Specification and Verification of Controlled Systems *

Henny B. Sipma and Zohar Manna

Department of Computer Science
Stanford University, Stanford, California 94305
e-mail: sipma@cs.stanford.edu, zm@cs.stanford.edu

Abstract. We propose a conceptual framework to support specification, design and verification of programs controlling physical systems. We introduce a computational model that represents the controller capabilities and distinguishes between synchronous and phase transitions. A graphical system description language is proposed that we believe is readily accessible to control engineers. We formalize the notion of control strategy in controller design.

1 Introduction

We present a conceptual framework to reason about controlled systems. We define a *controlled system* to be a plant consisting of both discrete and continuous components, subject to disturbances from the environment, and controlled by a digital control program, also called the *controller* (see Figure 1).

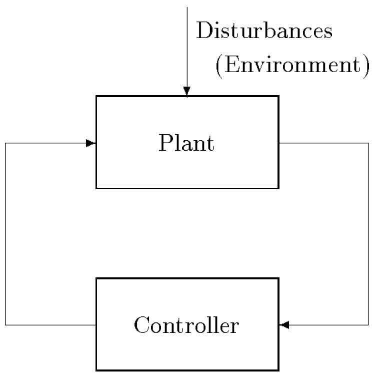

Fig. 1. Controlled system

* This paper appeared in the *Proceedings of the Third International Symposium on Formal Techniques in Real-time and Fault-tolerant Systems*, Lecture Notes in Computer Science **863**, Springer-Verlag, 1994, pp. 641–659.

The design of a reactive program usually starts with a specification of the program behavior. In the design of control programs this specification must first be derived. The control engineer is given a description of the plant, a description of the disturbances expected from the environment, and a set of requirements on plant behavior. His task is to specify and develop a control program such that the parallel composition of plant, environment and control program satisfies the plant requirements.

Informally, if we denote by S_P and S_E the set of behaviors allowed by the plant description and by the environment description respectively, and by R_P the set of behaviors allowed by the plant requirements, then it is the control engineer's task to find a control program specification S_C such that

$$S_P \cap S_E \cap S_C \quad \subseteq \quad R_P$$

The entire design process can be depicted as shown in Figure 2. Our work focuses on the Formal Description and the Control Specification, shown with double lines. Given an informal engineering description of the plant, we provide a formal description language to represent plant and environment behavior. It is well-known that, in practice, this formalization step is a major source of errors [JLHM91]; therefore an important consideration in the selection of a representation language is the readability of specifications written in it. Validation cannot be mechanized or formalized, since the original system description is provided informally; the engineer must convince himself that the formal description accurately reflects the intended system behavior. Ideally, the engineer should perform the formalization himself, so that whatever hidden assumptions may exist can be properly made explicit. We provide a graphical formal language, *Hierarchical Actor Activity Graphs (HAAG)* that we think is sufficiently similar to the engineer's usual language, yet has a well-defined underlying computational model. This computational model is then used as the basis for deriving and verifying a control program specification. HAAGs were in part inspired by Harel's Statecharts [Harel87], its hybrid extension [KP92] and by Ostroff's Extended State Machines [Ostroff90].

In section 2 we present the computational model. In sections 3 and 4 we present the requirements language and the system description language respectively. In section 5 we present a basis for control program design and in section 6 we work out an example from engineering description to controller design and verification to illustrate the languages. Section 7 concludes the paper with a discussion of the approach and a comparison with other work.

2 Computational Model

2.1 Transition System

We propose the Actor Phase Transition System (APTS) as a computational model for controlled systems. It is a combination of the phase transition system of [MP93] and the abstract phase transition system as described in [HMP93], augmented with the concept of actors.

From [MP93] we take the concept of discrete transitions that do not advance time, used to model controller transitions and those plant transitions that are an

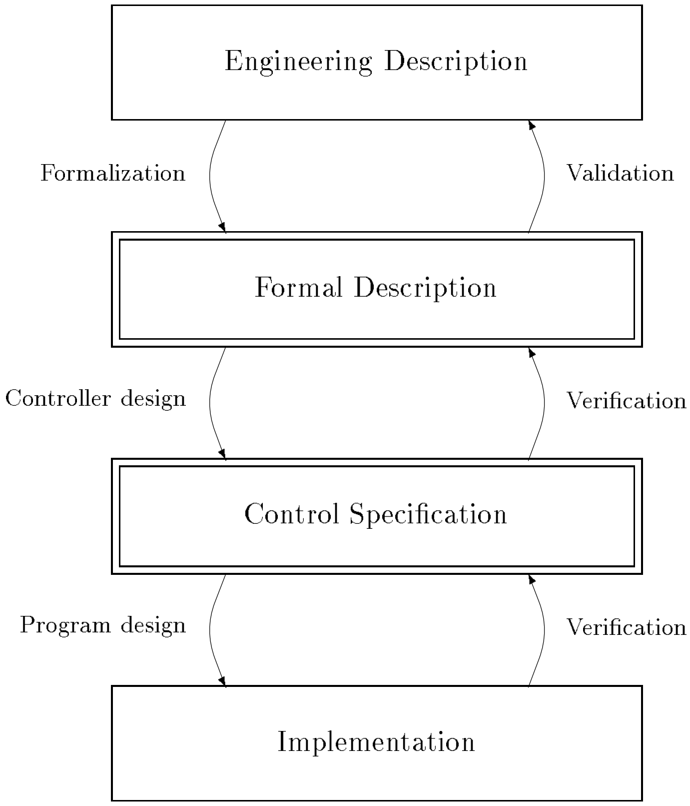

Fig. 2. Stages in control design

immediate consequence of controller actions. We assume that the controller is a digital program that interacts with the plant only a finite number of times in any finite time interval. As we are not interested in the real-time behavior of the controller, we model all controller transitions as happening at the time of the interaction. (Later we will see that we do not necessarily assume zero-computation time for the control program.) We call these transitions *synchronous transitions*; we consider the controller to be the synchronizer of a controlled system.

In our model, the concept of activities and important events [HMP93], which we believe is too restrictive to adequately describe controlled systems,[2] is replaced with the concept of phases and phase transitions [HMP93]. A *phase* is an interval

[2] Activities and important events are too restrictive as a general model. However, one of the approximate models we are considering is in fact equivalent to the [MP93] model.

of non-zero length combined with a family of functions defined over the interval, one function for each system variable. A *phase transition* is a binary relation on phases. Phase transitions are used to model most plant transitions and environment transitions.

We also add additional structure by introducing the concept of *actors*: variables are owned by actors, and transitions are performed by actors. In a controlled system, we have three principal actors: the *plant*, the *environment*, and the *controller*. The representation of actors in the model forms the basis for controller design. To derive a controller specification, we need to know the capabilities of the controller: which variables it can observe and which variables it can manipulate. In addition, we need to know which variables are, so to say, beyond control, being governed by the environment.

An APTS Φ has the following components:

- V: a finite set of *state variables*, partitioned into V_d, the set of *discrete variables*, V_c, the set of true *continuous variables*, i.e., variables that are continuous throughout a behavior; V_h, the set of *hybrid variables*. Hybrid variables can both vary continuously and undergo discontinuous changes, they are piece-wise continuous. A *state* is a type-consistent interpretation of V; the set of all states is denoted by Σ.
- Θ: the *initial condition*; it is an assertion which must be satisfied by the initial state.
- $\mathcal{T}_{\mathcal{S}}$: a finite set of *synchronous transitions*. Each transition $\tau \in \mathcal{T}_{\mathcal{S}}$ is a function

  $$\tau : \ \Sigma \mapsto 2^{\Sigma}$$

  defined by a transition relation $\rho_\tau(V, V')$. A synchronous transition can modify only discrete or hybrid variables.
- $\mathcal{P}$: a set of *phases* over V. Each phase is a pair (b, f), where $b \in \Re^+$ is the length of an interval $[0, b)$, and $f = \{f^x | x \in V\}$ is a family of continuous functions[3] defined on the interval $[0, b)$.
- $\mathcal{T}_{\mathcal{P}}$: a set of *phase transitions*. Each transition $\tau \in \mathcal{T}_{\mathcal{P}}$ is a binary relation on $\mathcal{P}$. We require for a pair of phases $(P_1, P_2) \in \tau \in \mathcal{T}_{\mathcal{P}}$ with $P_1 = (b_1, f_1)$ and $P_2 = (b_2, f_2)$ that for all $x \in V_c$

  $$\lim_{t \uparrow b_1} f_1^x(t) \ = \lim_{t \downarrow 0} f_2^x(t)$$

- $\mathcal{K}$: a finite set of actors. For controlled systems we consider three actors: *environment*, *plant*, and *controller*. Each of these can consist of a set of subordinate actors.
- $\mathcal{M}$: *modifiability* function; it is a mapping from $V \mapsto \mathcal{K}$. Each variable can be modified by exactly one actor. We also say the variable is owned by the actor. $\mathcal{M}$ induces a partition on V.

[3] The requirement that f^x be continuous differs from [HMP93], where only piece-wise continuity is required for phases. Our approach is as expressive, but it limits the combination of phase sequences into a new single phase.

- $\mathcal{O}$: *observability* function; it is a mapping from $V \mapsto 2^{\mathcal{K}}$. It denotes which variables are visible to which actors. We require that all variables modifiable by an actor are observable to that actor.
- $\mathcal{B}$: ownership of transitions; it is a mapping from $(\mathcal{T_S} \cup \mathcal{T_P}) \mapsto \mathcal{K}$. A transition is owned by exactly one actor. A transition owned by $k \in \mathcal{K}$ can only modify variables owned by k, and its guard can only contain variables observable to k.

2.2 Behavior

A *behavior* of APTS Φ is an infinite sequence

$$\sigma : \ \langle s_0, t_0 \rangle \ \overset{\alpha_1}{\to} \ \langle s_1, t_1 \rangle \ \overset{\alpha_2}{\to} \ \langle s_2, t_2 \rangle \ \overset{\alpha_3}{\to} \ldots$$

where

$$s_i \in \Sigma$$
$$\alpha_i \in \mathcal{K} \cup \{cont\}$$
$$t_i \in \Re^+$$

that satisfies

Initiation: $s_0 \models \Theta, \ t_0 = 0$
Consecution: for each $j = 0, 1, 2, \ldots$ the step $\langle s_j, t_j \rangle \overset{\alpha_{j+1}}{\to} \langle s_{j+1}, t_{j+1} \rangle$ is either
- a synchronous transition: $t_j = t_{j+1}$ and $s_{j+1} \in \tau(s_j)$ for some $\tau \in \mathcal{T_S}$ and $\alpha_{j+1} \in \mathcal{B}(\tau)$, or
- a continuous step: $t_{j+1} > t_j$ and $\langle s_{j+1}, t_{j+1} \rangle$ is a phase successor of $\langle s_j, t_j \rangle$ and $\alpha_{j+1} = cont$.

Urgency No continuous step may disable a synchronous transition.
Time divergence Time grows beyond any bound.

Synchronous transitions correspond to the interactions between plant and controller; they are also referred to as interaction instants.

A *phase successor* is defined as follows: $\langle s_{j+1}, t_{j+1} \rangle$ is a phase successor of $\langle s_j, t_j \rangle$ if there exists a finite phase sequence $P_0 \ldots P_n$, where $P_i \in \mathcal{P}$, and $P_i = (b_i, f_i)$ such that

$$s_j = \lim_{t \to 0} f_0(t)$$
$$s_{j+1} = \lim_{t \to b_n} f_n(t)$$

and

$$(\sum_{i=0}^{n} b_i) \ = \ t_{j+1} - t_j$$

and for all i: $0 \leq i < n$

$$(P_i, P_{i+1}) \in \tau \text{ for some } \tau \in \mathcal{T_P}$$

Similar to the semantic model of [MP93] and unlike the model of [HMP93], our model is not a function from time to states: the instants of plant-controller interaction are multi-valued.

3 Requirements Language

We use Hybrid Temporal Logic (HTL), as described in [HMP93] and [KHMP94], as our requirements language. As mentioned before, our semantic model differs from [HMP93] due to the interleaving of phases with discrete transitions that do not advance time. Furthermore, our definition of a phase does not allow concatenation of a sequence of phases into a single phase. Therefore, we need to adapt the definitions of satisfaction of formulas over arbitrary intervals.

We introduce the concept of a *superphase* (b, f) which represents a finite or infinite phase sequence $P_0, P_1, \ldots$, with $P_i = (b_i, f_i)$ such that for all $i = 0, 1, 2, \ldots$

$$f\left(t + \sum_{j=0}^{i-1} b_j\right) = f_i(t)$$

and $b = \sum_0^n b_i$ for finite phase sequences, and $b = \sum_0^\infty b_i$ for infinite phase sequences.

In our model, HTL formulas are interpreted over superphases in the same way as formulas are interpreted over phases in [HMP93]. Note that in a superphase the value of f at an interaction time point t is the limit from the right of the phase that starts at t. All intermediate values due to the synchronous transitions are invisible.

Making these intermediate values invisible is not unreasonable. Modeling transitions as taking zero time is an approximation of the real world. By doing so, one indicates not to be interested in the real-time behavior of these transitions, and therewith forfeits the option to obtain real-time information about them.

We give a brief description of the syntax and semantics of HTL . For the full exposition the reader is referred to [HMP93] and [KHMP94].

Syntax

A *local formula* is an atomic formula over the variables in V (a set of typed variables), their limits from the left and from the right, (denoted by $\overrightarrow{x}$ and $\overleftarrow{x}$ respectively), their left and right derivatives, constants, and the function symbols $<$, $>$, $\leq$, $\geq$, $+$, $\cdot$, and $=$. The formulas φ of HTL are then defined inductively:

$$\varphi \ := \ \psi \mid \mathit{fin} \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \varphi_1; \varphi_2 \mid \forall x.\, \varphi$$

where $x \in V$ and ψ is a local formula. A *state formula* is an atomic formula over the variables in V (i.e., it does not contain temporal operators, limits or derivatives).

Semantics

Satisfaction of a HTL formula φ over a superphase $P = (b, f)$ is defined inductively:

$\varphi \models \psi$ iff the local formula ψ evaluates to *true*, where $\overleftarrow{x} = \lim_{t\downarrow 0} f_x(t)$ and $\overrightarrow{x} = \lim_{t\uparrow b} f_x(t)$ (if the limit exists, undefined otherwise), and the left and right derivative are defined similarly.

$P \models \neg\phi$ iff $P \not\models \phi$.

$P \models \mathit{fin}$ iff $b < \infty$.

$P \models \phi_1 \vee \phi_2$ iff $P \models \phi_1$ or $P \models \phi_2$.
$P \models \phi_1 ; \phi_2$ iff there are two phases P_1 and P_2 that partition P such that $P_1 \models \phi_1$ and $P_2 \models \phi_2$.
$P \models \forall x . \phi$ iff $P' \models \phi$ for all phases $P' = (b, f')$ that differ from P at most in the interpretation f'_x of x.

The common abbreviations are defined as follows:

$\lhd \phi$ stands for $\phi \vee (\phi; true)$
$\rhd \phi$ stands for $\phi \vee (true; \phi)$
$\Diamond \phi$ stands for $(\lhd \phi) \vee (\rhd \phi) \vee (true; \phi; true)$
$\Box \phi$ stands for $\neg \Diamond \neg \phi$

In particular it was shown in [KHMP94] that for any state formula φ and phase $P = (b, f)$

$P \models \Box \varphi$ iff $\forall t \in [0, b)$ φ holds at t

Sampling Semantics

Interpreting formulas over superphases provides continuous semantics. Our model also suggests a sampling semantics in a natural way. Instead of considering all possible sampling sequences of a behavior, we take as the sampling sequence of an APTS Φ only one sequence: the behavior itself, as defined in section 2.2, i.e. the sequence of states at the points of controller-plant interaction. This view of a sampling sequence is similar to the approach taken by Nerode in [NK93].

For an important class of properties, namely invariants of the form

$$\Box(x \;\; op \;\; c)$$

where $op \in \{<, >, \leq, \geq\}$, $x \in V_c$, and c a constant, the sampling and continuous semantics can be related. Informally, let $\models_{SS}$ denote satisfaction over a sampling sequence, and let $\models_{CS}$ denote satisfaction over a superphase.

Proposition 1 *Let*

$$\varphi : \quad \Box(x \;\; op \;\; c)$$

and

$$\psi : \begin{cases} \Box(x \;\; op \;\; (c+\delta)) & \textit{for} \;\; op \in \{>, \geq\} \\ \Box(x \;\; op \;\; (c-\delta)) & \textit{for} \;\; op \in \{<, \leq\} \end{cases}$$

where $x \in V_c$ and c, $\delta \in \Re$ constants, and $op \in \{<, >, \leq, \geq\}$.
If

$$\delta \;\; > \;\; \frac{1}{2} \cdot \max(\Delta_i) \cdot \max(|\dot{x}|)$$

then

$$\sigma_S \;\models_{SS} \psi \quad \textit{implies} \quad \sigma_P \;\models_{CS} \varphi$$

where σ_S is a sampling sequence, and σ_P the corresponding superphase, and $\max(\Delta_i)$ is an upper bound on the sampling interval $t_{i+1} - t_i$, and $\max(|\dot{x}|)$ is an upper bound on the absolute value of the derivative of x over σ_P.

Corollary 1 *For a system with constant sampling interval, i.e. where for all* $i = 0, 1, 2, \ldots$, $t_{i+1} - t_i = \Delta$, *if*

$$\delta \quad > \frac{1}{2} \cdot \Delta \cdot \max(|\dot{x}|)$$

then

$$\sigma_S \models_{SS} \psi \quad \textit{implies} \quad \sigma_P \models_{CS} \varphi$$

where $\max(|\dot{x}|)$ *is an upper bound on the absolute value of the derivative of* x *over* σ_P.

The corollary expresses a useful property of sampling sequences. In the process industry, invariants of this type over continuous variables are very common, and most systems have fixed sampling intervals.

4 System Description Language

We use Actor Activity Graphs (AAG) to describe Actor Phase Transition Systems.

Actor Activity Graph (AAG) An AAG is a directed labeled graph $G = (K, E)$ consisting of

- Vertices K called actors. Each actor $k \in K$ is labeled by
  - a set of data variables $V(k)$, partitioned into discrete, continuous, and hybrid variables $V_d(k)$, $V_c(k)$, and $V_h(k)$, respectively, and
  - an initial condition $\Theta(k)$, which is a state formula, and
  - a hierarchical activity transition graph HATG(k) [HMP93].
- Edges E: each edge $e(k_1, k_2) \in E$ is labeled by a set of variables $V(e)$, such that $V(e) \subseteq V(k_1)$.

Intuitively, the edges between the actors can be viewed as communication channels: an edge $e(k_1, k_2)$ labeled by a set V denotes that the variables in V are modifiable by k_1 and visible to k_2.

Hierarchical Actor Activity Graph (HAAG) We allow for actors to be sets of subordinate actors. Incorporating this into the AAGs results in Hierarchical Actor Activity Graphs (HAAG). A HAAG is a directed labeled graph $G = (K, E)$ consisting of

- Vertices K called actors: each actor $k \in K$ is labeled by either
  - a finite number of HAAGs, or
  - a finite number of AAGs.
- Edges E: each edge $e(k_1, k_2) \in E$ is labeled by a set of variables $V(e)$ such that $V(e) \subseteq V(k_1)$.

Hierarchical Activity Transition Graph (HATG) A node in an AAG is labeled by a Hierarchical Activity Transition Graph (HATG). An HATG is the hierarchical version of an Activity Transition Graph (ATG), and can readily be converted into an ATG; the algorithm is described in [HMP93]. Here we only describe ATGs.

Activity Transition Graph (ATG) An ATG is a directed labeled graph $A = (L, E)$ consisting of

- Vertices L, called locations: each location $\ell \in L$ is labeled by an activity $\mu(\ell)$, which is an HTL formula.
- Edges E: each edge $e(\ell_1, \ell_2) \in E$ is labeled by a guarded command $\kappa(e) = \gamma \rightarrow \alpha$, where γ is a predicate over V, and α is an assignment $(\bar{v'} = \bar{\varphi})$ to variables in $V_d \cup V_h$.

From AAG to APTS In the following, we describe how an AAG $A(K, E)$, where each vertex $k \in K$ labeled with an ATG $G(L_k, E_k)$, defines an APTS. To simplify the description, we first define an auxiliary graph $\mathcal{H}$, the product graph of all actor ATGs. $\mathcal{H}$ is a labeled graph consisting of

- L_H, a set of vertices. L_H is the Cartesian product of the locations $\ell_k \in L_k$ of each actor $k \in K$. Let $|K| = N$, then each location $m \in L_H$ can be characterized by the tuple $(\ell_{k_1} \ldots \ell_{k_N})$ where ℓ_{k_i} denotes the location originating from the i^{th} actor. Each location $m = (\ell_{k_1} \ldots \ell_{k_N})$ is labeled by an activity:

  $$\mu_H(m) \;=\; \bigwedge_{k_i \in K} [\mu(\ell_{k_i}) \,\wedge\, (\pi_{k_i} = \ell_{k_i})].$$

- E_H, a set of edges. Let $m_1 = (\ell_{1,k_1} \ldots \ell_{1,k_N})$ and $m_2 = (\ell_{2,k_1} \ldots \ell_{2,k_N})$. $e(m_1, m_2) \in E_H$ iff there exists j such that

  $$(\forall i \neq j)(\ell_{1,k_i} = \ell_{2,k_i}) \;\wedge\; e(\ell_{1,k_j}, \ell_{2,k_j}) \in E_{k_j}$$

  in which case it is labeled by

  $$\kappa(e(\ell_{1,k_j}, \ell_{2,k_j})).$$

  Note that we do not have product edges in E_H.

The APTS Φ is now defined as follows:

1. The set of variables V includes the variables belonging to each actor plus a single control-variable for each actor, i.e.,

   $$V = \bigcup_{k \in K} V(k) \;\cup\; \{\pi_k | k \in K\}$$

   where π_k ranges over L_k. V is partitioned into V_c, V_d, V_h in the obvious way:

   $$V_c = \bigcup_{k \in K} V_c(k)$$

   $$V_h = \bigcup_{k \in K} V_h(k)$$

   $$V_d = \bigcup_{k \in K} V_d(k) \;\cup\; \{\pi_k | k \in K\}$$

2. The initial condition is the conjunction of the initial condition of all actors:

$$\Theta \;=\; \bigwedge_{k \in K} \Theta(k)$$

3. $\mathcal{T}_{\mathcal{S}}$ is the union of the set of all controller transitions and those plant transitions that are immediate consequences of controller transitions. Formally, for each edge $e(\ell_1, \ell_2) \in E_k$ labeled by $(\gamma \rightarrow \alpha)$, where $k = \mathit{controller}$ or $Var(\gamma) \subseteq V(\mathit{controller})$, $\mathcal{T}_{\mathcal{S}}$ contains the transition

$$\tau : \;\; (\pi_k = \ell_1) \, \wedge \, (\pi_k' = \ell_2) \, \wedge \, \gamma \, \wedge \, (\bar{v}' = \bar{\varphi})$$

4. The phases in $\mathcal{P}$ are defined over the locations of the product graph $\mathcal{H}$: for each location $m \in L_H$, $\mathcal{P}$ contains all phases P such that

$$P \models \mu_H(m)$$

5. The phase transitions $\mathcal{T}_{\mathcal{P}}$ are obtained from the edges of the product graph $\mathcal{H}$. For each edge $e_H(\ell_1, \ell_2) \in E_H$ labeled by $(\gamma \rightarrow \alpha)$, $\mathcal{T}_{\mathcal{P}}$ contains all transitions $\tau_e = (P_1, P_2)$ such that

$$\overrightarrow{P_1} \models (\pi = \ell_1) \, \wedge \, \gamma$$
$$\overleftarrow{P_2} \models (\pi = \ell_2)$$

   and for all $v \in V_c$:

$$(\overrightarrow{P_1}, \overleftarrow{P_2}) \;\models\; v = v'$$

   and $\overleftarrow{P_2}$ results from $\overrightarrow{P_1}$ by executing the assignment α. $\overleftarrow{P}$ and $\overrightarrow{P}$ denote the limit from the right and the limit from the left of a phase P, respectively.
6. The *modifiability* function $\mathcal{M}$ assigns to each variable the actor to which it belongs:

$$\mathcal{M}(v) = k \quad \text{iff} \quad v \in V(k)$$

7. The *observability* function $\mathcal{O}$ is derived from the edges of A:

$$k \in \mathcal{O}(v)$$

   iff there exists $e_A(k', k) \in E_K$ such that

$$v \in V(e_A)$$

In section 6 we show an example of an APTS defined by an HAAG.

5 Controller Design: Control Strategy

It is the controller's task to ensure that plant behavior meets the plant requirements for every environment behavior allowed by the environment description.

We start with an APTS Φ that describes plant and environment behavior, i.e., Φ describes the set $S_P \cap S_E$. Φ does not specify the behavior of the controller, nor the set of behaviors that is acceptable, i.e., R_P. Φ specifies the 'physics' of the plant and environment and, with $\mathcal{M}$ and $\mathcal{O}$, the capabilities of the controller to observe and manipulate the plant.

We can always partition transitions and variables between plant and environment (possibly with the aid of auxiliary variables) such that all plant transitions are deterministic. If we do so, then satisfying the plant requirements can be viewed as a two-player game between controller and environment. However, in this game, the controller is handicapped: it has only limited means to win the game. The controller can observe only a subset of the system variables whereas we must assume that the environment has access to all variables. In addition the controller can react only at fixed points in time.

Abadi and Lamport [AL90] introduced the notions of μ-strategy and μ-outcomes to describe the concept of realizability of a reactive program. We adapt these concepts to define control strategy and controlled behaviors. A *control strategy* is a rule that tells the controller what action to impose on the plant based on what it has observed so far. A *controlled behavior* is then a resulting behavior when applying a control strategy.

Formally, a *control strategy* g is a function from the set of finite behavior prefixes projected onto O_c to Σ, where O_c is the set of variables that can be observed by the controller, i.e., $O_c = \{v \in V \mid \mathcal{O}(v) \in controller\}$. A sequence σ is a *g-controlled behavior* of a phase transition system Φ if

- σ is a behavior of Φ, and
- for all $m > 0$ if $\alpha_m \in controller$ then

$$g(\sigma|_m) = s_m$$

where $\sigma|_m$ denotes the prefix of σ of length $m+1$. $\mathcal{C}_\Phi(g)$ is the set of all g-controlled behaviors of Φ with control strategy g.

With these definitions we can reformulate the control engineer's task as

finding a control strategy g such that all resulting
g-controlled behaviors meet the plant requirements, i.e.

$$\mathcal{C}_\Phi(g) \subseteq R_P$$

where R_P denotes the set of behaviors allowed by the set of plant requirements.

Often, in practice, it is desirable to find a class of strategies, preferably as large as possible, rather than a single strategy, to allow maximum flexibility for subsequent optimization. Therefore we define the realizable part $\mathcal{G}$ of a set of requirements R_P with respect to a controlled system Φ:

$$\mathcal{G} \;=\; \bigcup_g \{\mathcal{C}_\Phi(g) \mid \mathcal{C}_\Phi(g) \subseteq R_P\}$$

The handicap of the controller with respect to observed and manipulated variables is represented in the model Φ. However, we can impose more restrictions on the control strategy. For example, to simulate non-zero computation time we can limit the number of transitions the controller can take between continuous steps. We can also require that the controller can react only to observations obtained until the previous sampling point, i.e.,

$$g(\sigma|_{m-1}) \;=\; s_m$$

This is a common situation in the process industry.

It may also be useful to determine the severity of the handicap. We define an extended-observability control strategy $g(O_e)$ as a function from the set of finite behavior prefixes projected onto O_e to Σ, where O_e is a superset of O_c; the corresponding set of behaviors is denoted by $C_\Phi(g, O_e)$. The fully-informed realizable part $\mathcal{G}_V$ of a set of requirements R_P with respect to APTS Φ is

$$\mathcal{G}_V \;=\; \bigcup_g \;\{\mathcal{C}_\Phi(g,V) \mid \mathcal{C}_\Phi(g,V) \subseteq R_P\}$$

where V is the set of all system variables. Thus, $\mathcal{G}_V$ denotes the set of realizable behaviors if the controller could observe all system variables.

6 Engineering Example

6.1 Engineering Description

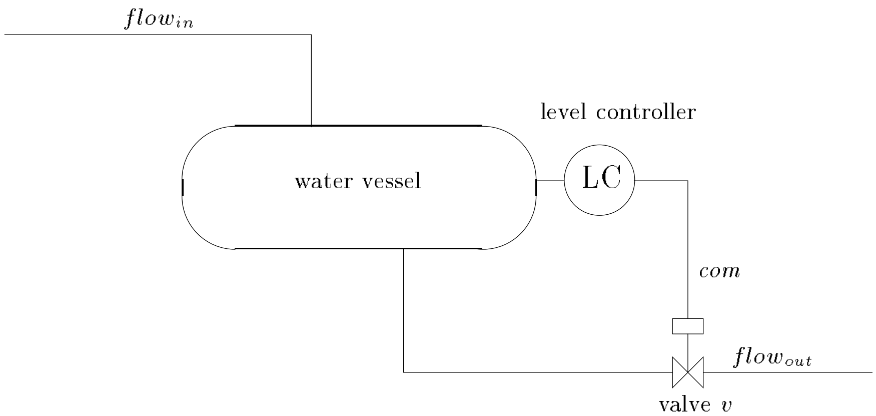

Fig. 3. Water-level system

In this section we illustrate the framework presented in the previous sections with the example given in [HMP93]. The controlled system, shown in Figure 3, consists of a water vessel with one inlet line and one outlet line. Water is entering the vessel through the inlet line, its flowrate determined by the environment, but not to exceed $1m^3/min$. The water level in the vessel is controlled by a level-controller LC that can manipulate a valve in the outlet line. The valve accepts two commands: *on* and *off*. When the command is *on* the valve opens at the rate of $(1m^3/min)/min$; when it is *off*, it closes at the same rate. Its extreme positions are $v = 0$ (fully closed) and $v = 1$ (fully open).[4] When starting up the system, the water level is 68, the valve is closed and the command is *off*. The engineering description of this system is:

– Environment S_E:

$$flow_{in} \; \leq \; 1m^3/min$$

– Plant S_P:

$$\frac{dw}{dt} \; = \; 8(flow_{in} - 2 \cdot v)$$

$$\text{com} = on: \quad \frac{dv}{dt} \; = \; 1 \;, \; v \leq 1$$

$$\text{com} = off: \quad \frac{dv}{dt} \; = \; -1 \;, \; v \geq 0$$

and we are asked to come up with a control strategy such that the following plant requirements are met:

$$R_P : \quad 60 \; \leq \; w \; \leq \; 76$$

6.2 Formal Description: HAAG

We first formalize the engineering description into the corresponding HAAG shown in figure 4. The double-sided boxes represent the actors. The ovals represent the nodes of the HATGs. We expect that an engineer can validate this representation. Notice that we have added some information not explicitly present in the engineering description, such as $flow_{in} \geq 0$, and the fact that the controller has access only to the water level w, and the fact that its only means of manipulating the plant is through *com*. To any engineer, this information is obvious from the picture given in Figure 3, and would be used subconsciously whenever needed. However, in the formal description we cannot allow such imprecisions.

This example illustrates another advantage of this representation. A system description can easily be constructed from standard, possibly parameterized, modules. For example, the valve HATG could be a library module.

6.3 Formal Representation: APTS

Once the HAAG has been validated by the engineer, the APTS it defines can be mechanically determined. The full APTS is given in appendix A. We have used phase relations as described in [KHMP94] to represent the phases.

[4] We realize that this example is not a particularly realistic one: *on-off* control and a linear dependence of flow on valve opening are not very common in practice. However it is simple and serves the purpose of illustrating the concepts in this paper.

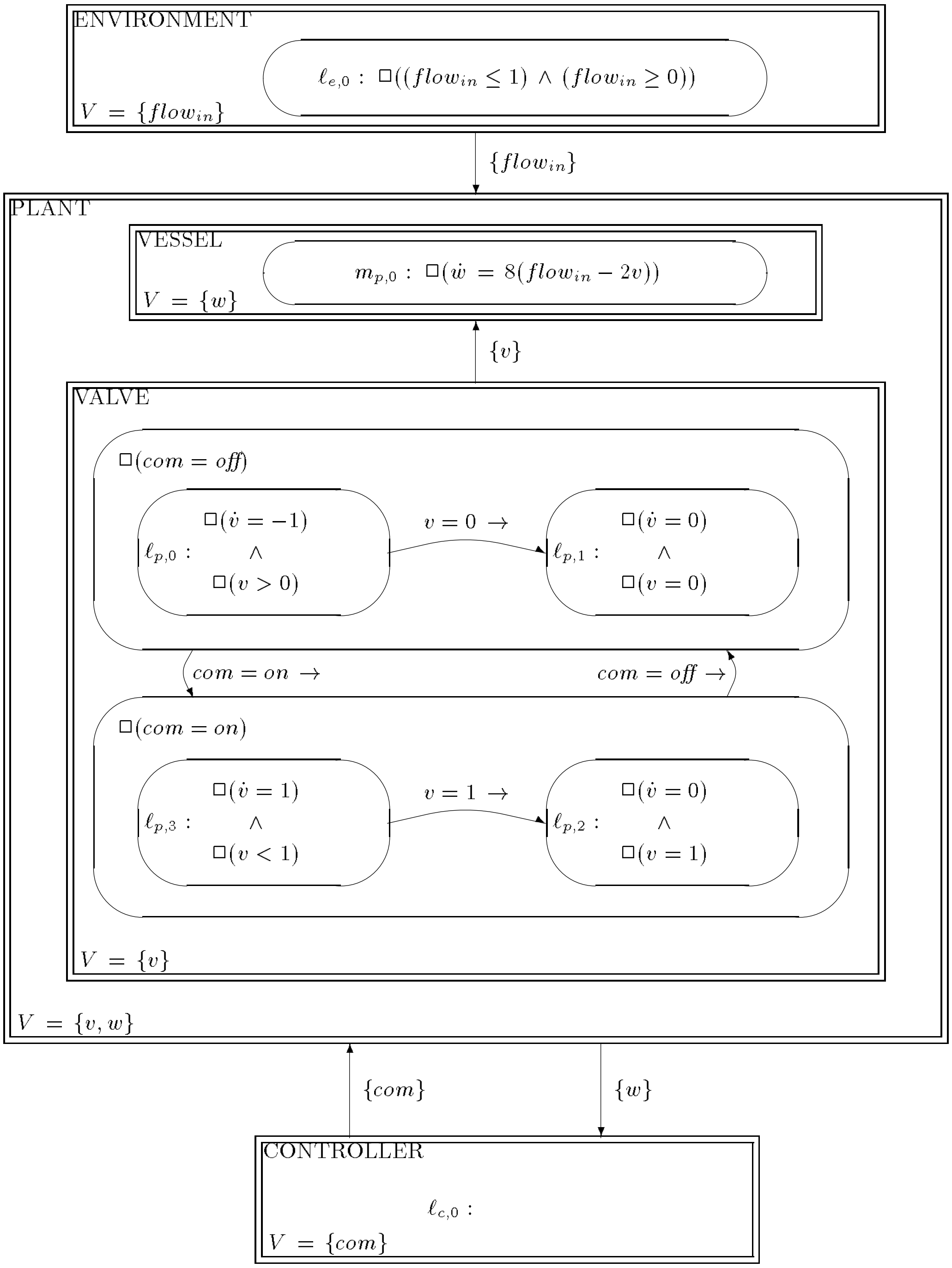

Fig. 4. HAAG of the water-level system

6.4 Controller design

It is shown in Appendix A that the continuous behavior of the water level w can be represented by four phase schemas. In table 1, for each of these schemas, the effect on w is given for the extremes of inlet flowrate. For example, in phases described by ϕ_3, w strictly decreases, and in phases described by ϕ_1, in case of maximum inletflow, w may decrease a maximum of 2, after which it will increase by a maximum of 2, and in case of minimum inletflow, w will drop 8 units.

phase schema	f^{com}	effect on w	
		$flow_{in} = 0$	$flow_{in} = 1$
ϕ_0	*off*	0	*inc*
ϕ_1	*off*	-8	$-2, +2$
ϕ_2	*on*	*dec*	$+2, -2$
ϕ_3	*on*	*dec*	*dec*

Table 1. Phase effects on water level

To design a controller to meet the requirements

$$R_P : \ \Box(60 \leq w \leq 76)$$

we look at the transitions between phases represented by these phase schemas. There are four synchronous transitions: (ϕ_0, ϕ_2), (ϕ_3, ϕ_1), (ϕ_1, ϕ_2), and (ϕ_2, ϕ_1), and two phase transitions: (ϕ_1, ϕ_0) and (ϕ_2, ϕ_3), see Figure 5 for a graphical representation; the phase transitions are shown with thick lines.

Clearly, ϕ_0 must be left before the level reaches 76, and ϕ_3 must be left before w reaches 60, so a first attempt at a specification gives:

$$w > 76 \rightarrow com = on$$
$$w < 60 \rightarrow com = off$$

However, from Figure 5, it can be seen that there are no transitions between ϕ_0 and ϕ_3, so the direction of change in w cannot be reversed at will by the controller. Furthermore, there is no synchronous transition between ϕ_1 and ϕ_0: the controller can cause the system to move from ϕ_3 to ϕ_1, but it is dependent on the plant to move to ϕ_0. In the worst case, according to table 1, w may drop another 8 units without the controller being able to do anything about it. Thus, to meet the requirements, the controller must ensure that ϕ_3 is left before the level reaches 68. With a similar reasoning about the path $\phi_0 \rightarrow \phi_2 \rightarrow \phi_3$ we arrive at the new specification S_C

$$w > 74 \rightarrow com = on$$
$$w < 68 \rightarrow com = off$$

To illustrate the effect of limited observability we add v to $\mathcal{O}_c$. With this extended set of observable variables we can achieve a less conservative specification, allowing

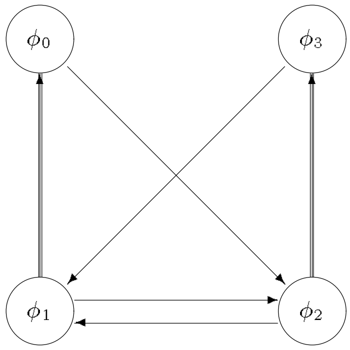

Fig. 5. Phase transitions

a larger set of behaviors, namely

$$\begin{aligned} w - 8v + 8v^2 &> 74 \rightarrow com = on \\ w - 8v^2 &< 60 \rightarrow com = \mathit{off} \end{aligned}$$

6.5 Verification of the Specification

The specification derived in the previous section still needs to be verified. It must be proven that any g-controlled behavior, where g satisfies S_C, satisfies the requirements R_P. This verification was performed using the proofrules for point-invariance for HTL described in [KHMP94]. Details of the proof will be given in a forthcoming paper [SM94]. An interesting, although in hindsight not surprising, aspect about the verification is that the property

$$(w \leq 76) \wedge (w \geq 60)$$

is not strong enough to be proven directly. Following the terminology of [MP94], we can say it is not inductive. The property must be strengthened into

$$(w \geq 60 + 8v^2) \wedge (w \leq 74 + 8v - 8v^2)$$

In general, we expect to encounter this phenomenon when the plant contains latency.

7 Discussion

We have proposed a novel approach to the specification of control systems which allows the inclusion of the capabilities of the controller. The inability to do so often results in awkward specifications with *adhoc* conditions. This can be seen for example, in [HMP93], where the restriction that a controller cannot perform an environment transition is added as an afterthought rather than as a design starting point. In other work we see that the capabilities are assumed to be so obvious they need not be mentioned. For example, in the Gas Burner example [RRH93], it seems obvious that the controller can manipulate the gas and observe the flame. In real design problems however, this is not always obvious.

It may seem restrictive to require that each variable can be modified by at most one actor. In the Gas Burner example, the variable *flame* is modified both by the plant (when the gas is on, then the flame is on) and by the environment (which can blow off the flame). However, it is always possible, with the use of auxiliary variables, to make each variable belong to exactly one actor. In the Gas Burner example, we could introduce an environment variable *blowoff*, which makes *flame* purely a plant variable.

We formalized the notions of control strategy and controlled behavior as a basis for controller design.

We have also proposed a new system description language with AAGs and HAAGs, which we believe is more readable by engineers than existing specification languages, and which facilitates composing specifications from standard library modules.

Acknowledgement

We thank Amir Pnueli, Eddie Chang, Hugh McGuire, and the anonymous referees for their careful reading of the manuscript and for many helpful suggestions.

References

[AL90] M. Abadi and L. Lamport. Composing specifications. In J.W. de Bakker, W.-P. de Roever, and G. Rozenberg, editors, *Stepwise Refinement of Distributed Systems*, Lecture Notes in Computer Science 430, pages 1-41. Springer-Verlag, 1990.

[Harel87] D. Harel. Statecharts: A visual Formalism for Complex Systems. *Science of Computer Programming*, **8**, pages 231-274, 1987.

[HMP93] T.A. Henzinger, and Z. Manna, and A. Pnueli. Towards Refining Temporal Specifications into Hybrid Systems. In R.L. Grossman, A. Nerode, A.P. Ravn, H. Rischel, editors, *Hybrid Systems*, Lecture Notes in Computer Science 736, pages 60-76. Springer Verlag, 1993.

[JLHM91] M.S. Jaffe, N.G. Leveson, M.P.E. Heimdahl, B.E. Melhart. Software Requirements Analysis for Real-Time Process-Control Systems. *IEEE Transactions on Software Engineering* **17**:3, pages 241-258, 1991.

[KHMP94] A. Kapur, T.A. Henzinger, Z. Manna, A. Pnueli. Proving Safety Properties of Hybrid Systems *This volume*, 1994.

[KP92] Y. Kesten, A. Pnueli. Timed and Hybrid Statecharts and their Textual Representation. In J.Vytopil, editor, *Formal Techniques in Real-Time and Fault-Tolerant Systems*, Lecture Notes in Computer Science 571, pages 591-619, Springer Verlag, 1992.

[MP93] Z. Manna, A. Pnueli. Models for reactivity. *Acta Informatica*, **30**, pages 609-678, 1993.

[MP94] Z. Manna, A. Pnueli. Temporal Verification of Reactive Systems. *To be published by Springer Verlag*, to appear 1994.

[NK93] A. Nerode, W. Kohn. Models for Hybrid Systems: Automata, Topologies, Controllability, Observability. In R.L. Grossman, A. Nerode, A.P. Ravn, H. Rischel, editors, *Hybrid Systems*, Lecture Notes in Computer Science 736, pages 317-365. Springer Verlag, 1993.

[Ostroff90] J.S. Ostroff. Temporal Logic of Real-Time Systems. Advanced Software Development Series, Research Studies Press (John Wiley & Sons), 1990.

[RRH93] A.P. Ravn, H. Rischel, K.M. Hansen. Specifying and Verifying Requirements of Real-Time Systems. *IEEE Transactions on Software Engineering*, **19**, pages 41-55, 1993.

[SM94] H.B. Sipma, Z. Manna. Interactive Design and Verification of Controlled Systems. In *Hybrid Systems and Autonomous Control Workshop*, to appear 1994.

APPENDIX

Water level system APTS

The HAAG given in Figure 4 defines the following APTS :

- $V = \{flow_{in}, w, v, com, \pi_{plant}, \pi_{env}, \pi_{controller}\}$, partitioned into

$$V_d = \{com, \pi_{plant}, \pi_{env}, \pi_{controller}\}$$
$$V_c = \{flow_{in}, w, v\}$$

- $\Theta = (w = 68) \wedge (v = 0) \wedge (com = \mathit{off}) \wedge (flow_{in} \geq 0) \wedge (flow_{in} \leq 1)$ $\wedge (\pi_{plant} = \{\ell_{p,1}, m_{p,0}\}) \wedge (\pi_{env} = \ell_{e,0}) \wedge (\pi_{controller} = \ell_{c,0})$.
- $\mathcal{T_S} = \{\tau_{s0}, \tau_{s1}, \tau_{s2}, \tau_{s3}\}$, where

$$\tau_{s0} = (\pi_{plant} = \{\ell_{p,0}, m_{p,0}\}) \wedge (\pi'_{plant} = \{\ell_{p,3}, m_{p,0}\}) \wedge (com = \mathit{off})$$
$$\tau_{s1} = (\pi_{plant} = \{\ell_{p,1}, m_{p,0}\}) \wedge (\pi'_{plant} = \{\ell_{p,3}, m_{p,0}\}) \wedge (com = \mathit{off})$$
$$\tau_{s2} = (\pi_{plant} = \{\ell_{p,2}, m_{p,0}\}) \wedge (\pi'_{plant} = \{\ell_{p,0}, m_{p,0}\}) \wedge (com = on)$$
$$\tau_{s3} = (\pi_{plant} = \{\ell_{p,3}, m_{p,0}\}) \wedge (\pi'_{plant} = \{\ell_{p,0}, m_{p,0}\}) \wedge (com = on)$$

- $\mathcal{P} = \{\phi_0, \phi_1, \phi_2, \phi_3\}$[5], where

$$\phi_0 = (\pi_{plant} = \{\ell_{p,0}, m_{p,0}\}) \wedge \mathcal{Q} \wedge (com = \mathit{off}) \wedge (v = 0)$$
$$\phi_1 = (\pi_{plant} = \{\ell_{p,1}, m_{p,0}\}) \wedge \mathcal{Q} \wedge (com = \mathit{off}) \wedge (v > 0) \wedge (v \leq 1) \wedge (\dot{v} = -1)$$
$$\phi_2 = (\pi_{plant} = \{\ell_{p,2}, m_{p,0}\}) \wedge \mathcal{Q} \wedge (com = on) \wedge (v \geq 0) \wedge (v < 1) \wedge (\dot{v} = 1)$$
$$\phi_3 = (\pi_{plant} = \{\ell_{p,3}, m_{p,0}\}) \wedge \mathcal{Q} \wedge (com = on) \wedge (v = 1)$$

where

$$\mathcal{Q} \;=\; \dot{w} = 8(flow_{in} - 2 \cdot v) \;\wedge\; (flow_{in} \geq 0) \;\wedge\; (flow_{in} \leq 1) \;\wedge\; (\pi_{env} = \ell_{e,0}) \;\wedge\; (\pi_{controller} = \ell_{c,0}).$$

[5] Each ϕ_i represents an infinite number of phases; the formula is a modified version of the phase schema, as introduced by [KHMP94].

- $\mathcal{T}_{\mathcal{P}} = \{(\phi_1, \phi_0), (\phi_2, \phi_3)\}$[6]
- $\mathcal{K} = \{environment, plant, controller\}$, where $plant = \{vessel, valve\}$.
- $\mathcal{M}$:

$$\begin{aligned}
\mathcal{M}(flow_{in}) &= environment \\
\mathcal{M}(w) &= plant \\
\mathcal{M}(v) &= plant \\
\mathcal{M}(com) &= controller \\
\mathcal{M}(\pi_{plant}) &= plant \\
\mathcal{M}(\pi_{controller}) &= controller \\
\mathcal{M}(\pi_{env}) &= environment
\end{aligned}$$

- $\mathcal{O}$:

$$\begin{aligned}
\mathcal{O}(w) &= \{controller, plant\} \\
\mathcal{O}(v) &= \{plant\} \\
\mathcal{O}(flow_{in}) &= \{environment, plant\} \\
\mathcal{O}(com) &= \{plant, controller\} \\
\mathcal{O}(\pi_{plant}) &= \{plant\} \\
\mathcal{O}(\pi_{controller}) &= \{controller\} \\
\mathcal{O}(\pi_{env}) &= \{environment\}
\end{aligned}$$

- $\mathcal{B}$:

$$\mathcal{B}(\tau_{s0}) = \mathcal{B}(\tau_{s1}) = \mathcal{B}(\tau_{s1}) = \mathcal{B}(\tau_{s2}) = plant.$$

[6] These transitions represent an infinite number of phase transitions.