On Control with Bounded Computational Resources*

Oded Maler[1], Bruce H. Krogh[2], and Moez Mahfoudh[1]

[1] VERIMAG
Centre Equation, 2, av. de Vignate
38610 Gières, France
Oded.Maler@imag.fr
[2] Dept. of Electrical and Computer Engineering
Carnegie Mellon University, 5000 Forbes Avenue
Pittsburgh, PA 15213-3890 USA
krogh@ece.cmu.edu

Abstract. We propose models that capture the influence of computation on the performance of computer-controlled systems, and make it possible to employ computational considerations in early stages of the design process of such systems. The problem of whether it is possible to meet performance requirements given resource constraints is phrased as a problem of synthesizing switching controllers for hybrid automata, for which we give algorithms that in some cases are guaranteed to terminate and in others can solve the problem in an approximate manner.

1 Background

In this work we build models that capture the influence of computational resources on the performance of computer-controlled systems. Such models allow one to employ computational considerations in early stages of the design process of control systems. We view computation as an essential resource for achieving high-quality control, a resource that in certain situations may become a bottleneck in the system. As a first step toward dealing explicitly with the interaction between control performance and the allocation of computational resources, we consider in this paper the problem of scheduling feedback computations on a single computer controlling multiple independent processes. Notwithstanding some anomalies, we may assume that the quality of control improves monotonically with the amount of computation invested (both on-line and off-line) in the control loop, as the following examples show:

1. *Sampling rates*. Usually the quality of control improves with the frequency of the basic loop (sample input, compute feedback function and output) and it approaches the ideal continuous model as the sampling rate goes to infinity. Naturally, the

* This work was partially supported by the EC projects IST-2001-33520 CC (Control and Computation) and IST-2001-35302 AMETIST (Advanced Methods for Timed Systems), the US Defense Advance Projects Research Agency (DARPA) contract no. F33615-00-C-1701, US Army Research Office (ARO) contract no. DAAD19-01-1-0485, and the US National Science Foundation (NSF) contract no. CCR-0121547.

amount of computational effort in a digital control implementation is proportional to the sampling rate which specifies, roughly, the number of times the feedback function is evaluated in any time interval.

2. *Alternative feedback function.* In certain situations, one may choose between several feedback functions whose complexity increases with their quality. For example, in model-predictive control, where the feedback function is computed using an optimization procedure over a bounded horizon, longer decision horizons increase the dimensionality of the optimization problem and hence its complexity.
3. *Control under noise.* In order to cope with noisy measurements, sophisticated filtering and state-estimation algorithms need to be applied. These functions can be implemented using dedicated hardware or, alternatively, using the same computer that does the feedback computations. In that case they compete with these computations for the "attention" of the computer.
4. *Dynamic Identification.* When the dynamics of the controlled plant is unknown or drifting, costly identification and re-calibration algorithms should be applied occasionally in order to update the control law.

In an ideal world of unlimited computational resources[1] (which is where classical control theory evolved, at least its theoretical foundation) one could use as complex control scheme as needed to achieve a desired quality of control. However, in any digital implementation only a *bounded amount of computation* can be performed in any given time interval. When computing power is significantly larger than the complexity of the plant to be controlled (in other words, a slow plant is controlled by a fast computer) one can work with the "separation of concerns" framework which can be summarized as follows:

- The control engineer fixes the control law based on purely "functional" considerations (the quality of control). The outcome of the design is a set of feedback functions, each with its associated rate of invocation.[2]
- It is the responsibility of the hardware/software engineers to meet the implied computational demands on an appropriate computer architecture (processor, I/O interface, scheduling policy). This is done without taking into account the *functional* content of the computations, i.e. their influence on the evolution of the plant. All the implementor knows about are computational tasks with *release times* and *deadlines*.

While this separation of concerns has its advantages (programmers need not know about differential equations and control engineers need not think about computations) it is not so useful when, due to technological and economic constraints, the control of fast and complex plants should be achieved using a bounded amount of computational resources. In such situations computation may become a major bottleneck in the control system

[1] Or equivalently in a world were mathematical functions are viewed as *instantaneous* objects without computation and transmission concerns.

[2] In fact, if you inspect closely the literature on digital control, you don't find a real theory for determining the sampling rate of a control loop, but rather "bandwidth" arguments applied, sometimes, beyond their scope of validity.

and one has to allocate computations in a smart way to meet the conflicting demands coming from different parts of the plant.

Mathematical models of plants and controllers that neglect computational issues are not suitable for computation-conscious design. On the other hand, the opposite approach, that is, the introduction of detailed models of the implementation such as operating systems and scheduling policies, may render the design and analysis of the control system intractable.[3] We present an intermediate approach whose novelty is twofold:

1. It suggests simple abstract models of computations that can be incorporated in the control design process. In these models only one essential feature of computation that is relevant for control is represented, namely *computation time*.
2. As an application we suggest a simple way to derive adaptive scheduling strategies that allocate computational resources to various parts of the plant based on the actual observed performance.

By breaking the "wall" separating control design from its digital implementation, we believe, a much larger part of the space of price/performance tradeoffs can be explored. This belief is apparently shared by other researchers from the control and real-time communities who have recently expressed interest in better mutual interaction between control and computation considerations during the design process, e.g. [SLSS96], [ABE+99], [PAB+00], [SLS+99], [ACV+01].

We present two generic models in which the controlled plant consists of several independent sub-systems which compete for the computational attention of the controller. The first model captures the computational investment in updating the control law of a system that drifts away from its current model while the second model is motivated by the problem of allocating the best feasible mix of sampling rates to sub-systems based on their current performance indices. These models can be easily adapted to other types of resource constraints.

It should be emphasized that the main contribution of this paper is conceptual. We present a new modeling methodology and demonstrate how it can be used to formulate and solve some generic problems of control under resource constraints. Despite the preliminary nature of this work, we believe it can serve as the basis for developing practical algorithms for solving real problems in the future.

2 The Setting

We assume a plant consisting of n independent sub-plants $P_1, \ldots, P_n$, controlled by n independent controllers $C_1, \ldots C_n$. The only dependency between these system components is that all controllers use the same processor to compute their feedback function (see Figure 1).

[3] It can be argued that the type of computer science taught to control engineers is sometimes very detail-oriented and lags behind the more abstract view of computation practiced in (some) computer science quarters.

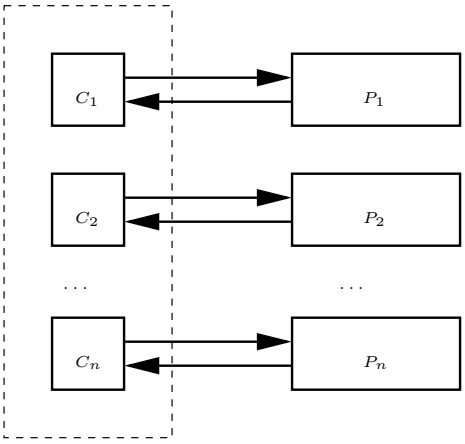

Fig. 1. Controlling n independent plants.

2.1 Controller Performance

The composition of a single plant P with its corresponding controller C yields a closed-loop system $S = P||C$. We assume that we can associate with S a scalar variable x tracking the evolution of some performance index of the system along its trajectories. This measure can be either a memoryless function of the trajectory, such as the distance from a reference point in the state-space, or some average measure of the trajectory over a moving time window. It is important that an evaluation of the current value of x during the actual evolution of the plant can be made from available observations within a bounded delay so that this value can be used for adaptive scheduling.

Our mathematical assumptions concerning this measure are:

1. It lives in $[0, \infty)$ with 0 considered optimal, and our goal is to keep it always inside a bounded interval $[0, m]$.
2. Its evolution is defined by a differential inclusion of the form $\dot{x} \in F(x)$ capturing all possible behaviors under various disturbances.
3. All these behaviors are *bounded from above* by a *worst-case* dynamics which can be characterized by a differential equation $\dot{x} = f(x)$.

When several plants are controlled in parallel the overall performance of the system is represented by $\mathbf{x} = (x_1, \ldots, x_n)$ with $\dot{\mathbf{x}} = f(x)$ where $f(\mathbf{x}) = (f_1(x_1), \ldots f_n(x_n))$. Note that f is "diagonal", that is, the evolution of every x_i depends only on x_i.

In the two models defined below, we assume that the worst-case behavior for each controller for a given allocation of computational resources is *monotone*. In Model I, the worst-case behaviors for all of the controllers is monotone increasing, representing the long-term behaviors of controllers that will always eventually require special attention to re-estimate parameters in internal models or to re-define the control function. The problem is to decide when these re-tuning computations should be made for each

controller. In Model II, the performance of each controller depends on the amount of computational resources (time) allocated to compute the feedback function. For each controller it is possible to allocate sufficient resources to make the performance index decrease (improve), but there are insufficient resources to make the performance indices of *all* the controllers decrease simultaneously. In this case, the problem is to re-allocate the computational resources dynamically to assure that all of the performance indices remain within the specified bounds.

We use the notation $\mathbf{x}_0 \xrightarrow{f,t} \mathbf{x}_1$ to denote the fact that the solution of the differential equation $\dot{\mathbf{x}} = f(\mathbf{x})$, starting from $\mathbf{x}_0$, leads to $\mathbf{x}_1$ at time t. Similarly $\mathbf{x}_0 \xrightarrow{f,t} G$ indicates that the solution reaches some point $\mathbf{x}_1 \in G$. If, in addition, the trajectory stays in $H \subseteq X$ during the interval $[0, t)$ we use the notations $\mathbf{x}_0 \xrightarrow[H]{f,t} \mathbf{x}_1$ and $\mathbf{x}_0 \xrightarrow[H]{f,t} G$, respectively.

We say that a set G is f-invariant if $\mathbf{x} \xrightarrow{f,t} G$ implies $\mathbf{x} \xrightarrow{f,t'} G$ for every t', $0 \leq t' \leq t$ and every $\mathbf{x} \in G$. Note that convex polyhedra are f-invariant when f is constant and that hyper-rectangles are f-invariant when f is monotone.

2.2 Model I: Re-calibrating Controllers

The first model represents the long-term behavior of adaptive controllers: the performance evolves according to $\dot{\mathbf{x}} = f(\mathbf{x})$ where $f(\mathbf{x}) > \mathbf{0}$. Here the only way to avoid divergence and keep the system within a bounded subset is to invest occasionally some time in updating the various controllers. We assume that it takes d_i time[4] to update the controller for plant P_i and that this action *resets* the value of x_i to 0. For every $i \in \{1, \ldots, n\}$ the reset function $R_i : X \to X$ is defined as $R_i(x_1, \ldots, x_i, \ldots, x_n) = (x_1, \ldots, 0, \ldots, x_n)$ and its inverse is $R_i^{-1} : X \to 2^X$.

Definition 1 (Resetting Dynamical Systems). *A resetting dynamical system (RDS) is $\mathcal{A} = (X, f, \mathbf{d})$ where $X = [0, \infty)^n$, $f : X \to \mathbb{R}^n$ is a positive vector field and $\mathbf{d} = (d_1, \ldots, d_n)$ is a vector of delay constants.*

Definition 2 (Update Strategies and Runs). *An update strategy for an RDS $\mathcal{A}$ is a function $s : X \to \{1, \ldots, n, \perp\}$. A run of $\mathcal{A}$ under an update strategy s starting from $\mathbf{x}$ is*

$$\mathbf{x} \xrightarrow{t_1} \mathbf{x}' \xrightarrow{d_i} \mathbf{x}'' \xrightarrow{0} R_i(\mathbf{x}'') \xrightarrow{t_2} \ldots$$

such that $\mathbf{x} \xrightarrow[s^{-1}(\perp)]{f,t_1} \mathbf{x}'$ and $s(\mathbf{x}') = i$.

In other words, a strategy is a rule that tells the system at any given point whether or not to start resetting one of the variables and which variable to reset. A run evolves without resetting as long as $s(\mathbf{x}) = \perp$ until $s(\mathbf{x}) = i$ for some i, then it continues to evolve for d_i time and resets x_i to 0. An example of a run of an RDS in $\mathbb{R}^2$ appears in Figure 2.

[4] In fact, d_i should cover both the time to perform the identification algorithm and the time for some transient behavior when the controller is changed. It is common in timed and hybrid systems models to decompose actions that take some time into time passage followed by an instantaneous transition.

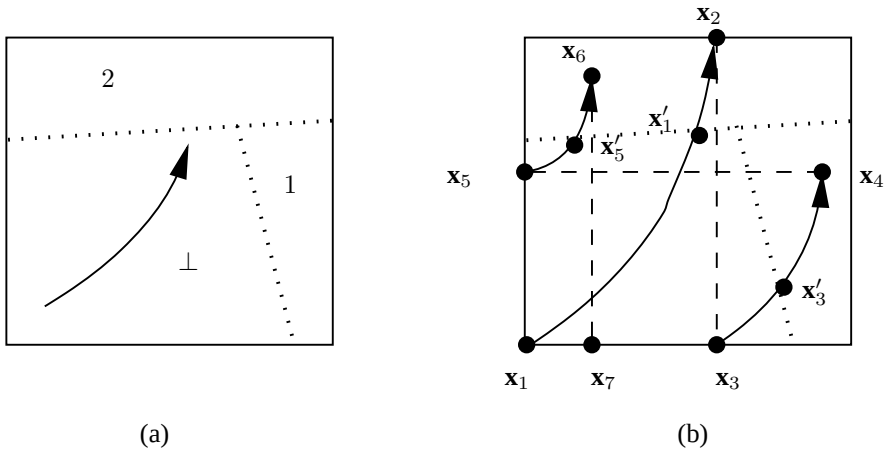

Fig. 2. (a) An update strategy in two dimensions. (b) An initial part of a run following that strategy with resets indicated by dashed lines. The run can be written as $\mathbf{x}_1 \xrightarrow{t_1} \mathbf{x}'_1 \xrightarrow{d_2} \mathbf{x}_2 \xrightarrow{0} \mathbf{x}_3 \xrightarrow{t_2} \mathbf{x}'_3 \xrightarrow{d_1} \mathbf{x}_4 \xrightarrow{0} \mathbf{x}_5 \xrightarrow{t_3} \mathbf{x}'_5 \xrightarrow{d_2} \mathbf{x}_6 \xrightarrow{0} \mathbf{x}_7$.

Definition 3 (Strategy Synthesis Problem for RDS). *Given an RDS $\mathcal{A}$ and a hyper-rectangle $G \subseteq X$ containing* **0***, find the maximal controlled invariant subset of G, that is, the maximal $F \subseteq G$ for which there exists an update strategy $s : F \to \{1, \ldots n, \bot\}$ such that all trajectories starting in F stay in F.*

To compute F and s we use a variant of the fixed-point computation described in [ABD+00]. This approach was first presented for timed automata [MPS95] where it is guaranteed to converge, and then adapted for hybrid automata [W97,TLS99,ABD+00].

Definition 4 (Delayed Predecessors). *Let H be a subset of X and let f be a vector field. The set*

$$\Pi_{(f,d)}(H) = \{\mathbf{x} : \exists t \geq d \, \mathbf{x} \xrightarrow{f,t} H\} \tag{1}$$

consists of all points from which the system can reach H after evolving for at least d time according to the dynamics f.

The relation between H and $\Pi(H)$ is illustrated in Figure 3. Note that we will always work inside a hyper-rectangle G and that Π is distributive over union: $\Pi(H_1 \cup H_2) = \Pi(H_1) \cup \Pi(H_2)$.

The following algorithm computes F:

Algorithm 1 (Winning Update Strategies for RDS)

$F^0 := G$
repeat
  $F^{k+1} := G \cap \bigcup_{i \in \{1,\ldots,n\}} \Pi_{(f,d_i)}(R_i^{-1}(F^k))$
until $F^{k+1} = F^k$
$F := F^k$

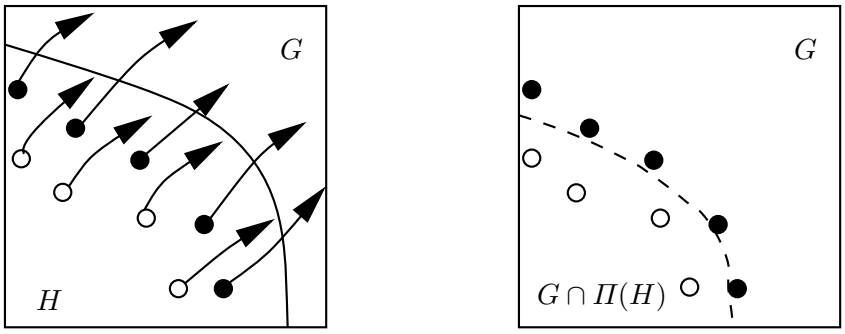

Fig. 3. A hyper-rectangle G, a set $H \subseteq G$ and the set of its delayed predecessors inside G, $G \cap \Pi_{f,d}(H)$. The dark points are outside $\Pi_{f,d}(H)$ because following f for d time leads outside H without being able to return to H.

Claim 1 (Properties of the Algorithm) *Algorithm 1 generates a decreasing sequence of sets $F^0, F^1, F^2, \ldots$ whose limit F is the maximal invariant set of Definition 3.*

Sketch of Proof: Let $d = \min\{d_i\}$. The proof is by showing that $\mathbf{x} \in F^k$ if and only if there is an update strategy such that the trajectory starting at $\mathbf{x}$ stays in G for at least kd time. This is done by induction where F^0 satisfies the property trivially and the inductive step works by showing that $\mathbf{x} \in F^{k+1}$ implies that for some i it is possible to stay in G for some t time, $t \geq d_i \geq d$ and then, after resetting, reach a point in F^k satisfying the inductive hypothesis. ∎

The extraction of an update strategy from F can be done in two steps. First we compute a non-deterministic strategy $\hat{s} : F \to 2^{\{1,\ldots,n,\perp\}}$ such that $i \in \hat{s}(\mathbf{x})$ if $\mathbf{x} \xrightarrow{f,d_i} F$ and $\perp \in \hat{s}(\mathbf{x})$ iff $\mathbf{x} \xrightarrow{f,t} F$ for some $t > d = \min\{d_i\}$. In other words $\perp \in \hat{s}(\mathbf{x})$ if updating is not urgent at $\mathbf{x}$. This non-deterministic controller is the maximal "least-restrictive" controller and any of its functional restrictions, i.e. any function s satisfying $s(\mathbf{x}) \in \hat{s}(\mathbf{x})$ is a winning strategy. Of course, practical considerations such as the description size of the strategy or the number of updates it induces will influence the actual choice of the strategy. For example, one may prefer strategies satisfying $s(\mathbf{x}) = \perp$ whenever $\perp \in \hat{s}(\mathbf{x})$.

The computational issues associated with the implementation of the algorithm and of the update strategy are discussed after the next model.

2.3 Model II: Different Quality Mixes

In this model, a plant P admits a family $\mathcal{C} = \{C^1, \ldots, C^p\}$ of controllers of increasing qualities, that is, for each j, the performance of the closed-loop system $S^j = P||C^j$ is inferior to the performance of $S^{j+1} = P||C^{j+1}$, i.e. $f^j(x) \geq f^{j+1}(x)$ for every x. One possible source of this characterization might be their sampling rates — the family $\mathcal{C}$ might consists of discrete realizations of the same continuous controller with decreasing time steps.

The set of controllers for the whole plant is $\mathcal{C} = \mathcal{C}_1 \times \ldots \times \mathcal{C}_n$ where for every i, the family $\mathcal{C}_i = \{C_i^1, \ldots, C_i^p\}$ of controllers for plant P_i contains *at least one good controller* with f_i negative. Each $C \in \mathcal{C}$ is called a control "mode" of the system, and when composed with the plant, it induces a worst-case dynamics for the performance measure **x**. Our goal, as before, is to maintain **x** inside a bounded rectangle $G \subseteq \mathbb{R}_+^n$, a goal which could be trivially achieved if there were no additional constraints on control modes — we would just stay in a mode such as $(C_1^p, \ldots, C_n^p)$ where all controllers are good. We assume, however, that only a subset $\mathcal{C}' \subseteq \mathcal{C}$ of control modes is feasible due to computational resources constraints and that for every element of $\mathcal{C}'$ there is at least one plant P_i which is controlled badly (positive f_i). Hence there is no mode in which the system can stay forever while maintaining **x** bounded, and the goal should be achieved, if possible, by properly *switching* between feasible modes.

Example: A typical situation that can be modeled naturally in this framework is when each family $\{C^1, \ldots, C^k\}$ is parameterized by the frequency of sampling and computing. Assume that for every plant P_i the computation time of its feedback function is D_i, and that the frequency for each controller is φ_i^j. Hence a mode $C = (C_1^{j_1}, \ldots C_n^{j_n})$ is feasible if it satisfies the Liu and Layland schedulability condition [LL73], i.e.

$$\sum_{i=1}^{n} D_i \cdot \varphi_i^{j_i} \leq 1.$$

If, using the techniques developed in the sequel, it is shown that the plant cannot be controlled by this set of modes, moving to a faster processor will reduce D_i, increase the set of feasible modes, and might make the system controllable.[5] There could be other reasons for certain modes to be infeasible, for example a mode in which two controllers use the same actuator or where the total required power exceeds the available power.

We also assume that it takes up to d time units between the decision to switch and its actual realization. This delay constant should cover the time is takes to compute **x** from observations and for completing some cycles in the schedule corresponding to the current mode before a new schedule is started. The whole situation can now be described using a hybrid automaton whose discrete states correspond to feasible modes.

Definition 5 (Multi-Mode Hybrid Automata). *A multi-mode hybrid automaton (MMHA) is $\mathcal{A} = (Q, X, f, d)$ where Q is a finite set of discrete states (modes), $X = [0, \infty)^n$, $f : Q \to (X \to \mathbb{R})$ is a family of vector fields associated with the modes and $d > 0$ is a delay constant.*

We write $f(q)$ as f^q and assume that for every q and i the sign of f_i^q is uniform all over X.

Definition 6 (Switching Strategies and Runs). *A switching strategy for a MMHA $\mathcal{A}$ is a function $s : Q \times X \to Q$ defined on a subset of $Q \times X$. The trajectory of $\mathcal{A}$ under a strategy s starting from $(q, \mathbf{x})$ is a sequence of the form*

$$(q, \mathbf{x}) \xrightarrow{t_1} (q, \mathbf{x}') \xrightarrow{d} (q, \mathbf{x}'') \xrightarrow{0} (q', \mathbf{x}'') \xrightarrow{t_2} \ldots$$

[5] To avoid confusion with other approaches, we emphasize that we do not care about the implementation of the scheduling *inside* a mode — in this case, since everything is periodic, a static schedule is sufficient in each mode.

such that $\mathbf{x} \xrightarrow[s^{-1}(q)]{f^q,t} \mathbf{x}'$, $s(q,\mathbf{x}') = q'$ *and* $\mathbf{x}' \xrightarrow{f^q,d} \mathbf{x}''$.

In other words, $\mathbf{x}$ evolves for t_1 time following the f_q dynamics without needing to switch until $\mathbf{x}'$ where $s(q,\mathbf{x}') = q'$. After this decision point the system still evolves for d time at q and then switches to q'. An example is depicted in Figure 4 for a MMHA with two variables and two modes such that x_1 diverges in q_1 and x_2 diverges in q_2. We can now formulate the synthesis problem.

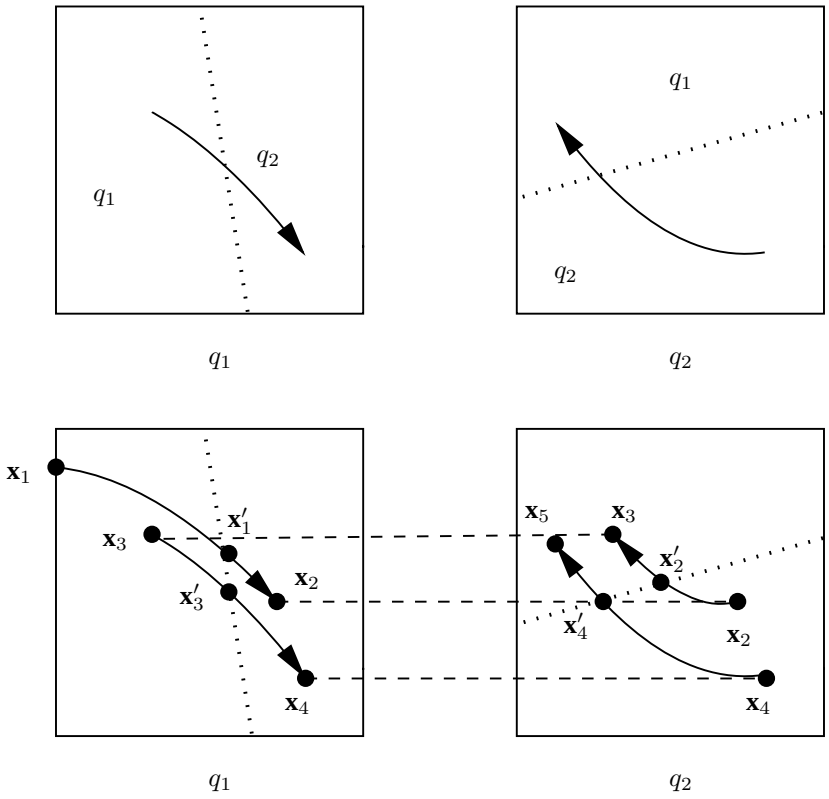

Fig. 4. A switching strategy and an initial part of a run of following this strategy. The run can be written as $(q_1,\mathbf{x}_1) \xrightarrow{t_1} (q_1,\mathbf{x}'_1) \xrightarrow{d} (q_1,\mathbf{x}_2) \xrightarrow{0} (q_2,\mathbf{x}_2) \xrightarrow{t_2} (q_2,\mathbf{x}'_2) \xrightarrow{d} (q_2,\mathbf{x}_3) \xrightarrow{0} (q_1,\mathbf{x}_3) \xrightarrow{t_3} (q_1,\mathbf{x}'_3) \xrightarrow{d} (q_1,\mathbf{x}_4) \xrightarrow{0} (q_2,\mathbf{x}_4) \xrightarrow{t_4} (q_2,\mathbf{x}'_4) \xrightarrow{d} (q_2,\mathbf{x}_5)$.

Definition 7 (Strategy Synthesis Problem for MMHA). *Given a MMHA $\mathcal{A}$ and a hyper-rectangle $G \subseteq X$ containing* $\mathbf{0}$*, find the maximal controlled invariant subset of $Q \times G$, that is, the maximal $\mathcal{F} \subseteq Q \times G$ for which there exists a switching strategy $s : \mathcal{F} \to Q$ such that all the trajectories starting in $\mathcal{F}$ stay in $\mathcal{F}$.*

Note that $\mathcal{F} = \bigcup_{q \in Q}(q, F_q)$ where each F_q is a subset of G. The algorithm for computing $\mathcal{F}$ is the following.

Algorithm 2 (Winning Switching Strategies for MMHA)

$$
\begin{array}{ll}
 & F_q^0 := G \\
 & \textbf{repeat} \\
\forall q \in Q & \quad F_q^{k+1} := G \cap \bigcup_{q' \in Q} \Pi_{(f_q,d)}(F_{q'}^k) \\
 & \textbf{until } F_q^{k+1} = F_q^k \\
 & F_q := F_q^k
\end{array}
$$

Claim 2 (Properties of the Algorithm) *Algorithm 2 generates a decreasing sequence of sets $\mathcal{F}^0, \mathcal{F}^1, \mathcal{F}^2, \ldots$ whose limit $\mathcal{F}$ is the maximal invariant set of Definition 7.*

Sketch of Proof: Similarly to [AMPS98,ABD$^+$00] we show that $\mathbf{x} \in F_q^k$ if and only if it is possible, starting from $(q, \mathbf{x})$, to stay in G for at least kd time. This is true trivially for $k = 0$. The inductive step is based on the definition of Π and on the fact that for every q and k, F_q^k is included in the hyper-rectangle G which f-invariant for every monotone f. Hence, being in F_q^{k+1} implies the ability to stay for at least d time in G and then switch to some q' such that $F_{q'}^k$ satisfies the inductive hypothesis. Hence, starting from a point in the limit $\mathcal{F}$, we can stay in G indefinitely and $\mathcal{F}$ is indeed the maximal invariant set. ⌟

An example of the behavior of the algorithm on a simple system appears in Figure 5.

2.4 Computational Issues

As discussed in [ABD$^+$00], abstract algorithms such as Algorithm 1 and Algorithm 2 can not be effectively implemented in a precise manner for systems where f is arbitrary because the sets of the form $\Pi_{(f,d)}(H)$ may have complex shapes. However, approximate versions of similar algorithms have already been implemented in tools such as $\mathbf{d/dt}$ [ADM01]. Such algorithms can find an under-approximation of the maximal invariant set and the corresponding switching strategy will be safe. The case when f is constant in each mode is discussed in the next section.

As in Algorithm 1, the extraction of a switching strategy from $\mathcal{F}$ can be done by first computing a least-restrictive non-deterministic strategy $\hat{s} : \mathcal{F} \to 2^Q$ defined as

$$\hat{s}(q, \mathbf{x}) = \{q' : \mathbf{x} \xrightarrow{f_q, d} F_{q'}\}.$$

This strategy allows to switch to any q' such that d time after the decision to switch the system will still be inside its safe set for q'. Any functional restriction of $\hat{s}$ is a good switching strategy. Again, due to practical considerations one may prefer strategies with less switches satisfying $s(q, \mathbf{x}) = q$ whenever $q \in \hat{s}(q, \mathbf{x})$.

Once the controller is extracted it can be implemented as a small additional module on top of the digital control system. It monitors the performance index of the systems and switches between the modes according to simple rules. Note that the controller is guaranteed to keep the system in G assuming the *worst-case* dynamics f for $\mathbf{x}$ in each

mode. Due to monotonicity this implies that it will work also for any better admissible behavior of **x**, probably with less switches. We believe the this approach represents a promising direction for adaptive scheduling of digital control systems.

3 Constant Slopes

In this section we study the special case where f^q is constant for each q. This case is interesting for several reasons. First, since we assume that the evolution of each variable is monotone inside a mode, systems with constant slopes exhibit the same qualitative phenomena as the more general ones. The constant derivatives can be interpreted as an upper-approximation of the real f^q or alternatively, when the system is linear, i.e. $f^q(\mathbf{x}) = A_q x$ with A_q a diagonal matrix, looking at $\log x$ we obtain a constant-slope system. From a computational point of view, such systems admit effective computation of successors and predecessors using linear algebra without numerical or symbolic integration and hence the algorithms can be easily incorporated into tools such as HyTech [HHW97], and there is even a hope that they will terminate after a finite number of steps (see below).

Constant slope systems are defined essentialy by assigining a constant vector field $\mathbf{c}^q = (c_1, \ldots, c_n)$ to every mode q. However we need an additional saturation construct to prevent variables from becoming negative.[6] Let $\lceil x \rceil = \max\{0, x\}$ and $\lceil \mathbf{x} \rceil = (\lceil x_1 \rceil, \ldots, \lceil x_n \rceil)$. For every constant c define the function

$$\bar{c}(x) = \begin{cases} c \text{ if } x > 0 \text{ or } c > 0 \\ 0 \text{ otherwise} \end{cases}$$

and let $\bar{\mathbf{c}}(\mathbf{x})$ be its pointwise extension, that is, $\bar{\mathbf{c}}(\mathbf{x}) = (\bar{c_1}(x_1), \ldots, \bar{c_n}(x_n))$. The evolution of **x** in a mode is then defined by an equation of the form $\dot{\mathbf{x}} = \bar{\mathbf{c}}(\mathbf{x})$. We assume further that all slope vectors are integer and that so is d (rational constants can be transformed into integers by changing the time-scale).

When $f(\mathbf{x}) = \bar{\mathbf{c}}(\mathbf{x})$ the definition of the predecessor operator specializes into:

$$\Pi_{(\mathbf{c},d)}(H) = \{\mathbf{x} : \exists t \geq d \ \ \lceil \mathbf{x} + \mathbf{c}t \rceil \in H\}.$$

Claim 3 (Preservation of Polyhedra) *1) If H is a convex polyhedron so is $\Pi_{(\mathbf{c},d)}(H)$. 2) When* **c** *is positive, and H is a hyper-rectangle with integer endpoints, so are $R^{-1}(H)$ and $\Pi_{(\mathbf{c},d)}(H)$.*

Sketch of Proof: 1) Follows from the definition of Π and the elimination of t (this closure of polyhedra under constant-derivative time passage underlies the algorithmic verification of timed automata and "linear" hybrid automata [ACH+95]). 2) Let $\mathbf{c} = (c_1, \ldots, c_n)$ be a positive vector and $H = \bigwedge_i x_i \leq m_i$. To be in $\Pi_{(\mathbf{c},d)}(H)$ one must ensure for every x_i that $x_i + c_i \cdot d \leq m_i$ and hence the operator transforms H into $\bigwedge_i x_i \leq m_i - c_i d_i$. ⌟
As we will see later, the second claim is not true for non-positive **c**.

[6] This is not needed if we use the logarithmic interpreation.

Corollary 1 (Effectiveness and Convergence). *1) The steps of Algorithms 1 (RDS) and 2 (MMHA) can be effectively implemented. 2) Algorithm 1 terminates after finitely many steps and the problem of finding the maximal invariant set under update strategies for constant-slope RDS is algorithmically solvable.*

Sketch of Proof: Effectiveness follows from the definition of the algorithms and the distributivity of Π over union. For convergence, Algorithm 1 performs a monotone iteration over the finite class of sets which can be written as unions of hyper-rectangles with integer endpoints and hence it reaches a fixed-point after finitely many steps. ∎

The problem of finding switching strategies for a MMHA with different slopes in every mode is much more difficult. A first observation is that a necessary condition for having such a switching policy is the existence of non-negative constants $\{\lambda_q\}_{q\in Q}$ such that $\sum_{q\in Q}\lambda_q = 1$ and

$$\sum_{q\in Q}\lambda_q \mathbf{c}^q \leq \mathbf{0} \tag{2}$$

The reason is that if we look at the long term behavior of the system under *any* switching strategy and let λ_q be the average time spent at state q, then the sum in (2) represents the total "average direction" of the system which must be non-positive for the system not to diverge. The condition is not sufficient, though, because it ignores the saturation at zero and the delay. To illustrate the algorithm consider a system in $\mathbb{R}^2$ with two states and slope vectors $\mathbf{c}^1 = (a_1, -b_1)$ and $\mathbf{c}^2 = (-a_2, b_2)$ such that a_1, b_1, a_2, b_2 are positive. We assume $d = 1$ and $G = [0, m]^2$. The necessary condition for controllability is $b_2/a_2 < b_1/a_1$. After the first iteration we obtain:

$$F_1^1 = \begin{array}{l} x_1 \leq m - a_1 \\ x_2 \leq m \end{array} \qquad F_2^1 = \begin{array}{l} x_1 \leq m \\ x_2 \leq m - b_2 \end{array}$$

which expresses the obvious condition that the diverging variables in each state (x_1 in the first and x_2 in the second) do not go beyond m within 1 unit of time. The second iteration gives

$$F_1^2 = \begin{array}{l} x_1 \leq m - a_1 \\ b_1 x_1 + a_1 x_2 \leq (b_1 + a_1)m - a_1 b_2 \end{array}$$

and

$$F_2^2 = \begin{array}{l} x_2 \leq m - b_2 \\ b_2 x_1 + a_2 x_2 \leq (a_2 + b_2)m - a_1 b_2 \end{array}$$

The set F_1^2 is a subset of F_1^1 obtained by removing points from which it is possible to stay in G and switch from q_1 to q_2 but only to points outside F_2^1. These are points where x_2 is larger than $m - b_2$ and cannot go below it before x_1 goes above m, as can be seen by re-arranging the second inequality:

$$\frac{m - x_1}{a_1} \geq \frac{x_2 - (m - b_2)}{b_1}$$

The interpretation of F_2^2 is similar. While for two modes in $\mathbb{R}^2$ the algorithm always converges, the question whether it terminates in higher dimensions is still open.

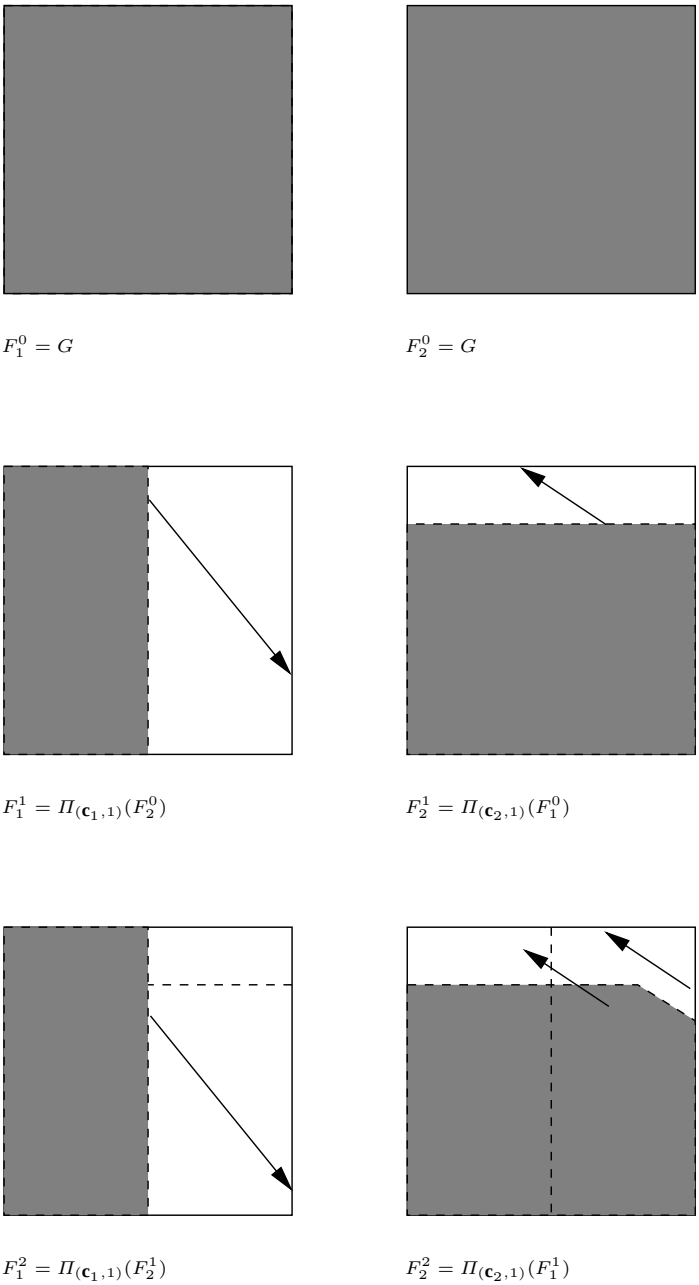

Fig. 5. Applying Algorithm 2 to a two-dimensional system with $m = 10$ and two modes $\mathbf{c}^1 = (5, -6)$ and $\mathbf{c}^2 = (-3, 2)$. We omit from the figure successive applications of the same $\Pi_{(\mathbf{c},1)}$ operator. The inequality $2x_1 + 3x_2 \leq 40$ in F_2^2 guarantees that x_1 will become smaller than 5 before x_2 becomes greater than 10.

It should be emphasized that, although the question of whether $\mathcal{F}$ can be computed *exactly* is an intriguing *theoretical* question about hybrid automata, its relevance for the practical application of the approach presented in this paper is marginal, because we already work with approximative performance measures. Other interesting mathematical questions concerning this model are whether there is an alternative, more "analytical", way to characterize $\mathcal{F}$ and whether some strategies lead to periodic behaviors.

4 Implementation

We have implemented Algorithm 2 for the case of constant slopes. In order to gain insight concerning possible decidability, we have used exact rational arithmetics. We have tried examples with up to 4 dimensions, and in all of them the algorithm terminated. An example of the input file and a fragment of the output for a 3-dimensional system appears below. In this example the maximal invariant set is empty but the iteration goes through rather complex unions of convex polyhedra before converging. The software is available from the authors upon request. We are currently investigating the possibility of implementing the algorithm for more general dynamics by modifying the control synthesis algorithm of $\mathbf{d/dt}$ [ABD$^+$00].

```
Input file:

3             % number of dimensions

10 10 10      % intial X

1             % delay constant

3             % no of modes

3 2 -1        % slope for state 1

2 -3 4        % slope for state 2

-2 4 3        % slope for state 3



Output file:



**** iteration #1 ******

--------- F1 -------------
    x_1  <=  7
 &  x_2  <=  8
 &  x_3  <= 10

--------- F2 -------------
    x_1  <=  8
 &  x_2  <= 10
 &  x_3  <=  6

--------- F3 -------------
    x_1  <= 10
 &  x_2  <=  6
 &  x_3  <=  7

**** iteration #2 ******

[...]
```

```
**** iteration #5 ******
--------- F1 -------------
     3 x_1 +  2 x_2  <= 20
 &   3 x_1 +  2 x_3  <= 20
 &   2 x_1 + x_2  <= 13
 &  x_1 +  3 x_3  <=  9
 &   4 x_2 +  3 x_3  <=  7
 &  x_2 +  2 x_3  <=  3
   <UNION>
    x_1 +  3 x_3  <=  2
 &  x_2 +  2 x_3  <=  7
   <UNION>
     3 x_1 +  2 x_2  <= 14
 &   3 x_1 +  2 x_3  <= 16
 &   2 x_1 + x_2  <=  9
 &  x_1 +  3 x_3  <= 10
 &   4 x_2 +  3 x_3  <= 10
 &  x_2 +  2 x_3  <=  5
   <UNION>
   - x_1  =  0
 &  x_3  <=  1
 &  x_2  <=  8
   <UNION>
     3 x_1 +  2 x_2  <= 16
 &   3 x_1 +  2 x_3  <=  8
 &   2 x_1 + x_2  <=  9
 &  x_1 +  3 x_3  <=  5
 &   4 x_2 +  3 x_3  <= 23
 &  x_2 +  2 x_3  <=  7
   <UNION>
     3 x_1 +  2 x_2  <=  9
 &   3 x_1 +  2 x_3  <= 13
 &   2 x_1 + x_2  <=  5
 &  x_1 +  3 x_3  <= 16
 &  x_1  <=  2
 &   4 x_2 +  3 x_3  <= 27
 &  x_2 +  2 x_3  <= 13
[...]
**** iteration #7 ******
--------- F1 -------------
<EMPTY>
--------- F2 -------------
<EMPTY>
--------- F3 -------------
<EMPTY>
```

5 Conclusions

The models presented in this paper capture natural, if not universal, situations of control under resource constraints: you have to handle many affairs simultaneously but your resources are insufficient for handling all of them properly. The natural solution is to invest your attention in the more burning issues, while letting the others deteriorate a bit, hoping that they will not deteriorate too much until you can pay them attention again. Our models provide methods, based on hybrid automata, for checking whether such a solution is feasible. We hope that such models and techniques will proliferate eventually into engineering practice.

Acknowledgments This work benefited from discussions with Eugene Asarin, Amir Pnueli and Pravin Varaiya and from comments of anonymous referees.

References

[ACV+01] P. Albertos, A. Crespo, M. Vallés, P. Balbastre and I. Ripoll, Integrated Control Design and Real Time Scheduling, unpublished, 2001.

[ABE+99] K.-E. Årzén, B. Bernhardsson, J. Eker, A. Cervin, K. Nilsson, P. Persson and L. Sha, *Integrated Control and Scheduling*, Internal report TFRT-7582, Department of Automatic Control, Lund University, August 1999.

[ABD+00] E. Asarin, O. Bournez, T. Dang, O. Maler and A. Pnueli, Effective Synthesis of Switching Controllers for Linear Systems, *Proceedings of the IEEE*, 88, 1011-1025, 2000.

[ADM01] E. Asarin, T. Dang, and O. Maler, $\mathbf{d/dt}$: a Tool for Reachability Analysis of Continuous and Hybrid systems, *5th IFAC Symposium Nonlinear Control Systems (NOLCOS)*, 2001.

[ACH+95] R. Alur, C. Courcoubetis, N. Halbwachs, T.A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis and S. Yovine, The Algorithmic Analysis of Hybrid Systems, *Theoretical Computer Science* 138, 3–34, 1995.

[AMPS98] E. Asarin, O. Maler, A. Pnueli and J. Sifakis, Controller Synthesis for Timed Automata, in *Proc. IFAC Symposium on System Structure and Control*, 469-474, Elsevier, 1998.

[AW84] K.J. Aström and B. Wittenmark, *Computer Controlled Systems*, Prentice-Hall, 1984.

[HHW97] T.A. Henzinger, P.-H. Ho and H. Wong-Toi, HyTech: A Model Checker for Hybrid Systems, *Software Tools for Technology Transfer* 1, 110-122, 1997.

[LL73] C.L Liu and L. Layland: Scheduling Algorithms for Multiprogramming in Hard Real-Time Environment, *Journal of the ACM* 20, 46-61, 1973.

[MPS95] O. Maler, A. Pnueli and J. Sifakis. On the Synthesis of Discrete Controllers for Timed Systems, in E.W. Mayr and C. Puech (Eds), *Proc. STACS'95*, 229-242, LNCS 900, Springer, 1995.

[PAB+00] L. Palopoli, L. Abeni, G. Buttazzo, F. Conticelli, and M. Di Natale, Real-Time Control System Analysis: an Integrated Approach, *Proc. RTSS'2000*, IEEE Press, 2000.

[SLSS96] D. Seto, J.P. Lehoczky, L. Sha and K.G. Shin, On Task Scheduling of Real-Time Control Systems. *Proc. RTSS'96*, 13-21, IEEE Press, 2000.

[SLS+99] J. Stankovic, C. Lu, S. Son, and G. Tao, The Case for Feedback Control Real-Time Scheduling, *EuroMicro Conf. on Real-Time Systems*, 1999.

[TLS99] C. Tomlin, J. Lygeros and S. Sastry, Controllers for Reachability Specifications for Hybrid Systems, *Automatica* 35, 349-370, 1999.

[W97] H. Wong-Toi, The Synthesis of Controllers for Linear Hybrid Automata, *Proc. CDC'97*, 1997.