computer control of movement." In *Proceedings of the 1988 IEEE International Conference on Robotics & Automation*, Philadelphia, PA, 1988. IEEE Computer Society Press.

[Broo86] Rodney Brooks and Jonathan Connell. "Asynchronous distributed control system for a mobile robot." *SPIE Proceedings*, 727, October 1986.

[Broo87] Rodney Brooks. "A robust programming scheme for a mobile robot." In Ulrich Rembold and Klaus Hörmann, editors, *Languages for sensor-based control in Robotics* -

NATO ASI series, pages 509–522. Springer-Verlag/NATO, 1987.

[Frie91] Dan Friedman and James Clark. "Silicon compilation of simple sensori-motor behaviors." , 1991. Private communication of ongoing research.

[Hoar85] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1985.

[Kael86] Leslie Pack Kaelbling. "An architecture for intelligent reactive systems." Technical Report Technical Note 400, SRI International, 333 Ravenswood Ave., Menlo Park, CA 94025, October 1986.

[Leve91] Nancy Leveson. "Software safety in embedded computer systems." *Communications of the ACM*, 34(2), February 1991.

[Lewi89] 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.

[Lewi90] 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.

[Lync88] Nancy Lynch and Mark Tuttle. "An introduction to Input/Output Automata." Technical Report MIT/LCS/TM-373, MIT, Cambridge, Massachusetts, November 1988.

[Lyon89] 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.

[Lyon90a] D. Lyons, R. Pelavin, and A. Hendriks D. Benjamin. "Task planning using a formal model for reactive robot plans." In *Proceedings of the 1990 Spring Symposium on Planning in Dynamic and Uncertain Environments*, Stanford, California, March 1990.

[Lyon90b] Damian Lyons. "A formal model for reactive robot plans." In *Proceedings of the 2nd International Conference on Computer Integrated Manufacturing*, Troy, New York, May 1990.

[Lyon90c] Damian Lyons. "A process-based approach to task plan representation." In *Proceedings of the 1990 IEEE International Conference on Robotics & Automation*, Cincinati, Ohio, May 1990. IEEE Computer Society Press.

[Lyon91a] D. Lyons and A. Hendriks. "Reactive planning." Technical Report Philips TR-91-016 (MS-91-023), Philips Laboratories, Briarcliff Manor, New York, April 1991. To appear in the 2^{nd} edition of the Encyclopedia of Artificial Intelligence (S. Shapiro, Editor-in-chief – John Wiley & Sons, Inc.).

[Lyon91b] D. Lyons, A. Hendriks, and S. Mehta. "Achieving robustness by casting planning as adaptation of a reactive system." Technical Report Philips TN-91-011, Philips Laboratories, Briarcliff Manor, New York, February 1991.

[Maes89] Pattie Maes. "How to do the right thing." *Connection Science journal*, 1(3), 1989.

[Maes90] Pattie Maes. "Situated agents can have goals." *Special issue of journal of Robotics and Autonomous vehicle control*, Spring 1990. Also, in Designing Autonomous Agents - Pattie Maes editor, MIT Press.

[Nils88] Nils Nilsson. "Action networks." In *Proceedings of the Rochester Planning Workshop: From Formal Systems to Practical Systems*, University of Rochester, Rochester, NY, October 1988.

[Nils90] Nils Nilsson and Azer Bestavros, November 1990. Private discussions.

[Pela88] R. Pelavin. *A formal approach to planning with concurrent actions and external events*. PhD thesis, Computer Science Department, University of Rochester, Rochester, NY, May 1988.

[Reed88] G. M. Reed and A. W. Roscoe. "A timed model for Ccommunicating Sequential Processes." *Theoretical Computer Science*, 58:249–261, 1988.

[Rose85] Stanley Rosenschein. "Formal theories of knowledge in AI and robotics." Technical Note 362, SRI International, 333 Ravenswood Avenue, Menlo Park, CA 94025, September 1985.

[Sree90] Ramavarapu Sreenivas. *Towards a system theory for interconnected Condition/Event systems*. PhD thesis, Carnegie Mellon University, Pittsburgh, PA, September 1990.

[Stan88] John Stankovic and Krithi Ramamritham, editors. *Hard Real-Time Systems*. IEEE Computer Society Press, 1988.

[Tutt88] Mark Tuttle, Michael Meritt, and Francesmary Modugno. "Time constrained automata." MIT/LCS, November 1988.

[Wirt77] Niklaus Wirth. "Toward a discipline of real-time programming." *Communications of the ACM*, 20(8), August 1977.