## On-line supervisory control of hybrid systems using embedded simulations

Millan, J.; O'Young, S.



**Publisher's version / Version de l'éditeur:**

*8th International Workshop on Discrete Event Systems [Proceedings], 2006*

# On-line Supervisory Control of Hybrid Systems Using Embedded Simulations

James P. Millan
Institute for Ocean Technology
National Research Council
St. John's, NL, Canada
A1B 3T5
Email: jim.millan@nrc.ca

Siu D. O'Young
Memorial University of Newfoundland
St. John's, NL, Canada
A1B 3X5
Email: oyoung@engr.mun.ca

***Abstract*— This paper describes a technique for synthesizing supervisory discrete event controllers for hybrid plants. The discrete event model of the plant behaviour is assembled for a limited lookahead horizon at run time based on discrete abstractions of embedded continuous simulations. A supervisory controller for this limited horizon model is then computed as a state avoidance problem. Since the controller is valid for only a limited time/event horizon, it must be re-computed on-line periodically, in order to extend the operation to an infinite time horizon. We present a scheme for enforcing guaranteed safe and nonblocking behaviour which is rooted in standard industrial practice.**

## I. Introduction

Obtaining a discrete event model for a system can be a daunting task for a control system designer for a variety of reasons. In general, many industry-standard simulation and modeling tools are based on continuous dynamical simulations, designers are unfamiliar with discrete event modelling techniques. When producing DE models based on a continuous model, the designer must face practical choices with regard to communications/synchronization (event set) and the quantization (state set). The results of these decisions will directly affect the complexity and non-determinism of the resulting model. And, once the model has been designed, is it a suitable level of abstraction of the continuous dynamics? Too complex and it is not feasible to compute a controller. Too abstract and the resulting controller will be too conservative. A further impediment to DE control implementation is the requirement that the controller be computed offline. Such a controller cannot deal with time varying conditions. Furthermore, the offline controller is impractical to compute for all but the most trivial of system models due to the exponential growth in the number of model states when plants are composed from parallel communicating automata.

The modeling and control synthesis approach described in this paper addresses the above issues. Firstly, a modeling scheme is used that enables the embedding of industrial continuous simulation tools by wrapping them in a discrete abstraction layer. This allows the designer to use models with a fit-for-purpose level of fidelity and to utilize tools they are most familiar with to synthesize a DE supervisor for a limited-horizon discrete event model of the continuous dynamics. In [1], the discrete event (DE) model was a discrete-time LTI continuous model combined with a discrete abstraction based on a truncated history of the input and output events. The modeling framework presented in this paper is a hybrid system model that uses a generalized discrete abstraction based on continuous functionals. It allows for switching of the continuous dynamics, which is more closely related to the work of [2] and [3] in which discrete abstractions of switched continuous systems have been set in a limited lookahead framework. Within the hybrid model framework, this paper takes the approach of synthesizing an online DES supervisor that is guaranteed nonblocking and safe but which is more restrictive than the standard limited lookahead policy (LLP) DES supervisors of [4].

The organization of the paper is as follows: section II deals with the modeling framework, section III details the limited lookahead controller synthesis, section IV introduces the object-oriented implementation scheme for control synthesis, and section V gives an illustrative example.

## II. Modeling Framework

In order to embed continuous simulations in our control framework, it is necessary to develop a generic way of discretizing the dynamics. This paper uses the same approach as [5], in which a discrete abstraction of a continuous model is obtained by partitioning the continuous state space with smooth continuous functionals. Discrete output events $\Sigma_{out}$ (called plant events in [5]) are then associated with the boundaries (hypersurfaces) between partitioned sets, so that there is an output interface (A/D) that allows the continuous model to communicate (synchronize) with DE processes.

### A. Continuous System Model

In this paper, the dynamics of a nonlinear ordinary differential equation $\dot{x} = f(x,t)$ will be used as a placeholder for the dynamics of the embedded continuous simulations. 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, $\dot{x} = f(x,t)$,

$\Psi$ is a finite set of partitioning functionals, $\Psi = \{F_i : \mathbb{R}^n \to \mathbb{R}, 1 \leq i \leq a\}$, where each $F_i$ is a continuously differentiable functional,

$x_0$ is the initial condition, $x(t_0)$.

A continuous trajectory of the system $x$ on a time interval $\Delta T = [t_0, t_1]$ for some initial condition is a solution of an initial value problem (IVP). With $f$ Lipschitz continuous, $x$ exists and is unique. If $x$ crosses a given hypersurface of functional $F(x)$, $\mathcal{N}(F) = \{x \in \mathbb{R}^n : F(x) = 0\}$, then this is considered as a discrete state transition. The state space of the continuous model is partitioned into a finite quotient set $\mathcal{X}$ of equivalence classes $Q_j$

$$\mathcal{X} = \{Q_j \subset \mathbb{R}^n : a + 1 \leq j \leq 2^a\} = \mathbb{R}^n / \sim_p$$

where the equivalence relation for $(x_1, x_2) \in \mathbb{R}^n \times \mathbb{R}^n$ is defined as

$$x_1 \sim_p x_2 \iff \operatorname{sign}(F_i(x_1)) \times \operatorname{sign}(F_i(x_2)) = 1, \ \forall i, \ 1 \leq i \leq N$$

Each equivalence class $Q_j \in \mathcal{X}$ may then be associated with a discrete state label, through a state labeling function.

### B. Switched Continuous Model

In[5], discrete event processes communicate with a continuous model's dynamics via input (actuator) symbols $\Sigma_{in}$. Input events are mapped to sampled inputs $u_k \in \mathbb{R}^m$ which are applied to the continuous system model $\dot{x} = f(x, t)$. In this work, the input events will switch between various continuous system dynamics.

*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 a set of CSMs, possibly infinite, each with its own discrete abstraction as in definition 1,

$\Gamma$ is the enabled system function, that embodies a 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 $\Gamma : \mathcal{F} \to \mathcal{A}$.

$s_0$ is the initial continuous system model.

An *execution* of a SCM is a sequence of selected continuous system models starting with the initial CSM, $v = \{s_0, s_1, \ldots, s_i, \ldots\}$. 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 from a finite set of possible future CSMs $\Gamma(s) \in \mathcal{A}$. 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 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 $\Psi_i$, 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.

### C. Discrete Event Behaviour

The future continuous and discrete event behaviour of the SCM $G = (\mathcal{F}, \Gamma, s_0)$ can be predicted by recursively constructing a set of reachable continuous system models $\mathcal{S}_R$, using the enabled system function $\Gamma$. At each choice point, the future execution of the system branches. The number of choices (branches at any choice point) is finite and is bounded above by $r$ which is defined as the maximum $|\Gamma(s_i)|, \forall s_i$. Let the finite lookahead time horizon for the prediction be $p$ `tick` events. If the number of inter-tick (asynchronous) events is bounded above by $q$, $q < \infty$, which is a condition of nonzeno execution, then it can be shown that the cardinality of the set of reachable system models is as follows

$$|\mathcal{S}_R| \leq \frac{r^{pq+1} - 1}{r - 1}, \ r > 1 \tag{1}$$

If the number of inter-tick events is infinite, these traces should be deemed to be unsafe and will be eliminated at runtime.

For each $s_i \in \mathcal{S}_R$ there exists a corresponding unique continuous solution or trajectory $x_i$ (again from Lipschitz continuity). From the reachable set of continuous system models, we can construct a discrete event graph.

*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 $\sigma_\alpha$ is the selection mechanism or guard event for the transition. The output event $\sigma_\beta$ occurs as the result of a continuous trajectory crossing a hypersurface in the SCM, or as the result of reaching the end of the designated simulation time interval, $\Delta t$, in which case the output event is `tick`. The input event initiates a continuous simulation which leads to the occurrence of the output event. Thus, there exists a transition $\tau_i$ for each $s_i \in \mathcal{S}_R$.The tree composed from the set of all discrete event transitions $\tau_i \in \mathcal{T}_R$ concisely captures the discrete event behaviour of the SCM on a limited time lookahead horizon. A constant time lookahead in our framework admits the possibility that each branch of the tree from root to the lookahead horizon will be of differing lengths. Therefore the language $L(G)$ generated by the transition structure $\mathcal{T}_R$ is a set of finite length strings $u_i \in L(G)$ such that $p \leq |u_i| \leq pq < \infty$.

### D. Controllability

In the DES supervisory control theory of Ramadge and Wonham (RW) [6], the plant is modeled as an automaton

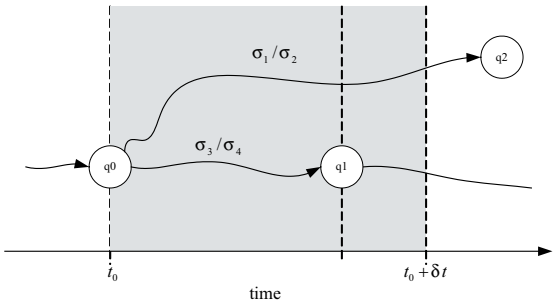


Fig. 1. Implementation dependent uncontrollability.

having discrete events that are divided into controllable events $\Sigma_c$, and uncontrollable events $\Sigma_u$. A controller is synthesized for a given plant so that the desired legal behaviour is enforced by disabling only controllable transitions. An example of an uncontrollable event in a timed DES plant is the `tick` event, since time cannot be halted by the supervisor. In contrast to the RW framework, the notion of controllability in the SCM framework is rooted partially in the implementation details of the controller, and partially in the plant model. In the SCM framework, the output events each occur as the result of a control action (the assertion of an input event). The choice of this control action is made at a choice point. Therefore, an uncontrollable event is interpreted as one that must occur *when there is no other choice available*. For example, the availability of controller actions at a particular choice point may be restricted by another component of the plant model so that only one event is available as a choice. This is termed plant-dependent uncontrollability, and the corresponding output event is a plant uncontrollable event (PUE). The other contributing factor to uncontrollability is implementation-dependent uncontrollability. This is due to the situation illustrated in Figure 1, in which an event occurs so close in time after a control action has been taken, that there is no time for the controller to act again in response to the event. These events will be termed implementation uncontrollable events (IUE). In the figure, the curved lines represent the continuous simulations that give rise to the discrete output events. The upper transition $(q_0, \sigma_1, \sigma_2, q_2)$ is a controllable transition, since it falls outside of the controller reaction time $\delta t$, while the lower transition $(q_0, \sigma_3, \sigma_4, q_1)$ is uncontrollable. In the SCM framework, uncontrollable events *must* occur, since it is clear in this situation that the uncontrollable event will preempt the controllable one. Therefore, for absolute safety, potentially unsafe trajectories must be prevented even though they may not occur. The controllable status of any particular event may vary as a function of both space and time. Fortunately, the occurrence of PUEs and IUEs is a modeled effect.

## III. Controller Synthesis

A significant body of work exists with respect to forming online DES supervisors in a limited lookahead framework (called LLP for limited lookahead policy), including [4] and [7]. The general approach with LLP control is to compute the controller based on the $N-$event truncated discrete event behaviour $L_N(G)$. The set of strings that make up this language are called the *pending* strings. The pending strings that are unambiguously illegal are first removed. Next, different attitudes can be adopted when deciding which of the remaining pending strings will be retained in the controller language. Taking a *conservative* attitude assumes that the next event after each pending string is illegal, and taking an *optimistic* attitude assumes that the next event is legal. If all strings are removed from the lookahead language $L_N(G) = \epsilon$, then this is considered a *run-time error*. Numerous variations on the basic LLP control have been proposed including variable lookahead with state information [8] and extension of traces beyond the lookahead horizon [9].

### A. Safety and Nonblocking

For the online controller proposed in this paper, a run-time error (or controller block) would be catastrophic. In the absence of a legal control choice, a real system must continue (since time cannot be stopped), and since only illegal choices remain, it will be forced to do so with a control action that ultimately violates the system safety. In this paper's formulation of the limited lookahead control problem, $blocking \implies unsafe$. To address this issue, we have taken a design practice from standard industrial control, the concept of an emergency shutdown (ESD) mechanism and we have incorporated it into our theoretical framework.

*Definition 4:* An emergency shutdown state is a safe, idle state.

An emergency shutdown is undesirable from the point of view that it performs no useful work, but it is preferable to a potentially catastrophic safety violation. Therefore, the ESD state is a way of gracefully handling the blocking scenario. We now define a new LLP attitude known as *ultra-conservative.* This is an implementation of limited lookahead policy control using additional state information. The supervisor, in addition to assuming a conservative stance towards the pending traces now has the added requirement that it should enforce controllable behaviour that permits an ESD state to be reached within $N$ or fewer events. Initially, if such a trace cannot be found, the controller does not exist; in [4] this was termed as a *starting error*. If the controller can be started, then it will always be possible to drive the system to an ESD state as a last resort at any point during the runtime.

To illustrate this control concept, consider the following control synthesis example.

*Example 5:* Let the simple lookahead tree with a 2 event horizon in Figure 2 represent the safe pending behaviour of a SCM (illegal traces have already been trimmed from the tree). Since the tree has been generated from the embedded continuous simulations, the state information is available. Controllable events are indicated by graph edges with an arrowhead, while uncontrollable events are indicated by edges with a small circle at the end. All states are legal, with the exception of state 10, which is a ESD state (shaded grey). Taking an optimistic attitude, it is assumed that the extension

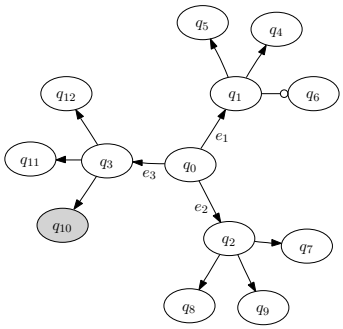


Fig. 2. Limited lookahead tree with emergency shutdown state.

of each trace by one event leads to a legal state. As a result, each subtree is legal in the graph and events $\{e_1, e_2, e_3\}$ are all legal controller actions. Taking a conservative attitude, the assumption is that all of the pending traces will lead (uncontrollably) to an illegal state in the next event, in which case the subtree with the uncontrollable transition from 1 to 6 must be disabled. Thus, the legal controller actions are $\{e_2, e_3\}$. Finally, adding the requirement that the system must be ESD-state coreachable, leads to the further trimming of the tree as the otherwise controllable and safe subtree of $e_2$ is no longer valid since it is uncertain whether an ESD state can be reached. Thus, the legal controller action is $\{e_3\}$.

The additional requirement of ESD state coreachability results in the a more restrictive control than that of previous LLP literature. The cost of ensuring non-blocking safe operation within an arbitrary lookahead horizon is a more restrictive controller.

## IV. Implementation Framework

This section outlines briefly the implementation of our control synthesis and modeling framework. A software package has been developed that computes DE controllers for hybrid systems, called HySynth. Developed as a MATLAB class structure, HySynth enables the user to leverage the high-level simulation capabilities of the MATLAB environment. This section presents a brief overview.

### A. Encapsulation of Simulation Tools

Suppose there exists a continuous simulation tool that, given an initial condition and some parameters, produces a numerical solution for a particular system model. Then under what conditions is it comparable to the ODE solver? The simulation tool, when given a set of parameters: a) must always produce an output (solution existence), b) the output must be repeatable for the same parameters (solution uniqueness) and c) the solution must be 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 a 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.

### B. Object Oriented Models

For efficient controller computation, the software implementation of a SCM must instantiate certain methods. These methods must allow for synchronous composition and graph exploration of models in a DE environment. Borrowing the concepts of inheritance from software engineering, we will define the abstract class *depObj* (discrete even process object), from which there are two derived classes: the *FSM* class and the *SCM* class. Both of these classes instantiate a set of common methods listed for the *depObj* class. Algorithm 1 demonstrates the *SCM* class method that computes the discrete enabled events for a SCM $G$ with some implicit control update interval, $\Delta t$. In this algorithm, the ODE solver with event detection of line 5, returns the matching discrete event once the solution crosses a hypersurface, or `tick` otherwise. Any industrial simulation tool with discrete event generation can be regarded as an implementation of an ODE solver. Thus, in practice, lines 4-7 of the algorithm represent the interface of the lookahead control strategy to the simulation package. The set of enabled discrete events is constructed by performing $|\Gamma|$ (or fewer) simulations. In this way, no *a priori* DE model is required for the discrete dynamics, since they are computed just in time (lazy computation). We now define the *product* class,

**Algorithm 1**: *nextEvents* method of the *SCM* class

```
1 Function nextEvents(G)
2 nextEventSet ← ∅
3 foreach s_i ∈ Γ do
4 |   solution of ODE posed by s_i on time interval Δt
5 |   σ ← solveODE(s_i,Δt)
6 |   nextEventSet ← nextEventSet ∪ σ
7 end
8 return (nextEventSet)
```

a polymorphic container for objects of class *depObj*. The *product* class also derives from *depObj*, so we can create hierarchical models. Associated with the *product* class is a product state, the *pstate* class.

The first step in the controller synthesis is to form the legal LL plant language. This is based on a depth-first, limited horizon reachability algorithm on the product $P$ (product object), from the product's present state, $ps$ (pstate object), to an integer event horizon of $rd$ events.

If $P$ holds a plant model as a SCM and a specification as one or more finite state models, then this function returns a boolean, with the value *false* indicating that the legal plant language is blocking. If the reachable transition structure is nonblocking, then it is passed to a function that prunes branches according to the ultra conservative attitude. The function ensures that the remaining controller graph is ESD state coreachable.

## V. Example: Ship Control

In the offshore oil industry, oil is produced and stored by a FPSO (Floating production, storage and offloading)

**Algorithm 2**: *reach* method of the *product* class

```
1  Function productReach(P,ps,rd)
2  if rd ≤ 0 then
3  |  return (true)
4  end
5  nonBlocking ← false
6  enabledEventSet ← nextEvents(P,ps)
7  foreach σ ∈ enabledEventSet do
8  |  nextStateSet ← nextStates(P,ps,σ)
9  |  foreach ns ∈ nextStateSet do
10 |  |  temp ← productReach(P,ns,rd-1)
11 |  |  if temp then
12 |  |  |  tranSet ← tranSet ∪ [ps, σ, ns]
13 |  |  end
14 |  |  nonBlocking ← nonBlocking ∨ temp
15 |  end
16 end
17 return (nonBlocking)
```

vessel, and then transferred to a second vessel (tanker) that takes it to shore. In this application, the transfer operation is a very dangerous and complex operation that is largely automated (from a continuous perspective) while most of the decision making (discrete events) is carried out by human operators. Within the limits of the available power, actuator thrust and other process considerations, a control system must coordinate the two vessels and prevent a collision from occurring. Ultimately, we wish 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; this task is currently carried out by human operators. In [10], the authors developed an offline controller for this task that was verified with the HyTech [11] software package. By necessity, the entire system was modeled with linear hybrid automata that approximated the nonlinear continuous vessel dynamics. An ad-hoc controller was developed that enforced a safe distance between the two vessels.

### A. Model Formulation

For this example, we design a DES supervisor based on the techniques described in §II and §III. We will embed a simulation of the closed loop dynamics of all major systems. We also model the power management system (PMS) which controls the power generation and distribution functions of the vessel. The coordinating controller that we will design controls the vessel movement by commanding the DP system and the PMS of both vessels. In this example, the controller coordinates a move of the tanker to avoid deck overheating due to natural gas flaring on the FPSO (Fig. 3). The controller may shut down the oil transfer and move to the safe distance $r > r_{sd}$ from the FPSO if its deck temperature rises too high. The coloured area denotes the flare-safe area.

We begin by designing a discrete abstraction layer based on functionals in order to extract discrete event information

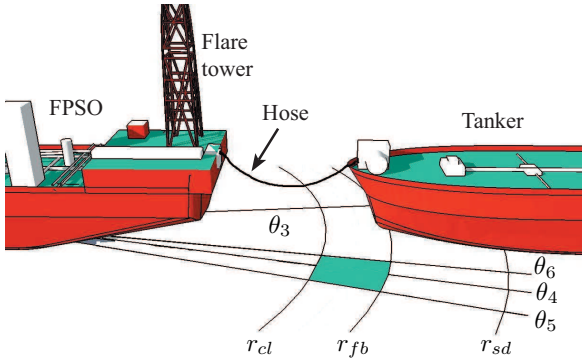


Fig. 3. Tanker and FPSO offloading oil.

TABLE I
OUTPUT EVENTS, WITH ASSOCIATED FUNCTIONALS.

| $\sigma_{out}$ | Functional | Dir | Alarm |
|---|---|---|---|
| *tcl* | $F_1(x) = r - r_{cl}$ | $\downarrow$ | too close to FPSO |
| *tfb* | $F_2(x) = r - r_{fb}$ | $\uparrow$ | too far from FPSO |
| *o3* | $F_3(x) = \theta + \theta_3$ | $\uparrow$ | riser area guard |
| *o4* | $F_4(x) = \theta + \theta_4$ | $\downarrow$ | enter flare safe area |
| *o5* | $F_4(x) = \theta + \theta_5$ | $\downarrow$ | cw exit flare safe area |
| *o6* | $F_4(x) = \theta + \theta_6$ | $\uparrow$ | ccw exit flare safe area |
| *tfp* | $F_7(x) = \pi - (\psi - \theta) - 0.2$ | $\uparrow$ | misalignment to port |
| *tfs* | $F_8(x) = \pi - (\psi - \theta) + 0.2$ | $\downarrow$ | misalignment to stbd. |
| *esd* | $F_9(x) = r - r_{sd}$ | $\uparrow$ | emergency shutdown |
| *tick* | $F_{10}(x) = \sin(2\pi t/\Delta t)$ | $\downarrow$ | controller update $\Delta t$ |

from a nonlinear ship simulation. In Table I each output event $\sigma_{out}$ is associated with a functional and a hypersurface crossing direction (Dir). The simulation (partial) state vector is $x = [r, \theta, \psi, t]^T$ where $r, \theta$ is the position of the tanker in polar coordinates, $\psi$ is its heading angle and $t$ is the simulation time (also see Fig. 3).

A SCM for this system is created by wrapping this discrete abstraction around a full nonlinear ship simulation, including the DP control system model; this becomes the basis of the set of continuous system models $\mathcal{F}$. Control actions available to the DES supervisor are tabulated in Table II; commands are issued to the tanker DP system as "jog" (relative) position commands $r_{jog}, \theta_{jog}, \psi_{jog}$, Vector $g \in \{0,1\}^3$ is the generator demand vector, a command to the PMS to switch on/off main generator 1 and 2 and standby generator respectively. The PMS and the closed-loop DP controller ensure that these commands are carried out. These control actions represent the set of continuous system models $s_i \in \mathcal{F}$ which the controller will switch amongst. During execution, each CSM inherits its simulation time and initial condition $x_0$, from the previous simulation. Without loss of generality, we use a common partitioning $\Psi$, based on the functionals of Table I.

The switched continuous model we have created here is the plant model. The specification is modeled as a finite state automaton, and is designed to ensure the safety of the system once a flare event has initiated (see Fig. 4).

This specification requires an $o4$ event (entry to the flare-safety area) to occur before 6 $ticks$.

TABLE II

VESSEL CONTROLS AND INPUT EVENTS.

| | Controls | | | | |
|---|---|---|---|---|---|
| $\sigma_{in}$ | $r_{jog}$ | $\theta_{jog}$ | $\psi_{jog}$ | $g$ | Description |
| $\alpha_1+$ | 0 | 0.1 | 0 | $[1,0,0]$ | jog cw |
| $\alpha_1-$ | 0 | $-0.1$ | 1 | $[1,0,0]$ | jog ccw |
| $\alpha_2+$ | 0 | 0.15 | 0 | $[1,1,0]$ | jog cw |
| $\alpha_2-$ | 0 | $-0.15$ | 1 | $[1,1,0]$ | jog ccw |
| $fwd$ | 1 | $-0.1$ | 0 | $[1,0,0]$ | ahead |
| $back$ | 1 | 0.1 | 1 | $[1,0,0]$ | astern |
| $hold$ | 0 | 0 | 0 | $[1,0,0]$ | hold station |
| $sd$ | $1.85^{\dagger}$ | ‡ | ‡ | $[1,0,1]$ | ESD |

† in absolute coordinates; ‡ indicates a don't care input

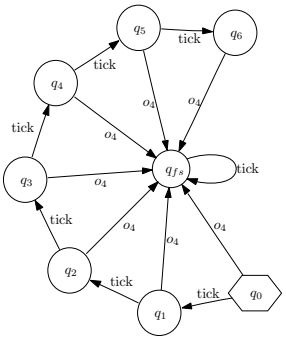


Fig. 4. The specification requires the tanker move to safety in $\leq 6$ ticks.

### B. Controller Synthesis

For this scenario, the online controller is based on a fixed-event lookahead of 6 events (although a time horizon could also be utilized) of the synchronous product of the plant and specification models. The controller graph is computed at each choice point and then pruned according to the ultra-conservative rule. If multiple possible control actions remain, then the choice of control action, i.e. which $s' \in \Gamma(s)$ to select, is made by some choice mechanism; in this case, a human operator. The operator is given a maximum time $\delta_t$ to select the control action from the ESD safe actions; if no human decision occurs, an algorithmic selection process will make the decision. Thus, plant-generated events occurring less than $\delta_t$ after a choice point are PUE, otherwise all $\sigma_{out} \in \Sigma$ are considered controllable. A run is pictured in Fig. 5 in which the vessel traverses safely to the flare-safe area. During this run, the controller required the operator to start a second generator ($\alpha_1^-$ to $\alpha_2^-$) at $t = 300$ in order to safely complete the move. During this sequence, the controller transition structure varied considerably $100 \leq |\mathcal{T}_R| \leq 5000$. Controller complexity is significantly reduced by using the specification to help prune the graph online [12].

## VI. CONCLUSION

In summary, we have proposed a limited lookahead control strategy for a hybrid system in which the dynamics are derived from industrial-strength simulation tools. The controller is synthesized online in response to events that may occur either due to continuous time or state. At each step the controller is a unique set of optimally controlled sub-trees such that each sub-tree can be safely extended

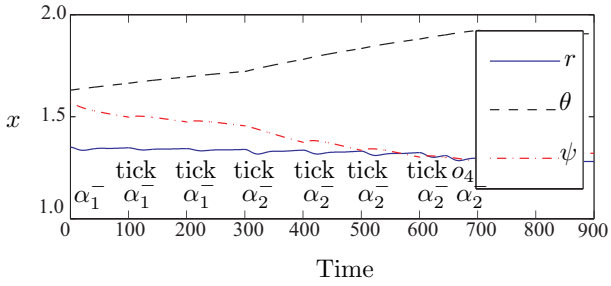


Fig. 5. Plot of tanker state vector $x$ during HIL controlled run.

to the next look-ahead step without blocking. The non-blocking property relies on additional state information of the plant and the coreachability of ESD states. The system is operated on the basis that if there are no other viable choices in the future horizon, the system can at least be safely shut-down within the limited look-ahead horizon. Our solution is more conservative than the typical conservative LLP because we treat all unexplored traces as temporarily unsafe, with the additional requirement that the plant must be able to be driven into shut-down if necessary. Finally, we have synthesized a controller and tested it using the HYSYNTH software. The example demonstrates the ability of our framework to use embedded simulations and a HIL scheme to implement control choice.

## REFERENCES


[1] J. Raisch and S. O'Young, "Discrete approximation and supervisory control of continuous systems," *IEEE Transactions on Automatic Control*, vol. 43, pp. 569–573, April 1998.

[2] R. Su, S. Abdelwahed, G. Karsai, and G. Biswas, "Discrete abstraction and supervisory control for switching systems," in *IEEE International Conference on Systems, Man, and Cybernetics*, vol. 1, pp. 415–421, IEEE, October 2003.

[3] S. Abdelwahed, R. Su, and S. Neema, "A feasible lookahead control for systems with finite control set," in *Proceedings of the 2005 IEEE Conference on Control Applications*, pp. 663–668, IEEE, August 2005.

[4] S. Chung, S. Lafortune, and F. Lin, "Limited lookahead policies in supervisory control of discrete event systems," *IEEE Transactions on Automatic Control*, vol. 37, pp. 1921–1935, December 1992.

[5] X. Koutsoukos, P. Antsaklis, J. Stiver, and M. Lemmon, "Supervisory control of hybrid systems," in *Proceedings of the IEEE*, pp. 1026–1048, IEEE, July 2000.

[6] P. Ramadge and W. Wonham, "The conrol of discrete event systems," *Proceedings of the IEEE*, vol. 77, pp. 81–98, January 1989.

[7] S. Chung, S. Lafortune, and F. Lin, "Supervisory control using variable lookahead policies," *Discrete Event Dynamic System: Theory and Applications*, vol. 4, pp. 237–268, July 1994.

[8] N. Hadj-Alouane, S. Lafortune, and F. Lin, "Variable lookahead supervisory control with state information," *IEEE Transactions on Automatic Control*, vol. 39, pp. 2398–2410, December 1994.

[9] R. Kumar, H. M. Chung, and S. I. Marcus, "Extension based limited lookahead supervision of discrete event systems," *Automatica*, vol. 34, no. 11, pp. 1327–1344, 1998.

[10] J. Millan and S. O'Young, "Hybrid modeling of tandem dynamically positioned vessels," in *Proceedings of the 39th IEEE Conference on Decision and Control*, December 2000.

[11] T. A. Henzinger, P. Ho, and H. Wong-Toi, "HyTech: A model checker for hybrid systems," *Software Tools for Technology Transfer*, vol. 1, pp. 110–122, 1997.

[12] J. Millan and S. O'Young, "Hybrid system control using an online discrete event supervisory strategy," in *IFAC Conference on Analysis and Design of Hybrid Systems*, IFAC, June 2006.