Structured Modeling of Concurrent Stochastic Hybrid Systems *

Mikhail Bernadsky, Raman Sharykin, and Rajeev Alur

University of Pennsylvania

Abstract. We propose a modeling language for structured specification of interacting components with both hybrid and stochastic dynamics. The behavior of a stochastic hybrid agent is described using a hybrid automaton whose dynamics is specified by stochastic differential equations and probabilistic jumps. Stochastic hybrid agents interact with other agents using shared variables. The operations of parallel composition, instantiation and hiding are defined to allow hierarchical descriptions of complex agents. We report on a stochastic extension of the modeling environment Charon for hybrid systems, a simulation tool, and case studies using the tool.

1 Introduction

Hybrid systems models combine discrete dynamics expressed using extended state machines with continuous dynamics expressed using algebraic and differential equations (see [15, 1, 14, 3] for sample models). In many applications, from embedded avionics controllers to biomolecular gene regulatory networks, there is some uncertainty inherent in the physical world that can be most appropriately described using stochastic concepts. In this paper, we extend the hybrid systems modeling with stochastic constructs for both continuous evolution and discrete switching.

Stochastic modeling is an extensively studied area. Models such as Piecewise Deterministic Markov Processes [6], Switched Diffusion Processes [8], and Stochastic Hybrid Systems [11] allow integrating discrete switching, continuous dynamics, and stochastic behavior in some manner (see [20] for a recent overview). The previous research in this area has focussed on mathematical properties such as computing the distributions for switching times. Our motivation is orthogonal, namely, developing a modeling language that will allow modular descriptions of complex systems. Modular descriptions with compositional semantics has been a central theme in concurrency theory dating back to process algebras such as CCS [17] and CSP [10]. Modularity has been studied, and exploited by analysis tools, for hybrid systems (for example, Charon [2], hybrid I/O automata [14]), and for timed probabilistic systems (c.f. [13, 5]). However, we are not aware of any concurrency formalisms that allow continuous evolution using stochastic differential equations.

* This research was supported by NSF award ITR/SY 0121431.

We develop our model by extending the modeling language Charon, a design environment for specification and analysis of embedded systems [3, 2]. In Charon, the building block for describing the system architecture is an *agent* that communicates with its environment via shared variables. The language supports the operations of *composition* of agents to model concurrency, *hiding* of variables to restrict sharing of information, and *instantiation* of agents to support reuse. The stochastic extension of Charon retains this hierarchical structure. A stochastic agent has private, output, and input variables, and its behavior is described using modes. A mode is annotated with stochastic differential equations that specify the continuous evolution of variables, and invariants that specify the region of the state-space where the current dynamics is valid. Mode switches are triggered when the invariant is violated, and are specified by probabilistic jumps with discrete distributions over the target modes and continuous distributions over the updated states.

In Section 2, we present our notion of a stochastic hybrid agent, define the underlying stochastic process for closed agents, and define the operations on agents. In Section 3, we describe the prototype implementation in Charon, along with the simulation tool. In Section 4, we present two modeling case studies and simulation-based analysis. The first example, inspired by the case study in [9], involves maneuvers by two aircrafts, and we estimate the minimum distance between them using simulations. The second example, inspired by the case study in [18], models power management strategy in a hard drive, and we use simulations to estimate the average power consumption.

2 Model

In this section we give a definition of an *agent* — a formal description of a component in a hybrid system, define its execution, specify conditions when execution is a stochastic process, and state the properties of this process.

Let $\mathbb{R}^n$, $n \geq 1$ be the n-dimensional space of reals. For a finite set X of variables, each of which ranges over $\mathbb{R}$, $V(X) = \mathbb{R}^{|X|}$ is the valuation space of X, and $\Delta(X)$ is the set of all open convex sets in $V(X)$. Given a set U, we denote by ∂U its boundary. Our modeling of continuous-time stochastic evolution builds on standard models of white noise: we use W_t^n to denote the n-dimensional standard Wiener process, the generalized derivative of this process is called the *Gaussian white noise*.

To be able to define *probability measures* on the state space of an agent, let us briefly review some terminology. Suppose that Ω is a set and $\mathcal{T}$ is a topology on Ω. Then for the topological space $(\Omega, \mathcal{T})$ the minimal σ-algebra which is generated by the open sets of $\mathcal{T}$ is called the *Borel σ-algebra* and denoted by $\mathcal{B}(\Omega)$. Recall that for $\mathbb{R}^n$ there exists the *usual topology* $\mathcal{T}_{\mathbb{R}^n}$ induced by all open rectangles in $\mathbb{R}^n$, and that for any finite set Q we can consider the *discrete topology* $\mathcal{T}_Q$ such that every subset of Q is, by definition, open in $\mathcal{T}_Q$. We denote by $\mathcal{B}(\mathbb{R}^n)$ and $\mathcal{B}(Q)$ Borel σ-algebras of $(\mathbb{R}^n, \mathcal{T}_{\mathbb{R}^n})$ and $(Q, \mathcal{T}_Q)$ respectively. Now we define $\mathcal{B}(Q \times \mathbb{R}^n)$ to be the *product topology* $\mathcal{T}_{Q\times\mathbb{R}^n} =$

$\{U_1 \times U_2 : U_1$ is open in $\mathcal{T}_Q, U_2$ is open in $\mathcal{T}_{\mathbb{R}^n}\}$. The space $Q \times \mathbb{R}^n$ and its σ-algebra $\mathcal{B}(Q \times \mathbb{R}^n)$ form a *measurable space* and therefore we can define probability measures on $(Q \times \mathbb{R}^n, \mathcal{B}(Q \times \mathbb{R}^n))$ in the standard way (c.f. [6]).

Definition 1. *An agent A is a tuple $(Q, X^p, X^o, X^e, Inv, F, G, Init, Jump)$ where*

- *Q is a finite set of* modes*;*
- *X^p is a finite set of* private variables*;*
- *X^o is a finite set of* observable variables *(or* outputs*); let $X^c = X^p \cup X^o$ be the set of* controlled variables*;*
- *X^e is a finite set of* external variables *(or* inputs*); let $X = X^c \cup X^e$ be the set of all variables.*
- *$Inv : Q \to \Delta(X)$ maps each mode q to an invariant on the variables of the agent.*
  Let $S = Q \times V(X)$ be the set of all states, $S^c = Q \times V(X^c)$ be the set of controlled states, and $D = \cup_{q \in Q}(q \times Inv(q))$, $D \subseteq S$ be the set of states satisfying the invariant.
- *$F = \{f_x : D \to \mathbb{R} | x \in X^c\}$ and $G = \{\ g_x : D \to \mathbb{R}^{|X^c|} | x \in X^c\}$ are sets of functions specified for every variable in X^c. Each $f_x(\cdot, \cdot) \in F$ and $g_x(\cdot, \cdot) \in G$ is bounded and Lipschitz continuous in the second argument;*
- *$Init : \mathcal{B}(S^c) \to [0, 1]$ is an initial probability measure on $(S^c, \mathcal{B}(S^c))$;*
- *$Jump : S \setminus D \to (\mathcal{B}(S^c) \to [0, 1])$ is called* jump measure *and maps every state s, $s \in S \setminus D$, to a probability measure on $(S^c, \mathcal{B}(S^c))$. We require that for a fixed $U \in \mathcal{B}(S^c)$, $Jump(\cdot)(U)$ is a measurable function.*

Private and observable variables are controlled by the agent. Each external variable $x \in X^e$ is an observable variable of some other agent A_x, and similarly the observable variables in X^o may be external variables of other agents. Such shared variables allow interaction among the agents. On the other hand, the variables in X^p are private, and their evolutions are hidden from the other agents. We say that a state $s \in S$ is a *flow state* if $s \in D$ and a *jump state* otherwise. An agent A is called *closed* iff $X^e = \emptyset$. Notice, that in this case $S = S^c$. In Section 2.2 we introduce the *composition* operation that allows the combination of all agents of a hybrid system into a single closed agent.

The *execution* of a closed agent A is the evolution of its (controlled) variables over time. Initially, the valuations for the variables in X^c are chosen according to the probability measure *Init*. In a flow state $s = (q, y) \in D$ the dynamics of the variables in X^c coincides with a (continuous) realization of an n-dimensional Itô diffusion $\bar{x}(t) = (x_1(t), \ldots, x_n(t))$, $n = |X^c|$, $x_i(t)$ corresponds to a variable $x_i \in X^c$. The diffusion is defined by the system of stochastic differential equations (SDEs):

$$dx_i(t) = f_{x_i}(q, \bar{x}(t))dt + g_{x_i}(q, \bar{x}(t))dW_t^n,\ i = 1, \ldots, n\ .$$

The conditions on the functions in F and G guarantee the existence and uniqueness of the diffusion (see, for example, [19]). Since $\bar{x}(t)$ is Markov, we can choose the initial condition to be $\bar{x}(0) = y$. During this evolution the mode q stays unchanged.

If A reaches a jump state s then the next state s_{new} is chosen according to the probability measure $Jump(s)(\cdot)$. The state $s_{new} = (q_{new}, y_{new})$ may be either a flow or a jump state. If it is a flow state then the evolution is determined, as before, by the corresponding SDEs with the initial condition $\bar{x}(0) = y_{new}$; if it is a jump state then the next state is chosen according to $Jump(s_{new})(\cdot)$. A sequence of successive jump states in an execution is called a *jump sequence*. We assume that jump sequences occur instantaneously.

To exclude aberrant executions we introduce the following definition.

Definition 2. *A closed agent A is* well-formed *if it satisfies the following:*

- *A flow state is reachable from every jump state in a finite number of jumps with probability one. Thus every jump sequence almost surely terminates.*
- *Expectation of N_t, which is the number of jump sequences in $[0, t]$, is finite for all t. This condition guarantees time-divergence.*

There exist sufficient tests to verify the conditions above. For example, the first condition is satisfied if $\exists \varepsilon > 0$ such that for all jump states $s \in S \setminus D$, $Jump(s)(D) > \varepsilon$. The second condition is satisfied if $\exists \varepsilon, \delta : \varepsilon > 0, 0 \leq \delta < 1$ such that for every jump state s_{jump}, $Jump(s_{jump})(S \setminus D_\varepsilon) = 1$, where $D_\varepsilon = \{s \in D | Pr(t_\star(s) < \varepsilon) > \delta\}$, $t_\star(s)$ is the time required for the diffusion to reach ∂D from s. The requirement ensures that if a flow state s is reached then, with strictly positive probability, the next jump state will be reached only after the time $t_\star(s) \geq \varepsilon$ elapses.

The execution of a well-formed agent A can be seen as a trajectory of a stochastic process P_A. The state space of P_A is S. The initial state $s_{init} = (q_{init}, y_{init})$ is chosen according to *Init*. Without loss of generality, we assume that s_{init} is a flow state. Starting from s_{init}, the process continuously evolves in the mode q_{init}. Suppose there is a stopping time T_1 such that $\bar{x}(T_1^-) = \lim_{t \uparrow T_1} \bar{x}(t)$ and $(q_{init}, \bar{x}(T_1^-))$ is a jump state. Now P_A starts a jump sequence. The jump sequences are Markov chains on the continuous space that are stopped when they reach a state in D. For every $s \in S \setminus D, U \in \mathcal{B}(D)$ we define transition kernel $K(s, U)$, which is the probability to reach U starting from s. In general we may have a sequence of stopping times $T_1, \ldots, T_m, \ldots$ such that during an interval $[T_i, T_{i+1})$ the process continuously evolves in a mode q_i, and these continuous evolutions are "glued" together by $K(\cdot, \cdot)$. Applying results from [16] (as shown in [4]) we obtain:

Theorem 1. *For a well-formed stochastic hybrid agent A, P_A is a right continuous strong Markov process.*

2.1 Relation to Other Models

In this section we briefly look at other stochastic hybrid models considered in [20] and show (informally) how they can be expressed in our framework. It should be noted that none of the models have a notion of inputs, outputs, and composition.

Piecewise Deterministic Markov Processes: A PDMP P (see [6]) is a tuple $(Q, d, Inv, f, Init, \lambda, Jump)$ where Q is a countable set of modes, each $q \in Q$ is associated with its invariant $Inv(q)$, which is an open set in $\mathbb{R}^{d(q)}$. Let $D = \cup_{q \in Q}(q \times Inv(q))$ then the state space of P is $S = D \cup \partial D$. The initial state is chosen according to the probability measure *Init* concentrated on D. A Lipschitz vector field $f : D \to \mathbb{R}^{d(\cdot)}$ gives deterministic dynamics $d\bar{x}(t) = f(q, \bar{x}(t))dt$ for P in D. The process makes a jump in two cases: when it reaches the boundary of the invariant and according to a generalized Poisson process with a transition rate function $\lambda : S \to \mathbb{R}^+$. The state after a jump is chosen according to a probability measure $Jump : S \to (\mathcal{B}(D) \to [0, 1])$; the probability measure depends on the state before the jump and is concentrated on D.

For P to be well-defined, Davis assumes that for any $t \geq 0$, $E(N_t) < \infty$ where N_t is the number of jumps in the time interval $[0, t]$.

Our framework allows modeling of a PDMP with a finite number of modes as a well-formed agent. Modeling of Q, d, *Inv*, f, *Init*, and *Jump* is straightforward. To model Poisson switches we can use the following trick inspired by [6]: Suppose we are given a Poisson process R with rate function $\lambda(\bar{x}(t))$, $\bar{x}(t)$ is an n-dimensional flow. It is known that the survivor function for a jump time T of R is

$$S(t) = Pr(T > t) = e^{-\int_0^t \lambda(\bar{x}(s))ds} \quad .$$

We can simulate a random variable T with the survival function $S(t)$ by generating a random variable U uniformly distributed on $[0, 1]$ and setting $T = S^{-1}(U)$. Equivalently, $T = \inf\{t : S(t) \leq U\}$ ($S(t)$ is a decreasing function). Now to simulate the time of a jump we generate U and define the process $Z(t) = -U + e^{-\int_0^t \lambda(\bar{x}(s))ds}$. When $Z(t)$ hits 0, a jump occurs. We can model $Z(t)$ using an algebraic variable and U by modifying the jump measure.

Switching Diffusion Processes: A SDP P (see [8]) is a tuple $(Q, S, f, \sigma, Init, \lambda_{ij})$. The state space of P is $S = Q \times \mathbb{R}^n$, Q is a finite set of modes. The initial state is chosen according to the probability measure *Init* on S. For each mode $q \in Q$, the evolution of P is given by the continuous solution of the SDEs: $d\bar{x}(t) = f(q, \bar{x}(t))dt + \sigma(q, \bar{x}(t))dW_t^n$. Transitions between modes happen according to a compound Poisson process and the transition rate function for switching between modes q_i and q_j is $\lambda_{ij} : S \to \mathbb{R}^+$. Notice that switches affect only the current mode — the trajectory in $\mathbb{R}^n$ remains continuous.

P can be modeled in our framework. The components Q, S, f, σ, *Init* have their immediate counterparts and for λ_{ij} we can apply the same trick as we used for PDMPs.

Stochastic Hybrid Systems: A SHS P (see [11]) is a tuple $(Q, S, Inv, f, \sigma, Init, G, Jump)$. The state space is $S = Q \times \mathbb{R}^n$, Q is a countable set of modes. The map *Inv* assigns each $q \in Q$ with an open set $Inv(q)$ in $\mathbb{R}^n$. Let $D = \cup_{q \in Q}(q \times Inv(q))$. The initial state is chosen according to the probability measure *Init* concentrated on D. The dynamics in D is given by the Lipschitz, bounded vector fields f and σ: $d\bar{x}(t) = f(q, \bar{x}(t))dt + \sigma(q, \bar{x}(t))dW_t^n$. The process jumps when it reaches a point in ∂D. For each q the map G partitions $\partial Inv(q)$ into disjoint measurable sets $G(q, q')$, which called guards. For each pair of q and q', $Jump(q, q')$ maps every point $s \in G(q, q')$ to a corresponding probability measure on S which is concentrated on $q' \times Inv(q')$.

We can model P with a finite number of modes as a closed agent by specializing *Jump* in our definition.

2.2 Operations

In this section we define three operations on the agents to facilitate structured description of systems.

Given an agent A and variables $x \in X^o \cup X^e$, $x' \notin X$, we denote by $A[x := x']$ a new agent in which all occurrences of x are replaced with the variable x'. This operation is called *renaming* and it is used to create instances of the same definition.

Being a component of a stochastic hybrid system, an agent interacts with other agents. In our framework an agent uses its observable and external variables as interface to the other agents. Suppose that a variable x is an external variable of an agent A and an observable variable of some other agent A', then the evolution of x in A is completely determined and coincides with its evolution in A'. Thus observable variables provide output for the other agents, and external variables serve as the inputs from the rest of the system.

Suppose that the agents A and A' interact using two variables $x_1 \in X^o_A \cap X^e_{A'}$ and $x_2 \in X^o_{A'} \cap X^e_A$. A typical scenario is the following. The variables of both A and A' are initialized and evolve over the flow states. Then A first reaches a jump state, the agent updates its state according to its jump measure. The value of x_1 changes in A. The new value of x_1 is not in the invariant of A'; therefore the state of A' changes according to its jump measure. As a result, variable x_2 obtains a new value in A' and A. If the new value of x_2 does not satisfy the invariant of A then a new jump occurs. In general, there may be a (finite) sequence of jumps until the agents settle for the new flow states and the evolution of their variables becomes continuous again.

Before defining composition formally, we notice that the *Jump* relation of an agent can be extended to another relation *GJump* defined on the entire state space S of the agent, i.e. $GJump : S \to (\mathcal{B}(S^c) \to [0, 1])$. Given a set of variables Y and a state $s = (q, y)$ in S, we denote by $y[Y]$ projection of y on $V(Y)$. Then for every state $s = (q, y)$ and $U \in \mathcal{B}(S^c)$:

$$GJump(s)(U) = \begin{cases} Jump(s)(U) & \text{if } s \in S \setminus D, \\ 1 & \text{if } s \in D \text{ and } (q, y[X^c]) \in U, \\ 0 & \text{if } s \in D \text{ and } (q, y[X^c]) \notin U \ . \end{cases}$$

Suppose we are given two agents

$$A_k = (Q_k, X^p_k, X^o_k, X^e_k, Inv_k, F_k, G_k, Init_k, Jump_k), k = 1, 2 \ .$$

We want to define an agent A which is the composition of A_1 and A_2 (we use notation $A = A_1 || A_2$). We require that $X^c_1 \cap X^c_2 = \emptyset$.

Let $X_{1\to 2}$ and $X_{2\to 1}$ be the set of variables that the agents use to interact, i.e. $X_{1\to 2} = X_1^o \cap X_2^e$, and $X_{2\to 1} = X_2^o \cap X_1^e$. Then A is the tuple $(Q, X^p, X^o, X^e, \mathit{Inv}, F, G, \mathit{Init}, \mathit{Jump})$ such that:

- $Q = Q_1 \times Q_2$;
- $X^p = X_1^p \cup X_2^p$;
- $X^o = X_1^o \cup X_2^o$ and $X^c = X_1^c \cup X_2^c$;
- $X^e = (X_1^e \cup X_2^e)\backslash(X_{1\to 2} \cup X_{2\to 1})$.
  As before, $X_k = X_k^c \cup X_k^e$, $S_k^c = Q_k \times V(X_k^c)$, for $k = 1, 2$; $X = X^c \cup X^e$, $S^c = Q \times V(X^c)$, $S = Q \times V(X)$, and $D = \cup_{q\in Q}(q \times \mathit{Inv}(q))$.
- Let $s = (q, y)$ be a state in S, $q = (q_1, q_2)$, $q_1 \in Q_1$ and $q_2 \in Q_2$. Then $y \in \mathit{Inv}(q)$ iff $y[X_1] \in \mathit{Inv}_1(q_1)$ and $y[X_2] \in \mathit{Inv}_2(q_2)$;
- $F = F_1 \cup F_2$ and $G = G_1 \cup G_2$;
- Let $U = U_1 \times U_2$, $U_k \in \mathcal{B}(S_k^c)$, $k = 1, 2$. Then $\mathit{Init}(U) = \mathit{Init}_1(U_1) \times \mathit{Init}_2(U_2)$. It is known, that we can uniquely extend $\mathit{Init}(\cdot)$ to the entire $\mathcal{B}(S^c)$.
- Let $s = (q, y)$ be a state in $S \setminus D$, $q = (q_1, q_2)$, $q_1 \in Q_1$ and $q_2 \in Q_2$. Then for $U = U_1 \times U_2$, $U_k \in \mathcal{B}(S_k^c)$, $k = 1, 2$:

$$\mathit{Jump}(s)(U) = \mathit{GJump}(q_1, y[X_1])(U_1) \times \mathit{GJump}(q_2, y[X_2])(U_2) \ .$$

  Again, we are able to uniquely extend the product probability measure $\mathit{Jump}(s)(\cdot)$ to the entire $\mathcal{B}(S^c)$.

Theorem 2. *The composition operation is associative, i.e. for any agents A_1, A_2, A_3, $(A_1||A_2)||A_3$ and $A_1||(A_2||A_3)$ are isomorphic.*

Finally, we consider the *hiding* operation. Given an agent A and an observable variable x, we write $A' = \mathit{Hide}\ x\ \mathit{in}\ A$. The agent A' has the same structure as A except its sets of private and observable variables are changed: $X_{A'}^p = X_A^p \cup \{x\}$ and $X_{A'}^o = X_A^o \backslash \{x\}$. This operation is useful to restrict the scoping of variables.

3 Implementation in Charon

Charon is a language for modular specification of interacting hybrid systems based on the notions of agents and modes. The language supports the operations of agent composition for concurrency, hiding of variables for information encapsulation, and instantiation of agents for reuse. For more details please visit `http://www.cis.upenn.edu/mobies/charon/`.

3.1 Charon Extension

Syntax Extension To make Charon suitable for our purposes, we extended the current version with the syntax for specifying initial probability measures, jumps, and SDEs.

The syntax for specifying an invariant is: `inv <condition>` where condition depends on the variables of the agent.

The syntax for specifying a jump is:

```
jump from <source_mode> when <guard>
( to <destination_mode> do { <update_1>; ... ;<update_k> }
 weight <weight> )+
```

where the guard depends on the variables of the agent and defines a part of the complement of the invariant assigned to the source mode. The union of all guards of a mode must be equivalent to the complement of the invariant of that mode, and the guards must be pairwise disjoint. A jump may have multiple transition branches. Each branch is specified by its destination mode, a sequence of updates, and the weight assigned to it. The weight can depend on the variables of the agent. The probability for a branch to be executed is proportional to its weight:

$$Pr(branch) = \frac{weight\ of\ the\ branch}{the\ sum\ of\ the\ weights\ of\ all\ branches}\ .$$

Updates `<update_1>; ... ;<update_k>` are assignments of the form `variable_name = f(...)` where f is a random variable whose distribution can depend on the variables of the agent. The whole sequence of updates specifies the probability measure on the set of the variable valuations of the destination mode.

Random variables with the following predefined distributions can be used to construct f:

- `randExp(parameter)` specifies exponential distribution;
- `randNorm(mean,variance)` specifies normal distribution;
- `randUniform(begin,end)` specifies uniform distribution on the interval `[begin,end]`;
- `randPareto(parameter_a,parameter_b)` specifies Pareto distribution.

Random variables with arbitrary distributions can be obtained by calling external Java functions.

The syntax for specifying a stochastic differential equation is:

```
SDE { d(<variable name>) == f(...)*dt + g(...)*dW(t) }
```

where $f(\ldots)$ and $g(\ldots)$ are functions which depend on the variables of the agent.

To specify the initial probability measure we use the following syntax:

```
jump from default when true
(to <destination_mode> do { <update_1>; ... ;<update_k> }
 weight <weight>)+
```

which is similar to jumps but with the source mode defined by the keyword `default` and the trivial guard.

A sample stochastic hybrid agent coded in Charon is shown in Fig. 1. The agent `system` is composed of two agents: `agent1` and `agent2`. The agent `agent1` has an observable variable y which becomes an observable variable of the composite agent `system`. The agent `agent2` controls the dynamics for the external variable x of `agent1`.

```
agent system() {
   private variable x;

   agent agent1 = agent1();
   agent agent2 = agent2();
}



agent agent1() {
   external variable x;
   observable variable y;
   private variable z;

   mode mode1 = agent1_mode1()
   mode mode2 = agent1_mode2()
   mode mode3 = agent1_mode3()

   jump from default when true
       to mode1
          do { y = randUniform(1,2); z=0 }
   jump from mode1 when x=<0 || x>=2
       to mode2
          do { z = randUniform(2,3) } weight 2
       to mode3
          do { z = randUniform(1,2) } weight 3
   jump from mode2 when z=<0
       to mode1
   jump from mode3 when z=<0
       to mode1
}

mode agent1_mode1() {
   SDE { d(y) == 2*dt + dW(t) }
   SDE { d(z) == 0 }
   inv x>0 && x<2
}

mode agent1_mode2() {
   SDE { d(y) == dt + 3*dW(t) }
   SDE { d(z) == -dt }
   inv z>0
}

mode agent1_mode3() {
   SDE { d(y)=0 }
   SDE { d(z) == -dt }
   inv z>0
}



agent agent2() {
   observable variable x;

   mode mode1 = agent2_mode1();
   mode mode2 = agent2_mode2();

   jump from default when true
      to mode1
          do { x = 1 + randExp(0.5) }
   jump from mode1 when x=<0
      to mode2
          do { x = randExp(1) }
   jump from mode2 when x=<0
      to mode1
          do { x = randUniform(1,3) }
}

mode agent2_mode1() {
   SDE { d(x) == -dt + dW(t) }
   inv x>0
}

mode agent2_mode2() {
   SDE { d(x) == -2*dt + 3*dW(t) }
   inv x>0
}
```

Fig. 1. Sample stochastic hybrid system in charon syntax

The initial probability measure for `agent1` is defined in such a way that the agent always starts in `mode1`, y is chosen according to the uniform distribution on $[1, 2]$, and z is set to zero. In `mode1` the dynamics of y is specified by an SDE and z remains constant.

When the value of the external variable x is outside of the interval $(0, 2)$, the agent jumps to either `mode2` with probability $2/5$ or to `mode3` with probability $3/5$, as specified by the weights of these two branches. With this jump the agent picks a value for z according to the distribution specified in the chosen branch of the transition. In `mode2`, the dynamics for y is specified by another SDE and z plays the role of a clock. It linearly decreases, and when it hits the boundary of the invariant $z > 0$, the agent jumps back to `mode1`. In `mode3`, y remains constant, and z plays the same role as in `mode2`. When z becomes zero the agent jumps to `mode1`.

As defined in Fig. 1, `agent2` has two modes: `mode1` and `mode2`. It always starts in `mode1` and jumps from `mode1` to `mode2` and back when x hits the boundary of the corresponding invariant, and sets the new value of x according to the specified distributions. The evolution of x in each mode is given by a stochastic differential equation.

3.2 Simulator

Charon simulator consists of two main components: the part responsible for jumps and the part responsible for modeling stochastic differential equations.

To model a jump we need to pick a branch and execute the updates associated with the branch. For this, the simulator generates a random variable N_B that determines which branch to choose. N_B takes its values in the set $\{1, \ldots, k\}$, where k is the number of branches in the transition. The probability to pick a particular value is proportional to the weight of the corresponding branch.

When a branch is chosen, the updates associated with the branch are executed. These updates can be constructed from random variables with exponential, normal, uniform, Pareto, or user-defined distributions.

To obtain realizations of solutions of stochastic differential equations the simulator uses Euler-Maruyama method. Consider a stochastic differential equation

$$dx(t) = f(\bar{x}(t))dt + g(\bar{x}(t))dW_t^n, \quad x(0) = x_0, \quad 0 \leq t \leq T \ .$$

The Euler-Maruyama simulation scheme of this SDE, over the interval $[0, T]$ divided into L subintervals of the equal length Δt, has the form

$$x_k = x_{k-1} + f(\bar{x}_{k-1})\Delta t + g(\bar{x}_{k-1}) \cdot (W_{\tau_k}^n - W_{\tau_{k-1}}^n), \quad k = 1, \ldots, L$$

where $\tau_0 = 0$, $\tau_k = \tau_{k-1} + \Delta t$. Each of the n components in the vector $W_{\tau_k}^n - W_{\tau_{k-1}}^n$ is simulated as a random variable $\sqrt{\Delta t}N(0, 1)$ where $N(0, 1)$ is a random variable with the standard normal distribution.

4 Examples

We present two examples in which we use Stochastic Charon to model a system and perform Monte-Carlo simulations to obtain statistical information. In the first example, we model a system of two aircrafts taking off simultaneously from two airports and estimate the probability that these aircrafts will come, during their flights, to a particular minimum distance. In the second example we simulate a model of a hard drive for three different values of a parameter which determines the power management policy and estimate the probability that a service request is lost.

In the examples we use the following notation presented in Fig. 2. When we write `stay for randDistr(params)` in a mode, we mean that when an agent enters the mode, we pick time according to the specified distribution and jump from the mode when that time has elapsed.

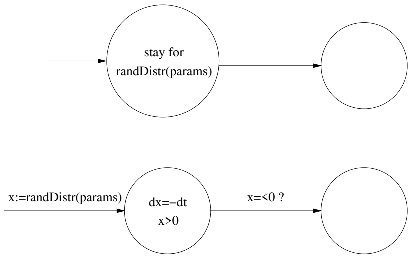

Fig. 2. Notation

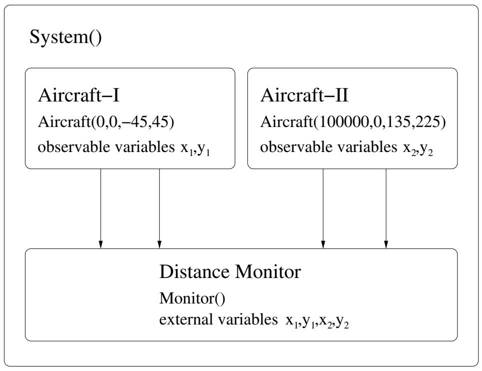

Fig. 3. System of two aircrafts

4.1 Air Traffic Control

Models for automatic air traffic management systems are important since they can be used for conflict prediction, conflict resolution, and validation of conflict detection and resolution strategies [9].

We model a pair of aircrafts flying from two airports located 100 kilometers apart. The motion of each aircraft is determined by the system of stochastic differential equations derived from the basic aerodynamics, and the stochastic part of the motion is due to the changing wind.

The aircrafts have their initial directions chosen randomly from a specified interval. During their flights each aircraft performs two turns to randomly chosen new directions. Using Monte-Carlo simulations we estimate the percentage of flights in which the minimum distance between the aircrafts reaches $r_{\min}$ for $r_{\min} = 1, \ldots, 50$ kilometers. The parameters of the model are realistic and taken from BADA database [7] for a particular aircraft. However, the flight trajectories are artificial and have the purpose to illustrate our framework.

Aircraft Model We model Airbus A300-B4 flying at 20000 feet altitude with the speed $V = 250$ knots (130 m/sec). The system of SDEs we use to model the motion of the aircraft is due to [9]. Let X, Y, and V denote the position of the aircraft and its speed. Let ψ be the flight path angle, ϕ be the bank angle, and γ be the angle of attack. Then the aircraft motion can be modeled by the following system of SDEs:

$$
\begin{aligned}
dX &= V\cos(\psi)\sin(\gamma)dt + W_x dW_t \\
dY &= V\sin(\psi)\sin(\gamma)dt + W_y dW_t \\
dh &= V\sin(\gamma)dt \\
dV &= (T/m - C_D S\rho V^2/2m - g\sin(\gamma))dt \\
d\psi &= (C_L S\rho V\sin(\phi)/(2m))dt \\
dm &= -\eta T dt
\end{aligned}
$$

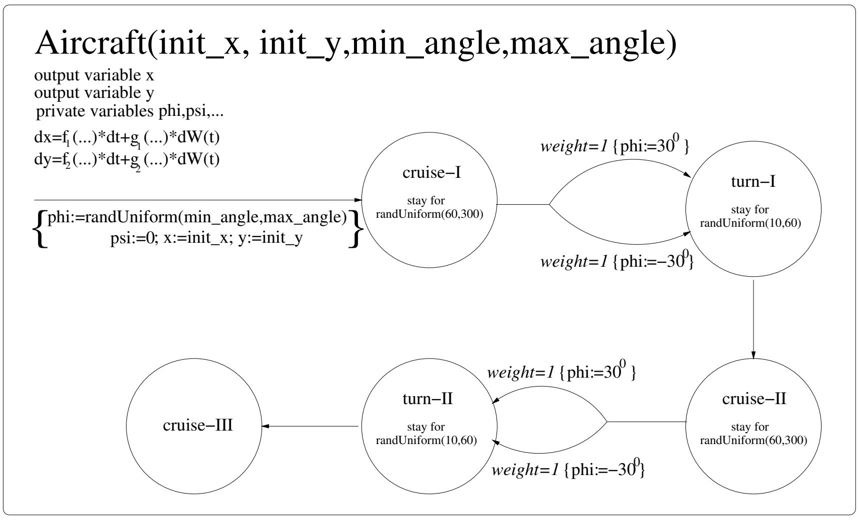

Fig. 4. Aircraft

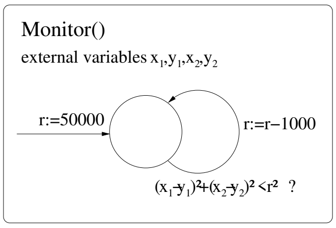

Fig. 5. Minimum distance monitor

where the concrete parameters to use in the model can be found in BADA database.

Notice that the bank angle ϕ affects the path angle ψ, and therefore it can be used to change the direction of the flight.

System The system is presented in Fig. 3. It consists of two instances of the same agent that models the dynamics of an aircraft. The aircrafts take off from the airports A and B and move towards each other. The distance between the airports is 100 kilometers. On (x, y)-plane the airport A is located at the origin. The airport B is located at $(100000, 0)$. These values are assigned to the positions of the aircrafts during instantiation of the aircraft agents in the composite agent `system`.

The agent that specifies the aircraft is shown in Fig. 4. It starts flying from the airport A or B depending on the initial parameters `x_init` and `y_init`. We consider only the flights in which the angle between the initial direction and the

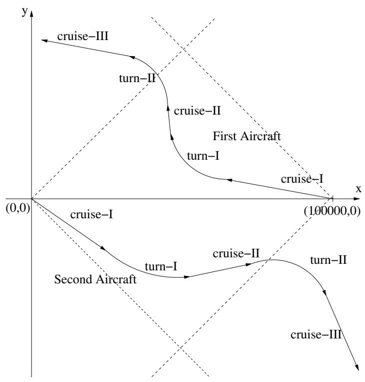

Fig. 6. Sample paths of the aircrafts

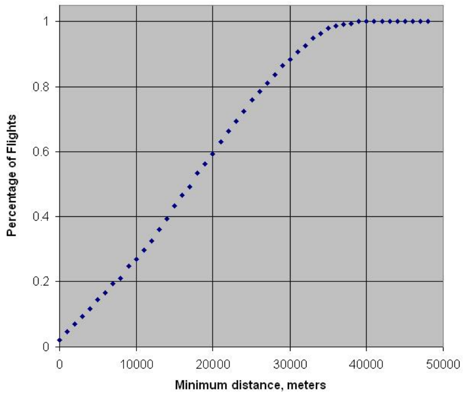

Fig. 7. Percentage of the flights with a particular minimum distance

x-axis of the first aircraft is restricted to the interval $[-45^0, 45^0]$, and the angle of the second aircraft is restricted to the interval $[135^0, 225^0]$.

After choosing the initial direction, the flight of each aircraft proceeds as follows. An aircraft flies in the cruise mode (without attempting to change the direction) for a duration of time distributed uniformly from 1 to 5 minutes. After that it performs a turn. The turn can be to the left or to the right with the equal probability. During the turn the bank angle ϕ is set either to -30^0 or to 30^0 for the left or for the right turn respectively, as specified by BADA. The aircraft performs the turn for a random time, which is uniformly distributed from 10 to 60 seconds, providing a random new direction. After that it returns to the cruise mode and flies along the new direction for an additional duration distributed uniformly from 1 to 5 minutes. After that it performs the second identical turn and continues the final course. A sample motion of two aircrafts is depicted in Fig. 6. The Charon code for the example can be found at `http://www.math.upenn.edu/~rsharyki/`.

Simulation The minimum distance is measured by the monitor presented in Fig. 5. It starts with the minimum distance equal to 50 kilometers and each time the distance becomes smaller than the current minimum distance the monitor updates it by subtracting one kilometer. At the end of the simulation, monitor's observable variable `r` contains the minimum distance that appeared during the simulation. It has precision of 1 kilometer.

We performed 1000 simulation runs of the system evolving for 20 minutes to estimate the percentage of the flights during which the aircrafts will approach to a particular minimum distance.

The simulations took 3 hours on a Celeron 1.8 GHz laptop with 512 megabytes of memory. The results are presented in Fig. 7.

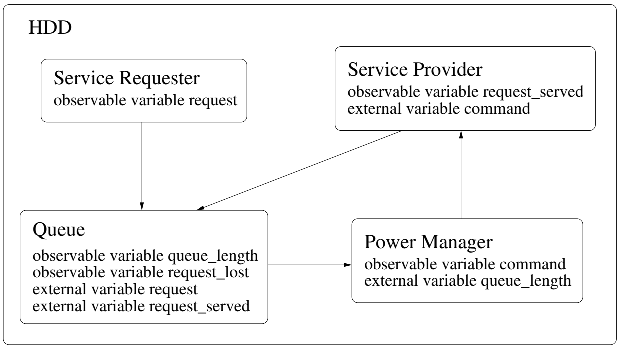

Fig. 8. Hard drive model

4.2 Hard Drive Modeling

Portable systems have very strict constraints on energy consumption. To reduce system's power consumption and increase the battery life, system-level dynamic power-management algorithms are used. These algorithms shut down idle running components and awake them when necessary.

A hard drive model presented in this section is adopted from [18]. The model in [18] uses constant jump rates between modes. To make it more precise, we specify jump rates according to the results in [21].

The structure of the system is depicted in Fig. 8. The system consists of four agents: Service Requester (SR), Queue (Q), Service Provider (SP), and Power Manager(PM). SR generates signal `request` to Q. The distribution of the time between consecutive requests is exponential. Q increases `queue_length` by one when it receives a request from SR. Q decreases `queue_length` when it receives a signal `request_served` from SP. If `queue_length` exceeds 20, the queue sets the signal `request_lost` to on. SP can be in two modes: active and stand-by. The time of the transitions from active to stand-by and from stand-by to active have uniform distributions. SP serves requests in active mode with the service time distributed exponentially. Stand-by mode is characterized by low energy consumption. PM's policy is the following one. SP starts in stand-by mode. PM watches the queue length. When the queue length reaches some particular value Q_{on}, PM issues command `go-to-active` to SP. When the queue length becomes zero PM issues command `go-to-standby` to SP. The Charon code for the example can be found at `http://www.math.upenn.edu/~rsharyki/`.

Simulation We implemented the model of the hard drive described above in Charon and used Monte-Carlo simulations to estimate the percentage of the simulation runs during which a request is lost at time t due to overflow of the queue for three different values of the awakening queue length parameter Q_{on} of Power Manager.

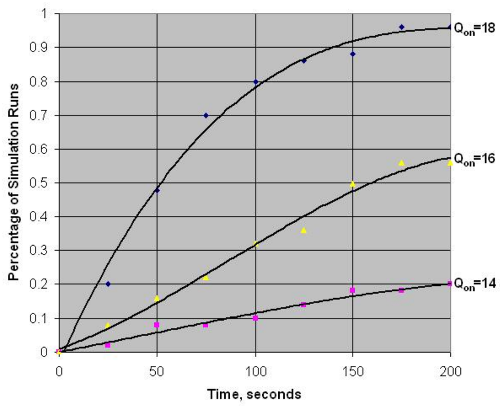

Fig. 9. Percentage of the simulation runs with a request lost up to a particular time

The simulations took 6 hours and the results are presented in Fig. 9. The x-axis represents time and y-axis represents the estimated percentage of simulation runs during which a request has been lost up to this time. The three curves depict the results obtained for the values of the awakening queue length: $Q_{\text{on}} = 14$, $Q_{\text{on}} = 16$, and $Q_{\text{on}} = 18$. We performed 50 simulation runs for each value of the parameter. During the simulations the system was modeled for 200 seconds. As the results show, a slight change in the parameter Q_{on} leads to considerable changes in the percentage of simulation runs during which a request has been lost.

5 Future Work

We have presented a modeling language that allows structured modeling of interacting stochastic hybrid components, and a prototype implementation in a simulation toolkit. Combining ideas from concurrency theory and theory of stochastic processes leads to a rich research agenda, and there are many directions for future work. First, developing a useful compositional semantics and refinement calculus in presence of all these features is a challenging problem. Second, we are investigating techniques for accurate event detection in simulating stochastic differential equations so that jumps are accurately simulated. Finally, we believe that systems biology will be a fruitful application domain for stochastic hybrid systems [12], and we are constructing models of gene regulatory processes in our toolkit.

References

1. R. Alur, C. Courcoubetis, N. Halbwachs, T.A. Henzinger, P. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems.

Theoretical Computer Science, 138:3–34, 1995.
2. R. Alur, T. Dang, J. Esposito, Y. Hur, F. Ivancic, V. Kumar, I. Lee, P. Mishra, G. Pappas, and O. Sokolsky. Hierarchical modeling and analysis of embedded systems. *Proceedings of the IEEE*, 91(1), 2003.
3. R. Alur, R. Grosu, Y. Hur, V. Kumar, and I. Lee. Modular specifications of hybrid systems in CHARON. In *Hybrid Systems: Computation and Control, Third Intl. Workshop*, LNCS 1790, pages 6–19, 2000.
4. M. Bujorianu. Extended stochastic hybrid systems and their reachability problem. In *Hybrid Systems: Computation and Control, 7th Intl. Workshop*, LNCS 2993, pages 234–249, 2003.
5. P. D'Argenio, H. Hermanns, J.-P. Katoen, and R. Klaren. Modest - a modeling and description language for stochastic timed systems. In *Proc. of the PAPM-PROBMIV Joint Intl. Workshop*, LNCS 2165, pages 87–104, 2001.
6. M.H.A. Davis. *Markov processes and optimization.* Chapman & Hall, 1993.
7. Eurocontrol Experimental Centre. *User manual for the base of aircraft data (BADA) revision 3.3*, 2003.
8. M.K. Ghosh, A. Arapostathis, and S.I. Marcus. Ergodic control of switched diffusions. *SIAM Journal on Control Optimization*, 35(6):1952–1988, 1997.
9. W. Glover and J. Lygeros. A stochastic hybrid model for air traffic control simulation. In *Hybrid Systems: Computation and Control, Seventh Intl. Workshop*, LNCS 2993, pages 372–386, 2004.
10. C.A.R. Hoare. Communicating sequential processes. *Communications of the ACM*, 21(8):666–677, 1978.
11. J. Hu, J. Lygeros, and S. Sastry. Towards a theory of stochastic hybrid systems. In *Hybrid Systems: Computation and Control, Third Intl. Workshop*, LNCS 1790, pages 160–173, 2000.
12. P. Lincoln and A. Tiwari. Symbolic systems biology: Hybrid modeling and analysis of biological networks. In *Hybrid Systems: Computation and Control, 7th Intl. Workshop*, LNCS 2993, pages 660–672, 2004.
13. G. Lowe. Probabilistic and prioritized models of Timed CSP. *Theoretical Computer Science*, 138(2):315–332, 1995.
14. N.A. Lynch, R. Segala, and F.W. Vaandrager. Hybrid I/O automata. *Information and Computation*, 185(1):105–157, 2003.
15. O. Maler, Z. Manna, and A. Pnueli. From timed to hybrid systems. In *Real-Time: Theory in Practice, REX Workshop*, LNCS 600, pages 447–484, 1991.
16. P.A. Meyer. Renaissance, recollectments, mélanges, ralentissement de processus de markov. *Annales de l'institut Fourier*, 25(3-4):465–497, 1975.
17. R. Milner. *A Calculus of Communicating Systems.* LNCS 92. Springer, 1980.
18. G. Norman, D. Parker, M. Kwiatkowska, S. Shukla, and R. Gupta. Using probabilistic model checking for dynamic power management. In *Proc. of Third Workshop on Automated Verification of Critical Systems*, 2003.
19. B. Øksendal. *Stochastic Differential Equations.* Springer-Verlag, 2003.
20. G. Pola, M. Bujorianu, J. Lygeros, and M. Di Benedetto. Stochastic hybrid systems: An overview. In *Proc. of the IFAC Conference on Analysis and Design of Hybrid Systems*, pages 45–50, 2003.
21. T. Simunic, L. Behini, P. Glynn, and G. De Micheli. Dynamic power management for portable systems. In *Mobile computing and networking, 6th Intl. Conference*, 2000.