Preprints of the 2nd IFAC Conf. on Analysis and Design of Hybrid Systems (Alghero, Italy), 7-9 June 2006

HYBRID SYSTEM CONTROL USING AN ON-LINE DISCRETE EVENT SUPERVISORY STRATEGY

James Millan [*], **Siu O'Young** [**].

[*] *Institute for Ocean Technology, National Research Council, St.John's, NL, Canada*
[**] *Memorial University of Newfoundland, St.John's, NL, Canada*

Abstract: This paper describes a technique for synthesizing controllers for hybrid plants. Our modeling framework allows for the efficient online construction of limited lookahead discrete abstractions of the nonlinear continuous dynamics of the plant model. Discrete event supervisory controller synthesis techniques are used to construct a controller based on a DES specification and the abstracted plant model. The controller is advanced in a moving horizon approach. The modeling framework, synthesis techniques and the on-line computational strategy are discussed. A simple illustrative example is presented in detail and a realistic industrial application is outlined.

Keywords: Discrete-Event Systems, Control, Hybrid Systems, On-line Control, Supervisory Control

1. INTRODUCTION

Theoretical developments in the area of hybrid system control have yet to lead to the widespread solution of any practical industrial problems. While hybrid system modeling is recognized clearly as being central to future control development, the intractability of computations, coupled with a steep learning curve for control system designers, have acted as a barrier to the adoption of hybrid system theory by industry.

A variety of software tools are available for both hybrid system analysis and verification such as HyTech (T.A.Henzinger *et al.* 1997), and *CheckMate* (Chutinan and Krogh 2003). In verification problems, the controller is assumed to be given. Thus, the controller synthesis is a manual process that relies upon the domain knowledge and intuition of the designer. A MATLAB toolbox for simulation and control synthesis for mixed logical dynamical (MLD) hybrid systems in discrete time is available (Torrisi and Bemporad 2004). In the case of simulation tools such as MATLAB Stateflow, the controller design is tested under a variety of conditions to evaluate its safety and correctness. Due to the ad-hoc choice of the test conditions, this technique may miss the particular combination of conditions that leads to design failure. With hybrid verification tools, the computational burden of an exhaustive reachability requires the use of simplified continuous dynamical models. Due to these limitations, the ad-hoc simulation technique is the accepted industry solution, which leaves two problems unsolved: How does the designer take a specification and produce the design? And, how can the resulting design be verified to be correct?

This work attempts to answer both of these questions, by seeking a solution that is familiar to system designers, automates the controller synthesis,

and verifies the resulting controller design. The approach is to harness the power of industrially proven system modeling (simulation tools). The nonlinear continuous models are wrapped in a discrete abstraction layer that is based on event detection. The discrete abstraction combined with a DES specification in a limited horizon reachability computation, produces a discrete event (DE) controller that is, within this limited space and time, locally safe. Furthermore, a reduction in computational complexity is achieved by exploiting a lazy (just-in-time) synchronous composition of the specification and plant models at design time. This scheme is implemented as an online computation, in order for the controller operation to be extended into an infinite time horizon.

Limited lookahead (LL) supervisory control has been extensively studied in a DES setting by (Chung *et al.* 1992). In (Raisch and O'Young 1998), discrete abstractions based on the truncated time history of discrete-time LTI continuous models were used to synthesize DES supervisory controllers. Others, (Su *et al.* 2003) and (Abdelwahed *et al.* 2005), have also used discrete abstractions of switched continuous systems in a LL framework to effect control over hybrid systems. Similar to our work is (Stursberg 2004), in which the nonlinear continuous dynamics are retained as embedded simulations, and a graph search algorithm has been described for optimal hybrid control. Our approach differs in that we do not limit switching to discrete time intervals and controller graph pruning is done in a maximally permissive sense with respect to safety as is commonly done in optimal DES supervisory control (Ramadge and Wonham 1987).

Although the discrete event controller size is exponential in lookahead horizon, our computational approach can significantly reduce the complexity of computing a controller, provided that the specification is sufficiently restrictive, since many trajectories are eliminated during the reachability sweep. On the other hand, an overly restrictive specification may lead to the failure of the online synthesis to find a suitable control solution within the lookahead horizon.

This paper is organized as follows: section 2 develops the continuous system modeling and the associated discrete event abstraction technique; section 3 develops a switched continuous model that provides a means of switching between abstractions of continuous model simulations; section 4 develops controller synthesis in this hybrid framework; and finally in section 5 presents an illustrative controller design process.

2. CONTINUOUS SYSTEM ABSTRACTION

It is desirable to utilize a natural and expressive continuous modeling framework, overlaying it with a discrete event, input/output (I/O) interface. This approach has been adopted by (Koutsoukos *et al.* 2000). For now, we will consider the output aspects of the interface, or the conversion of the continuous dynamics to that of discrete event dynamics.

Let the continuous dynamics of a system be described by a nonlinear ordinary differential equation (ODE),

$$\dot{x}(t) = f(x,t) \tag{1}$$

For now, the dynamics described by equation 1 will serve as a placeholder for the complex continuous dynamics that may be produced by industrial/commercial simulation packages. A solution x of the system modeled by equation 1 on a time interval $[t_0, t_1]$ and for some initial condition is a solution to an initial value problem (IVP). We now define the continuous system model with abstraction framework, as follows:

Definition 1. Let a continuous system model (CSM), s, be defined as a triple $s = (f, \Psi, x_0)$, where:

- f is a Lipschitz-continuous ordinary differential equation, $\dot{x} = f(x,t)$,
- Ψ is a finite set of partitioning functionals, $\Psi = \{F_i : \mathbb{R}^n \to \mathbb{R}, 1 \leq i \leq N\}$, where each F_i is a continuously differentiable functional,
- x_0 is the initial condition, $x(t_0)$.

The set of functionals Ψ, establishes an equivalency, for $x_1, x_2 \in \mathbb{R}^n$:

$$x_1 \sim_p x_2 \iff sign(F_i(x_1)) \times sign(F_i(x_2)) = 1, \quad 1 \leq i \leq N$$

The state space of the CSM is partitioned into a finite quotient set, $\mathcal{X}$ of equivalence classes Q_j (referred to as regions):

$$\mathcal{X} = \mathbb{R}^n / \sim_p = \{Q_j \subseteq \mathbb{R}^n\}$$

A state transition occurs when a continuous trajectory of the CSM crosses the hypersurface $\mathcal{N}(F) = \{x \in \mathbb{R}^n : F(x) = 0\}$, that lies between adjacent regions. For example, a trajectory x on the time interval $[t_0, t_1]$, such that $x(t_0) \in Q_1$ and $x(t_1) \in Q_2$, the state transition from Q_1 to Q_2 is notationally indicated as $Q_1 \rightsquigarrow Q_2$. The CSM communicates with outside discrete event processes via output events $\sigma \in \Sigma_{out}$ that are uniquely associated with these state transitions. Note that the transition direction is important, that is $Q_1 \rightsquigarrow Q_2 \neq Q_2 \rightsquigarrow Q_1$. For any CSM s_i, it can be shown that a unique trajectory x_i exists for a finite time interval and that this trajectory will generate a finite number of output events.

3. SWITCHED CONTINUOUS MODEL

In (Koutsoukos *et al.* 2000), input symbols to the continuous abstraction are translated into actuator actions, or sampled inputs (similar to a D/A converter). In this work, we have chosen to have each input event $\sigma_i \in \Sigma_{in}$ map to the selection (or choice) of a CSM s_i from a set of available continuous system models $\mathcal{F}$. Thus, control is achieved by switching amongst a set of continuous systems that represent either different operating modes of a system or different systems (hot swapping).

Definition 2. Let a switched continuous model (SCM) be defined as an automaton-like triple $G = (\mathcal{F}, \Gamma, s_0)$, where:

- $\mathcal{F}$ is an infinite set of CSMs each with its own discrete abstraction, as in definition 1,
- Γ is the enabled system function, which embodies an implementation specific selection mechanism. Let $s' \in \mathcal{F}$ be the currently selected model, and let $\mathcal{A} = \{a \subset \mathcal{F} : 1 \leq |a| < \infty\}$ be the set of non-empty finite subsets of $\mathcal{F}$, then $\Gamma : \mathcal{F} \rightarrow \mathcal{A}$.
- s_0 is the initial continuous system model.

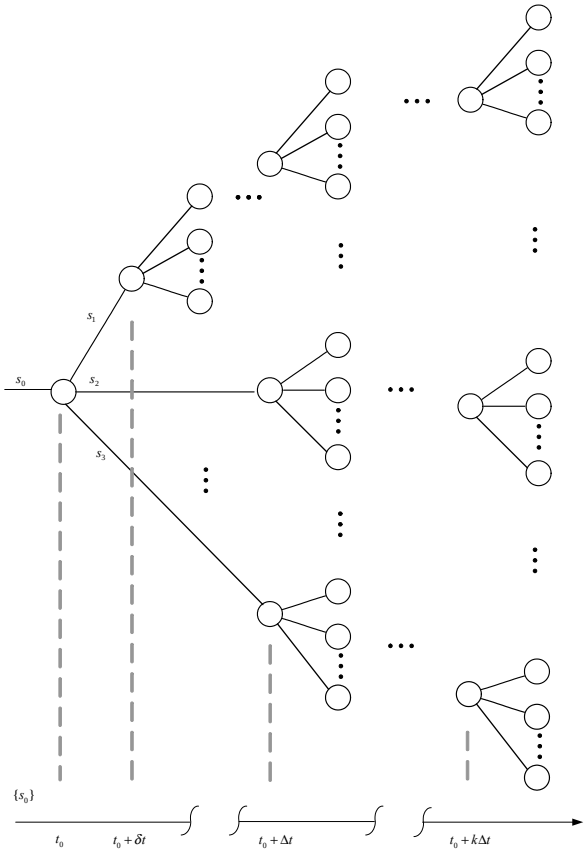

Fig. 1. Reachable continuous system models.

3.1 SCM Execution

An *execution* v of a SCM is a set of sequentially selected CSM, $v = \{s_0, s_1, \ldots s_\alpha \ldots\}$ starting with the initial model s_0. The point in the execution at which the execution changes from one system to another is known as a *choice point*. The term *choice* refers to the ability of the controller at this point to influence the future dynamics of the system, by the selection of the next CSM. In this framework, the choice points occur on some predictable (not necessarily regular) timed schedule, that is governed by a universal timebase. These choice points will be associated with the `tick` (output) event. Additionally, choice points occur whenever an output event is generated by a CSM, so that the controller is able to respond asynchronously to modeled events as they occur. Note that this theoretical modeling framework of the SCM allows both time and state-dependent switching. Because each CSM s_i has its own partitioning set Ψ_i, and dynamics f_i, in the most general case, then our framework admits both partitioning and dynamics changes within a region, due to time dependent switching. This is analogous to the operation of industrial control systems in which a synchronous control cycle is augmented by interrupt-driven control.

3.2 Prediction

Predicting the future execution of an SCM $G = (\mathcal{F}, \Gamma, s_0)$ consists of extending simulations (solutions) for each of the continuous system models, creating new choice points, and again extending the simulations. Thus, the set of future enabled CSMs $\mathcal{S}_R$, can be computed in either a depth or breadth-first reachability. This is the set of CSMs reachable from s_0, or the set of all future executions v of G that originate with s_0 (figure 1). The choice points are identified by the nodes of the graph and the edges are the selected continuous system models. To ensure finiteness of set $\mathcal{S}_R$, it is computed for a limited time lookahead, which can be measured by an integer number p, $p \neq \infty$, of `tick` events. Also, the number of choices (branches at any choice point) is finite and is bounded above by r defined as the maximum $|\Gamma(s_i)|, \forall s_i$. If the number of asynchronous events within one `tick` is bounded above by q, $1 \leq q < \infty$, (a non-zeno condition on the execution) then it can be shown that $\mathcal{S}_R$ is finite and has bounds as follows:

$$pq + 1 \leq |\mathcal{S}_R| \leq \frac{r^{pq+1} - 1}{r - 1},\ r > 1 \qquad (2)$$

3.3 Discrete Event Equivalent

The set of $s_i \in \mathcal{S}_R$ uniquely and concisely describes both the future continuous and discrete event dynamics of the SCM in dense time. Thus, the SCM is capable of generating a language (with timing) that consists of a finite number of finite length strings. Examining figure 1, at any choice point, there are a finite number of choices of CSM. Each choice (or branch) corresponds to a continuous simulation with event detection and each simulation has an associated discrete event equivalent transition defined as follows.

Definition 3. Let $G = (\mathcal{F}, \Gamma, s_0)$ be a SCM and let $s_a \in \mathcal{F}$, $s_a = (f, \Psi, x_0)$ be a CSM. Let $x_a \in \mathbb{R}^n$ be a solution to the IVP posed by s_a on a time interval $t \in [t_0, t_1)$ then let the DE equivalent transition be $\tau_a = (q_0, \sigma_\alpha, \sigma_\beta, q_1)$. Where $q_0 = (x_a(t_0), t_0), q_1 = (x_a(t_1), t_1) \in \mathbb{R}^n \times \mathbb{R}$ are timed continuous states, the endpoints of the solution x_a, and $\sigma_\alpha \in \Sigma_{in}$ and $\sigma_\beta \in \Sigma_{out}$ are discrete events.

The input event σ_α is the selection mechanism or guard event for the transition. The output event σ_β occurs as a result of the transition of the continuous solution into another state.(crossing a hypersurface), or as a result of reaching the end of the designated simulation time interval, Δt, in which case the output event is `tick`. Thus, the input event can be seen as initiating the occurrence of the output event. The set of transitions $\mathcal{T}_R$, corresponding to the set of reachable CSMs $\mathcal{S}_R$, forms a DE transition structure similar to a Mealy implementation of a finite state automaton. The transition structure $\mathcal{T}_R$ also gives rise to a language $L(G)$ based on the output events of the transitions $\tau_i \in \mathcal{T}_R$. Each string $u_i \in L(G)$ such that $p \leq |u_i| \leq pq < \infty$, represents the discrete event behaviour of a particular future execution v_i of the SCM out to the time horizon $p\Delta t$ in the future. The set $\mathcal{T}_R$ (and likewise $\mathcal{S}_R$) is valid for the current state and time only, and must be recomputed (propagated) from control point to control point as the online control is exercised. Clearly, the language $L(G)$ is also valid only for the current state and time as well.

4. CONTROLLER SYNTHESIS

Having established a framework that allows the discrete abstraction of a continuous model to coexist with discrete event models, we will look at the controller synthesis including the computational model.

4.1 Control

The previous section (3.3) outlined how a limited horizon DE representation of the SCM dynamics can be constructed. Let P be a switched continuous model of a plant. And at any point in time and space, there exists a LL plant language where each string $u_i \in L(P)$ is a discrete abstraction of a possible controlled execution in the future (out to some lookahead horizon). The decision of which execution to use must be made based on our knowledge of the plant dynamics represented by this LL model language. A specification can be used to partially implement this decision. Let S be a DE automaton model of a specification, such that $K = L(S)$, the legal language. From a DES perspective, we wish to remove from $L(P)$ any strings that may carry the system to an illegal state. In its simplest form, this can be achieved by taking the reachable part of synchronous product of the plant and specification, $P \parallel S$. Then, by ensuring that only strings that can carry the system to the LL horizon remain in $L(P \parallel S)$, we are left with a nonblocking controller. This problem was studied extensively by (Chung *et al.* 1992).

It should be noted that there is a possibility that no control solution exists, that is the pruned $L(P \parallel S) = \emptyset$, known as a run-time block. While this is a serious problem for an online controller, it is possible to handle it gracefully through the use of special states that represent safe, but non-useful states of the system; essentially an emergency shutdown system, which is consistent with industrial practice (Millan 2006*a*).

If $L(P \parallel S) \neq \emptyset$, then the remaining strings represent the legal traces available to the system. The choice of which particular event (or CSM) that will be selected to advance the system is left to another process or a person (in the event of human in the loop control), to make the final decision. This is the nature of an underspecified system for which control must be implemented (Dietrich *et al.* 2002).

4.2 Complexity

The philosophy of the computational and modeling framework is to compute the controller in an efficient manner, that allows for real-time computation. The DE behaviour of a SCM on a finite lookahead horizon is finite, and so it is therefore computable. Unfortunately, the number of states required to represent this language is bounded above by an expression that is exponential in the event lookahead horizon (see equation 2). However, the number of states is bounded below by a linear expression, implying that a range of poten-

tially sub-exponential complexity computations exist.

The controller state complexity is reduced from the unconstrained plant size by the inclusion of the specification S, at design time (which is also runtime). This is because many unsafe traces are invalidated in the joint behaviour of the plant and specification due to the requirement for synchronization on common events. The extent to which the state complexity will be reduced by the inclusion of the specification is difficult to predict, since it is dependent on the plant and specification models as well as the state and time of the execution. This reduction in complexity due to the inclusion of a specification has been noted for validation of hybrid control systems (Stursberg *et al.* 2003).

The controller graph represents the set of Such a controller merely disables unsafe transitions. As was stated earlier some sort of decision mechanism is still required to choose the actual event (actuation) that will be used to send the system forward in time. We will assume that this decision mechanism is implemented by either another controller module, or possibly even a human in the loop (HIL). In any case, the system will find itself at a new state, and the controller will have to be recomputed before the next decision is taken.

4.3 Encapsulation of Simulation Tools

Suppose there exists a simulation tool that, given an initial condition and some parameters, produces a numerical solution for a particular system model. Then without loss of generality, this is comparable to the ODE solver of the IVP of equation 1. Indeed, our only stipulations on the simulation tool are, given a set of parameters: a) it always produces an output (solution existence), b) the output is repeatable for the same parameters (solution uniqueness) and c) the solution is computed in less time than it takes the actual system to execute (real-time implementation). Whether the latter requirement (c) is met, hinges on the extent to which the specification limits the legal trajectories of the plant. If the simulation tool meets each of the above requirements, then with suitable wrapper functions (object methods), an SCM can be built around it, and an online hybrid controller is feasible.

Based on the computation structure outlined here, an experimental software package that computes DE controllers for hybrid systems, called HySynth has been developed. HySynth was developed as a MATLAB class structure to enable developers to leverage the high-level simulation capabilities of the MATLAB environment.

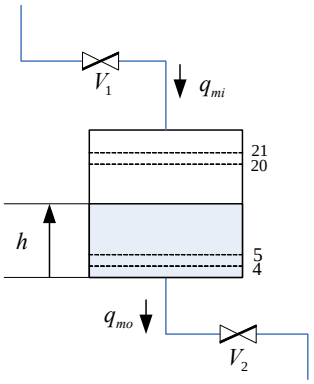

Fig. 2. Tank control schematic, V_1 and V_2 are control valves.

5. EXAMPLES

5.1 Tank Control

In this example, a SCM will represent the model of the tank, which has predominantly continuous dynamics, and a finite state machine will model the specification. Together, the specification and plant models will be used to form a DES controller that can be propagated to enforce the desired behaviour of the tank. In figure 2, the tank has both a fill and drain pipe, which can be controlled independently. If there is turbulent flow from the tank, then the liquid level dynamics are described by the differential equation $\rho A\dot{h} = q_{mi} - q_{mo}$, where h is the liquid level, ρ is the density of the liquid, and A is the cross sectional area of the tank. The liquid mass flow into the tank is q_{mi}. Assuming turbulent flow from the tank, and choosing appropriate values, the resulting nonlinear relation $\dot{h} = q_{mi} - \sqrt{h}$ applies when the tank is draining . If the tank can be switched between filling only, draining only, and simultaneously filling and draining. there are three different CSM dynamics.

$f_1 : \dot{h} = q_{mi}$, filling only
$f_2 : \dot{h} = -\sqrt{h}$ draining only
$f_3 : \dot{h} = q_{mi} - \sqrt{h}$, both filling and draining

Each continuous system model $s_i \in \mathcal{F}$ will contain one of these dynamics. The family of continuous system models $\mathcal{F}$ will be infinite if the initial condition, $h_0 \in \mathbb{R}$ is inherited from the preceding CSM. Table 1 defines a common set of partitioning functionals, shared by all CSMs, and associates a set of output events with them. The specification will be designed so that the tank fills to the overfill mark (21) after one `tick`, draining it back through the overfill 2 mark (20), again after one `tick`, repeating this cycle *ad infinitum*. The finite state machine that represents this specification is given in figure 3. This example was coded in HySynth, and the tank was given an initial level of $h = 20$.

Table 1. Functionals with related output events for the tank control example.

Functional	Alarm	Output Events
$F_1(h) = h - 21$	overfill 1	$of1^+, of1^-$
$F_2(h) = h - 20$	overfill 2	$of2^+, of2^-$
$F_3(h) = h - 5$	underfill 1	$uf1^+, uf1^-$
$F_4(h) = h - 4$	underfill 2	$uf2^+, uf2^-$

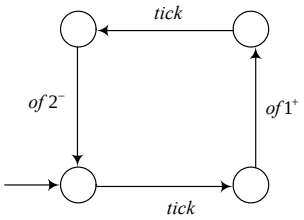

Fig. 3. The tank controller specification.

Using an event horizon lookahead of 15 events, a controller was synthesized repeatedly, propagating the solution by simulation and random selection of controller action. The evolution of the controller structure is illustrated in figure 4(a)-(c); each of the subgraphs in the figure represents the controller at successive time steps. The states are tiny dots connected by edges and the initial state of the graph, i.e. the current state of the controlled system, is indicated by the larger dot. An arrow marks the same state in (a)-(c) and is used to illustrate the growth of the graph ends as the horizon is extended. Any path from the initial state to the end of a branch is 15 events, and represents a safe nonblocking execution of the plant. The past path of the system is indicated by the light line as the initial state moves along. In (b), a choice between the upper and lower left-hand branches must be made. In (c) the lower left-hand branch has been discarded through a choice made by a runtime choice mechanism.

For this example, the number of controller states varied from 51 to 231 over 40 iterations of controller synthesis. Clearly, from equation 2, without the specification, the plant would have a theoretical upper limit of over 21 million states (equation 2, $r = 3$ and $pq = 15$). In this example, the inclusion of the specification at design time has dramatically reduced the computational (state) complexity of the controller. Based on the empirical results of this example, figure 5 clearly illustrates the dramatic improvement in state complexity as a function of lookahead that can be achieved. The upper line is the theoretical state size of the unconstrained plant, while the lower line is a fit through the mean of empirically derived data.

5.2 Oil Offloading

Figure 6 illustrates a control problem that inspired the control techniques that have been developed by the authors. In the offshore oil industry, oil is produced and stored by one vessel, and then transferred to a second vessel that takes it to shore. The goal is to encode the complex and extensive operations manual for this offloading task as a DE specification which will be used to enforce a safe subset of operations for a HIL control. In (Millan and O'Young 2000), the authors developed a simplified version of this problem as a linear hybrid automaton and verified an ad hoc emergency shutdown controller. HySynth has since been used successfully to synthesize a controller for this problem with a complex nonlinear continuous dynamics (Millan 2006*b*).

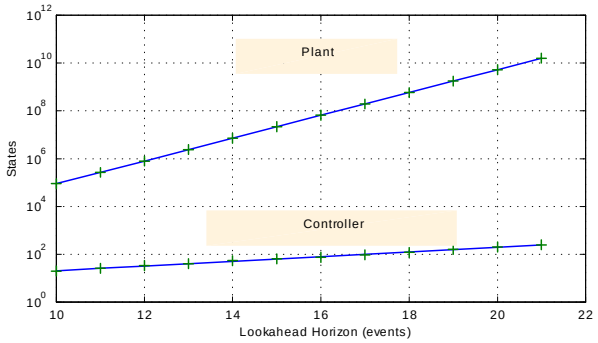

Fig. 5. Comparative complexity of controller and unspecified plant model.

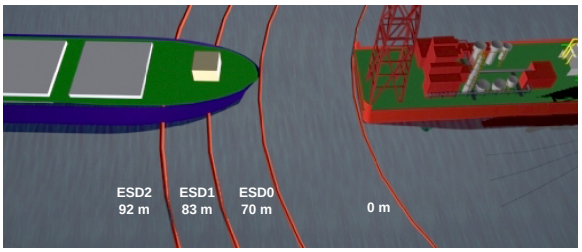

Fig. 6. The offloading of oil from a production vessel (at right) to a shuttle tanker (at left).

6. CONCLUSION

In this paper a switched continuous modeling framework was described that allows generalized nonlinear continuous models to be included seamlessly in a discrete event supervisory control synthesis process. Control is effected by switching between multiple continuous models that may represent either differing operating modes of a system, control inputs or differing systems. Controller nonblocking is identified as the existence of at least one complete string (one that carries the system to the lookahead horizon) in the controller language. A significant reduction in computational complexity is achieved by the inclusion of the specification at design time. An experimental software tool (HySynth) designed for the MATLAB environment, allows for a high level com-

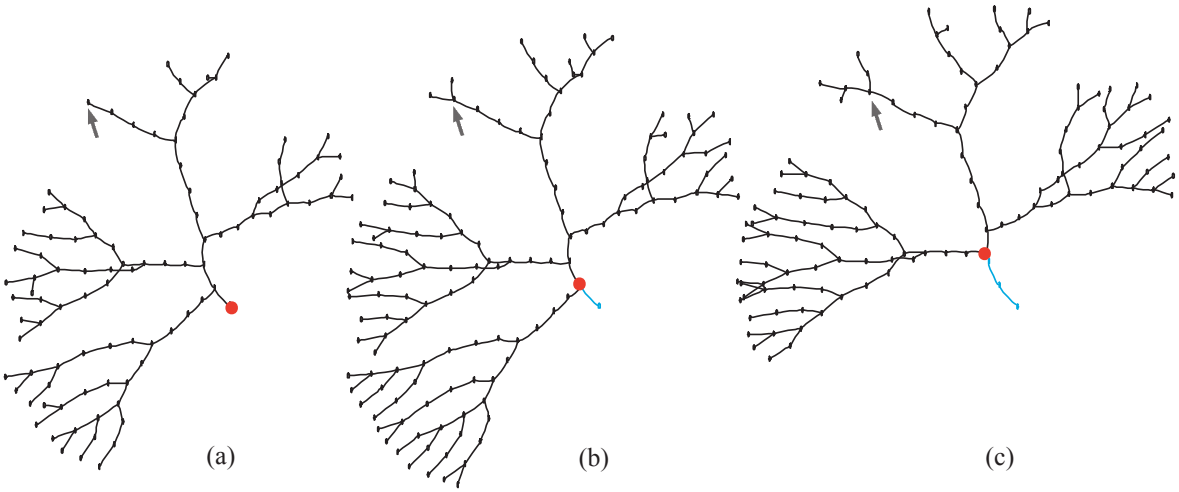

Fig. 4. Propagation of the controller through three updates.

mand interface, coupled with the native MATLAB simulation tools.

Future work will consist of improving the software to allow for enhancing design flexibility and improvements in efficiency. As of now, the control scheme has only been tested in simulation. It would be desirable to put it to a real-world test. Future areas of particular interest are developing techniques to deal with unmodeled effects such as disturbance, and modeling errors.

REFERENCES

Abdelwahed, S., R. Su and S. Neema (2005). A feasible lookahead control for systems with finite control set. In: *Proceedings of the 2005 IEEE Conference on Control Applications*. IEEE. pp. 663–668.

Chung, S., S. Lafortune and F. Lin (1992). Limited lookahead policies in supervisory control of discrete event systems. *IEEE Transactions on Automatic Control* **37**(12), 1921–1935.

Chutinan, A and B. H. Krogh (2003). Computational techniques for hybrid system verification. *IEEE Transactions on Automatic Control* **48**(1), 64–75.

Dietrich, P., R. Malik, W.M. Wonham and B.A. Brandin (2002). Implementation considerations in supervisory control. *Synthesis and control of discrete event systems* pp. 185–201.

Koutsoukos, X.D., P.J. Antsaklis, J.A. Stiver and M.D. Lemmon (2000). Supervisory control of hybrid systems. In: *Proceedings of the IEEE*. pp. 1026–1048. IEEE.

Millan, J. P. (2006*a*). On-line supervisory control of hybrid systems using embedded simulations. In: *Proceedings of the 8th International Workshop on Discrete Event Systems WODES06*.

Millan, J. P. (2006*b*). Online Discrete Event Control of Hybrid Systems. PhD thesis. Memorial University of Newfoundland. Expected July 2006.

Millan, James and Siu O'Young (2000). Hybrid modeling of tandem dynamically positioned vessels. In: *Proceedings of the 39th IEEE Conference on Decision and Control*.

Raisch, J. and S.D. O'Young (1998). Discrete approximation and supervisory control of continuous systems. *IEEE Transactions on Automatic Control* **43**(4), 569–573.

Ramadge, P.J. and W.M. Wonham (1987). Supervisory control of a class of discrete event processes. *SIAM Journal on Control Optimization* **25**(1), 206–230.

Stursberg, O. (2004). A graph search algorithm for optimal control of hybrid systems. In: *Proceedings of the 43rd IEEE Conference on Decision and Control*. pp. 1412–1417.

Stursberg, O., A. Fehnker, Z. Han and B. H. Krogh (2003). Specification-guided analysis of hybrid systems using a hierarchy of validation methods. In: *IFAC Conference on Analysis and Design of Hybrid Systems*. IFAC.

Su, R., S. Abdelwahed, G. Karsai and G. Biswas (2003). Discrete abstraction and supervisory control for switching systems. In: *IEEE International Conference on Systems, Man, and Cybernetics*. Vol. 1. IEEE. pp. 415–421.

T.A.Henzinger, P. Ho and H. Wong-Toi (1997). HyTech: A model checker for hybrid systems. *Software Tools for Technology Transfer* **1**, 110–122.

Torrisi, F.D. and A. Bemporad (2004). HYSDEL - a tool for generating computational hybrid models. *IEEE Transactions on Control Systems Technology* **12**(2), 235–249.