# Specification and Analysis of Distributed Object-Based Stochastic Hybrid Systems

José Meseguer and Raman Sharykin

University of Illinois at Urbana-Champaign, USA

**Abstract.** In practice, many stochastic hybrid systems are not autonomous: they are objects that communicate with other objects by exchanging messages through an asynchronous medium such as a network. Issues such as: how to compositionally specify distributed object-based stochastic hybrid systems (OBSHS), how to formally model them, and how to verify their properties seem therefore quite important. This paper addresses these issues by: (i) defining a mathematical model for such systems that can be naturally regarded as a generalized stochastic hybrid system (GSHS) in the sense of [6]; (ii) proposing a formal OBSHS specification language in which system transitions are specified in a modular way by probabilistic rewrite rules; and (iii) showing how these systems can be subjected to statistical model checking analysis to verify their probabilistic temporal logic properties.

## 1 Introduction

Stochastic hybrid systems (see the survey [28] and references there) generalize ordinary hybrid systems (see, for example, [23, 3, 22, 4]) by allowing continuous evolution to be governed by stochastic differential equations (SDE's) and/or by allowing instantaneous changes in system modes to be probabilistic. This fits well the intrinsic uncertainty of the environments in which many hybrid systems must operate, and is also very useful when some of the system's algorithms are probabilistic. Indeed, there is a wide range of application areas, including communication networks [16], air traffic [18, 19], economics [9], fault tolerant control [13], and so on. Bioinformatics, where symbolic, hybrid, and probabilistic cell models are used, e.g., [11, 21, 15], seems also a field ripe for stochastic hybrid system applications.

While a solid foundation already exists about the mathematical properties of stochastic hybrid system models, such as being a strong Markov process, the question of how to *specify* such systems in a compositional way, so that larger systems can be described and understood in terms of smaller subsystems, remains to a good extent open, although some proposals discussed below and in Section 6 have already been made. Likewise, the question of how to *formally analyze* such systems in ways that substantially extend the analytic power of current simulation methods seems very much open. Since some application areas (for example, air traffic control) require very high assurance, specification and verification are important issues to address.

J. Hespanha and A. Tiwari (Eds.): HSCC 2006, LNCS 3927, pp. 460–475, 2006.

The main goal of this paper is to address these specification and verification issues, presenting a concrete proposal for how to formally specify and verify stochastic hybrid systems that are *distributed*, and consist of different kinds of stochastic hybrid *objects* that interact with each other by asynchronous message passing. A distributed object-oriented style with asynchronous communication seems very natural for specifying many such systems: for example, networked embedded systems, or systems made out of aircraft and other, possibly unmanned, vehicles. However, we are not aware of any formal model currently supporting this specification style for stochastic hybrid systems. Our contributions in this regard include: (i) a mathematical model of distributed and asynchronous object-based stochastic hybrid systems (OBSHS) that has the strong Markov property and can be mapped to the GSHS model of [6] (Section 3); and (ii) a formal specification language in which such systems can be specified in a modular and natural way using probabilistic rewrite rules (Section 4).

We also address formal verification issues in Section 5. Our specifications can be simulated by translating them into Maude rewriting logic specifications [7]. They can also be subjected to statistical model checking analysis using the VeStA tool [29]. In this way, probabilistic temporal logic properties of a stochastic hybrid system can be model checked with a desired degree of statistical confidence, based on Monte Carlo simulations. We explain and illustrate this kind of model checking analysis with two case studies.

In Section 6 we discuss related work and make some concluding remarks. In particular, we discuss three other models that also address composition and concurrency issues for stochastic hybrid systems, namely, those proposed in [2, 31, 12]. As we further explain in Section 6 , although the model in [2] supports objects, and that in [12] supports delayed interaction, none of these models supports distributed object communication by asynchronous message passing. To make the paper reasonably self-contained and ease the presentation in Sections 3–4, we provide basic background on term rewriting, probabilistic rewriting, and object-based specification in Section 2.

## 2 Probabilistic Rewriting and Distributed Objects

We review basic concepts on term rewriting, probabilistic rewriting, and distributed objects. This will help motivate our mathematical model of OBSHS in Section 3 and our proposed OBSHS specification language in Section 4. The exposition below is informal; we refer to [10] for more details on term rewriting, and to [1] for more details on probabilistic rewriting.

We assume a *signature* $\Sigma$ of function symbols, say, $f, g, h, a, b, \ldots \in \Sigma$, having an arity function $ar : \Sigma \longrightarrow \mathbb{N}$ specifying the number of arguments of each function symbol. We then denote by $T_\Sigma(X)$ the algebra of $\Sigma$-terms on a set $X$ of variables. For example, $f(x, g(b, y)) \in T_\Sigma(X)$ is a $\Sigma$-term with $ar(f) = ar(g) = 2$, $ar(b) = 0$, and $x \in X$. We illustrate with an example the notions of *subterm*, *subterm position*, and *subterm replacement* For example, $x$, $g(b, y)$, and $b$ are subterms of $f(x, g(b, y)$. If we think of a term as a labeled tree, then its subterms are its subtrees. We can indicate subterm positions by finite strings of natural numbers denoting paths from the root of the tree. For example, the above three subterms are at positions 1, 2, and 2.1, respectively. Given a position $p$ in a term $t$, $t/p$ denotes the subterm at position $p$. Given terms $t$ and $u$, and a position $p$ in $t$, we denote by $t[u]_p$ the new term obtained by replacing the subterm $t/p$ by $u$ at position $p$. For example, for $t$ our example term and $u = h(a)$ we have $t[u]_2 = f(x, h(a))$. Note that each term $t$ has a set $vars(t)$ of variables appearing in it. A *substitution* $\theta$ is a function $\theta : Y \longrightarrow T_\Sigma(X)$ with $Y$ a set of variables. It extends in a unique way to a $\Sigma$-homomorphism $\theta : T_\Sigma(Y) \longrightarrow T_\Sigma(X)$. For our example term $t$, if $\theta(x) = h(z)$ and $\theta(y) = c$, then $\theta(t) = f(h(z), g(b, c))$. A *rewrite rule* is a sequent $l \longrightarrow r$ with $l, r \in T_\Sigma(X)$. We call $l$ the rule's *lefthand side*, and $r$ its *righthand side*, Let $R$ be a set of rewrite rules. We say that a term $t$ is *rewritten in one step* by $R$ to $t'$, denoted $t \longrightarrow_R t'$, if there is a position $p$ in $t$ and a substitution $\theta$ such that $t/p = \theta(l)$ and $t' = t[\theta(r)]_p$. We denote by $\longrightarrow_R^*$ the reflexive and transitive closure of $R$. Intuitively, we will think of terms as the *states* of a system. Then a set $R$ of rewrite rules can be understood as a set of parametric state *transitions*, and $\longrightarrow_R^*$ as the system's *reachability relation.* We call a rewrite rule $l \longrightarrow r$ *nondeterministic* if $vars(r) \not\subseteq vars(l)$.

We can generalize this picture by rewriting not just terms, but equivalence classes of terms *modulo* an equational theory $E$. This is accomplished by the notion of a *rewrite theory* $(\Sigma, E, R)$ [24], with $\Sigma$ a signature, $E$ a set of $\Sigma$-equations, and $R$ a set of rewrite rules. Intuitively, the idea is to view the states of our system as elements of the *algebraic data type* $T_{\Sigma/E}$ specified by the equations $E$, its so-called initial algebra. The elements of $T_{\Sigma/E}$ are $E$-equivalence classes $[t]$ of $\Sigma$-terms $t$ without variables modulo the equations $E$. Now $R$ rewrites such equivalence classes instead of rewriting just terms. This is particularly useful for modeling distributed object systems that communicate with each other by message passing. We can view an object, say of a given object class $C$, as a record-like term of the form $\langle o : C \mid a_1 : v_1, \ldots, a_n : v_n\rangle$, where $o$ is the object's name or identifier, $C$ is its class name, and the $a_i$ are its state variables (each of an appropriate type) with the $v_i$ the corresponding values. We can similarly view a message addressed to $o$ as another term of the form $\langle o \leftarrow c\rangle$ with $c$ its contents and $o$ its addressee. We can then model the distributed state of an object system as a multiset or "soup" of objects and messages. We denote multiset union with the parallel composition operator $\_\|\_$, where the two underbars indicate argument positions. For example the distributed state

$$\langle o : C \mid a_1 : v_1, \ldots, a_n : v_n\rangle \parallel \langle o \leftarrow c\rangle \parallel \langle o' : C' \mid b_1 : v'_1, \ldots, b_k : v'_k\rangle \parallel \langle o' \leftarrow c'\rangle$$

has two objects $o$ and $o'$ of classes $C$ and $C'$, each with a message addressed to it and not yet received. Since multiset union is *associative and commutative*, the order of objects and messages is immaterial. In the case of objects in an OBSHS, the only additional fact is that some of the variables $a_i$ of an object $\langle o : C \mid a_1 : v_1, \ldots, a_n : v_n\rangle$ are *continuous*, that is, they take real numbers as values, while other variables can be *discrete*. Since for OBSHS real time is of the essence, in addition to ordinary messages ready for instantaneous reception, there will also be *scheduled messages* of the form $[d, \langle o \leftarrow c\rangle]$, where $d$ is a time called the *deadline*, which is decreased by time elapse. This allows us to model the fact that *asynchronous communication in a distributed system takes time*, so that a message sent is not immediately available for reception. In an OBSHS at any given time there will be *at most one* message available for reception, called the *active message*: all other messages will be scheduled messages.

The discrete transitions of a distributed object systems typically take place in response to messages: an object, upon receiving a message, may change its state, send other messages, and may disappear and/or spawn new objects. Such discrete concurrent transitions, as we will illustrate in a moment, can be naturally specified by rewrite rules. The point, however, is that for such systems the rewriting should be *multiset rewriting*, in which the order of objects and messasages in the "soup" is immaterial. This can be neatly captured by a rewrite theory $(\Sigma, AC \cup E, R)$, where $\Sigma$ specifies all the operators building up objects and messages, and the parallel composition operator $\_\|\_$, $AC$ are equations of associativity and commutativity for $\_\|\_$, and $E$ are other equations specifying auxiliary functions.

However, in an OBSHS the rewrite rules $R$ specifying the instantaneous object transitions are typically *probabilistic.* A *probabilistic rewrite rule* [20, 1] is a rewrite rule of the form

$$l(\boldsymbol{x}) \to r(\boldsymbol{x}, \boldsymbol{y}) \ \ \textit{with} \ \ \textit{probability} \ \ \boldsymbol{y} := p(\boldsymbol{x})$$

The first thing to observe is that such a rule is *nondeterministic*, because the term $r$ has new variables $\boldsymbol{y}$ disjoint from the variables $\boldsymbol{x}$ appearing in $l$. Therefore, a substitution $\theta$ for the variables $\boldsymbol{x}$ appearing in $l$ that matches a subterm of a term $t$ at position $p$ does not uniquely determine the next state after the rewrite: there can be many different choices for the next state, depending on how we instantiate the extra variables $\boldsymbol{y}$ in $r$. In fact, we can denote the different next states by expressions of the form $t[r(\theta(\boldsymbol{x}), \rho(\boldsymbol{y}))]_p$, where $\theta$ is fixed as the given matching substitution, but $\rho$ ranges over all the possible substitutions for the new variables $\boldsymbol{y}$. The probabilistic nature of the rule is expressed by the notation: *with probability* $\boldsymbol{y} := p(\boldsymbol{x})$, where $p(\boldsymbol{x})$ is a probability measure on the set of substitutions $\rho$ (modulo the equations $E$ in the given rewrite theory). However, the probability measure $p(\boldsymbol{x})$ *may depend on the matching substitution* $\theta$. We sample $\boldsymbol{y}$, that is, the substitution $\rho$, probabilistically according to the probability measure $p(\theta(\boldsymbol{x}))$.

A simple example can illustrate many of the ideas presented so far. A possible object in an OBSHS can be a *bidder* object in an auction (Section 5.1). This is an object of the form $\langle o : \textit{Bidder} \mid \textit{motivation} : m\rangle$, with *motivation* a continuous variable measuring the bidder's degree of interest in the auction. The bidder sends bids to the auction at random times, but a bidder with greater motivation will bid more often. This can be modeled by the probabilistic rewrite rule

$$\begin{array}{l} \langle X : \textit{Bidder} \mid \textit{motivation} : M\rangle \parallel \langle X \leftarrow \textit{schedule.bid}\rangle \quad \longrightarrow \\ \langle X : \textit{Bidder} \mid \textit{motivation} : M\rangle \parallel [T, \langle X \leftarrow \textit{place.bid}\rangle] \\ \qquad\qquad \textit{with} \ \ \textit{probability} \ \ T := \textit{Exp}(0.1 * \textit{duration}/0.1 + M) \end{array}$$

were upon receiving a message $\langle X \leftarrow \textit{schedule.bid}\rangle$ the bidder $X$ schedules its next bid according to an exponential distribution whose rate involves both the auction duration and its own motivation. The probability measure crucially depends on the bidder's motivation, which is determined in each rule instance by the substitution $\theta$ instantiating the lefthand side variable $M$.

## 3 Object-Based Stochastic Hybrid Systems

This section presents our OBSHS model and shows its relation to the GSHS model. To simplify the mathematical details, we adopt a more Spartan notation for object states as tuples $(o, q, v)$, with $o$ the objects name, $q$ a single discrete element that tuples together the class name and the discrete variables, and $v$ the vector of values of the continuous variables. For $(X, \mathcal{O})$ a topological space, $(X, \mathfrak{B}(X))$ denotes its associated measurable space.

**Definition 1.** *Given measurable spaces* $(X, \mathfrak{F}_X)$, $(Y, \mathfrak{F}_Y)$, *we call a function* $K : X \times \mathfrak{F}_Y \to [0, 1]$ *a* Markov kernel *(from* $(X, \mathfrak{F}_X)$ *to* $(Y, \mathfrak{F}_Y)$*) iff* $K$ *satisfies: (i)* $\forall x \in X$, $K(x, \cdot)$ *is a probability measure, and (ii)* $\forall B \in \mathfrak{F}_Y$, $K(\cdot, B)$ *is measurable. Intuitively, we think of* $K$ *as a "probabilistic transition relation" from* $X$ *to* $Y$. ■

**Definition 2.** *A* stochastic hybrid object class (OBSHS class) $C$ *is a tuple* $C = (Q_C, Oid, Inv_C, \mu, \sigma, Jump_C)$, *where:*

***Discrete States:*** $Q_C$ *is a countable set of discrete states.*
***Object Identifiers:*** *A countable set* $Oid$ *of object names.*
***Invariants:*** *For a fixed dimension* $l$, *a function* $Inv_C : Q_C \to \mathcal{O}(\mathbb{R}^l)$, *where* $\mathcal{O}(\mathbb{R}^l)$ *is the set of open sets of the Euclidean space* $\mathbb{R}^l$.
***Object States:*** *The state of an object* $o$, *with* $o \in Oid$, *is a triple* $s = (o, q, v)$ *with* $q \in Q_C$ *and* $v \in Inv_C(q)$. *The set of all such states for all objects in the class* $C$ *is denoted*

$$S_C = \bigcup_{o \in Oid,\ q \in Q_C} \{o\} \times \{q\} \times Inv_C(q)$$

*we also define its closure* $\overline{S}_C$ *as the set*

$$\overline{S}_C = \bigcup_{o \in Oid,\ q \in Q_C} \{o\} \times \{q\} \times \overline{Inv_C(q)}$$

*with* $\overline{Inv_C(q)}$ *the topological closure of the open set* $Inv_C(q)$, *and its boundary* $\partial S_C = \overline{S}_C \setminus S_C$. *Note that* $S_C$ *is a disjoint union of metric spaces and therefore has an associated measurable space* $(S_C, \mathfrak{B}(S_C))$.
***SDE Dynamics:*** *is specified by a pair of functions* $\mu : D_C \to \mathbb{R}^l$ *and* $\sigma : D_C \to \mathbb{R}^{l \times m}$, *with* $D_C = \bigcup_{q \in Q_C} \{q\} \times Inv_C(q)$, *and with* $\mu(q, x)$, $\sigma(q, x)$ *bounded and Lipschitz continuous in* $x$.
***Jump Kernel:*** *a Markov kernel* $Jump_C : \partial S_C \times \mathfrak{B}(S_C) \to [0, 1]$. ■

A message has an object $o$ from some class $C$ as its addressee, and can also contain discrete and continuous parameters.

**Definition 3.** *Given an OBSHS class $C$, a* message type *$M$ for objects of class $C$ is a tuple $M = (Oid_C, Q', d)$ with $Oid_C$ the object names of class $C$, $Q'$ a countable set of discrete parameters, and $d \in \mathbb{N}$ the dimension of the set of continuous parameters $\mathbb{R}^d$. The* set *$S_M$* of messages *of type $M$ is then $S_M = Oid_C \times Q' \times \mathbb{R}^d$. Similarly, the set $S_{SM}$ of* scheduled messages *of type $M$ is $S_{SM} = S_M \times \mathbb{R}_{\geq 0}$.* ■

**Definition 4.** *An* Object-Based Stochastic Hybrid System (OBSHS) *$A$ is given by:*

- *a set $C_1, \ldots, C_n$ of OBSHS classes.*
- *a set$M_1, \ldots, M_m$ of message types, each involving some $C_i$ among the $C_1, \ldots, C_n$.*

*The* states *of an OBSHS are multisets which contain objects in $C_1, \ldots, C_n$, scheduled messages in $M_1, \ldots, M_m$, and at most one message in one of the $M_j$:*

$$s = \{((o_1, q_1, v_1), \ldots, (o_k, q_k, v_k), [(o', q', v')], ((o'_1, q'_1, v'_1), t_1), \ldots, ((o'_s, q'_s, v'_s), t_s))\}$$

*where the object identities $o_1, \ldots, o_k$ are all different, we have a set inclusion $\{o', o'_1, \ldots, o'_s\} \subseteq \{o_1, \ldots, o_k\}$, and $[(o', q', v')]$ means that the single message $(o', q', v')$ may not be present. The* discrete component *of the above state is the multiset*

$$q = disc(s) = \{(o_1, q_1), \ldots, (o_k, q_k), <(o', q')>, (o'_1, q'_1), \ldots, (o'_s, q'_s)\}$$

*where the angle bracket operator $<(o', q')>$ acts as a marker to distinguish the discrete part of the unique active message $(o', q', v')$ if such a message is present. The set $Q_A$ of all discrete components $disc(s)$ of the states $s$ of an OBSHS $A$ is by construction a countable set. The continuous component of the state $s$ is of course scattered through the different objects and messages, but we can easily consolidate it into a single component as follows. Without loss of generality we may assume that the sets $Oid_{C_1}, \ldots, Oid_{C_n}$ and $Q'_{M_1}, \ldots, Q'_{M_m}$ are all disjoint and that the discrete message parts $q', q'_1, \ldots, q'_s$ are all different*[1]*. Then, by linearly ordering the $C_1, \ldots, C_n$, $M_1, \ldots, M_m$, and assuming linear orders in the $Oid_{C_i}$ and $Q'_{M_j}$, we can lexicographically sort the elements of any discrete state $disc(s)$ in a unique way. Suppose that the sorted form of the state $s$ above is exactly the order in which its elements are listed. Then, the* continuous component *of $s$ is the vector*

$$v = cont(s) = (v_1, \ldots, v_k, v', v'_1, t_1, \ldots, v'_s, t_s)$$

*This means that we can represent the set of all states of the OBSHS $A$ as a disjoint union*

$$S_A = \bigcup_{q \in Q_A} \{q\} \times Inv(q)$$

[1] They can always be made different, for example, by including a message identifier in each message.

*where if* $(o_1, q_1), \ldots, (o_k, q_k)$ *(of classes* $C_{i_1}, \ldots C_{i_k}$*) are the discrete objects parts of the state* $q$*, then*

$$Inv(q) = Inv_{C_{i_1}}(q_1) \times \ldots \times Inv_{C_{i_k}}(q_k) \times \mathbb{R}^{md(q)}$$

*where* $md(q)$*, the* message dimension *of* $q$*, is obtained by adding all the dimensions of the continuous components in the optional active message and in the scheduled messages.*

- *a Markov kernel* $Msg : \widehat{S}_A \times \mathfrak{B}(S_A) \rightarrow [0, 1]$ *called the* instantaneous message reception kernel*, with* $\widehat{S}_A \subset S_A$ *the measurable subset of states containing exactly one active message.*
- *an* initial probability measure $Init : \mathfrak{B}(S_A) \rightarrow [0, 1]$. ■

**Assumption 1 .** *(i)* $Msg$*, when thought of as a "probabilistic transition relation" leaves all scheduled messages untouched, and all new scheduled messages introduced by the transition have a deadline in* $\mathbb{R}_{>0}$[2]*. (ii) From a state in* $\widehat{S}_A$*, a state in* $S_A \setminus \widehat{S}_A$ *(no active messages) is reached in a finite number of* $Msg$ *"transitions" with probability 1; therefore, all* $Msg$ *transition sequences almost surely terminate.*

Since $S_A$ is a disjoint union of metric spaces, it has a measurable space structure $(S_A, \mathfrak{B}(S_A))$. The $Jump_{C_i}$ kernels specified for each class $C_1, \ldots, C_n$ in an OBSHS $A$ can be "glued together" to define a jump kernel $Jump_A : \partial S_A \times \mathfrak{B}(S_A) \rightarrow [0, 1]$, where, by definition, $\overline{S}_A = \bigcup_{q \in Q_A} \{q\} \times \overline{Inv(q)}$, and $\partial S_A = \overline{S}_A \setminus S_A$. The proof of the following proposition can be found in Appendix B of [25].

**Proposition 1.** *Given an OBSHS* $A$ *with classes* $C_1, \ldots, C_n$*, the jump kernels* $Jump_{C_1}, \ldots, Jump_{C_n}$ *can be extended to a jump kernel* $Jump_A : \partial S_A \times \mathfrak{B}(S_A) \rightarrow [0, 1]$ *in such a way that for states in* $\partial S_A$ *consisting of a single object* $(o, q, v)$ *of class* $C_i$*, then* $Jump_{C_i}((o, q, v), \cdot)$ *and* $Jump_A(\{o, q, v\}, \cdot)$ *agree, when* $\partial S_{C_i}$ *is homeomorphically embedded as a subspace of* $\partial S_A$*.* ■

An *execution* of an OBSHS is a trajectory of a stochastic process $P$. The state space of $P$ is $S_A$. The initial state is chosen according to the initial distribution $Init$. The system has evolutions of two types: *continuous evolution* and *discrete evolution.*

The system follows the *continuous evolution* (CE) when in its state all objects have their continuous states inside their boundaries, there is no active message, and there are no scheduled messages with their deadline times equal to zero. We denote the set of all such states by $S_A^{CE}$. During its continuous evolution the system evolves with each object evolution governed by the SDE of its class. The deadline time of each scheduled message decreases by the time elapsed.

When some objects reach their boundary or the deadline times of some scheduled messages reach zero, the system starts its *discrete evolution.* We denote by

[2] Note that, by definition of $S_A$, $Msg$ will introduce at most one active message.

$S_A!$ the set of states in $S_A$ such that at least one scheduled message has reached its deadline. Therefore, discrete evolution begins when the process hits $\partial S_A \cup S_A!$. During the discrete evolution the system proceeds as follows:

(i) if there are some objects whose continuous state is in the boundary of their invariants, the $Jump_A$ kernel is used to perform a transition to the new state.
(ii) if there are no objects in the boundary, and there is an active message in the state, then the $Msg$ kernel is used to perform a transition to the new state[3].
(iii) if there are no objects in the boundary and no active message, but some scheduled messages have their deadline time equal to 0, a scheduled message is chosen uniformly among the scheduled messages with 0 deadline and becomes the active message.

By the fact that $Jump_A$ moves states outside $\partial S_A$, plus Assumption 1, plus the fact that each transition of type (iii) decreases the number of zero-deadline messages, we know that after a finite number of iterations of transitions (i)-(iii) a state in $S_A^{CE}$ (which is an absorbing state for transitions (i)-(iii)) is reached with probability 1. Therefore, all such transition sequences almost surely terminate.

After reaching a state in $S_A^{CE}$ through a finite number of instantaneous transitions (i)-(iii), the system continues in time according to its continuous dynamics until a new time $T_{i+1}$ is reached in which $P$ hits $\partial S_A \cup S_A!$. Therefore, instantaneous transitions happen at discrete times $T_1 < T_2 < T_3 < \ldots$..

**Assumption 2** ***(Non-Zeno Dynamics).*** *The expectation of $N_t$, the number of instantaneous transition times on $[0, t]$ is finite for all $t$.*

We are now ready to relate the OBSHS model to a very general model proposed by Bujorianu and Lygeros, namely, *General Stochastic Hybrid Systems* (GSHS) [6]. Intuitively, the key observation is that the sequence of instantaneous transitions (i)-(iii) after a process hits a state in $\partial S_A \cup S_A!$ can be packed together into a single Markov kernel. The proof of the following proposition is given in Appendix B of [25].

**Proposition 2.** *Under Assumptions 1-2 an OBSHS A can be naturally understood as a GSHS.*[4] ■

As corollary of the above proposition, by using the same results proven for GSHS's in [6], we immediately obtain

**Theorem 1.** *Under Assumptions 1-2 an OBSHS A is a Borel right process.* ■

## 4 Specifying OBSHS's with Rewrite Rules

The different components of an OBSHS should be specified in a simple and highly reusable way by means of class specifications that are then composed into

[3] Note that, by the definition of $Msg$, the resulting state will have no objects in the boundary.

[4] In fact, by GSHS we mean a slight generalization of the model introduced in [6]. See Appendix B of [25] for details.

overall OBSHS specifications involving different class objects and messages and specifying their message-passing communication. We discuss below an object-based stochastic hybrid system example specified in SHYMaude, an extension of the PMaude language [1] supporting OBSHS features. PMaude itself supports specification of probabilistic rewrite theories in the Maude style; and simulation of such theories in the underlying Maude rewriting logic language [7].

A SHYMaude (Stochastic HYbrid Maude) *module* specifies an OBSHS and may contain several class declarations. It is introduced with the keyword `shymod` followed by its name and ends with the keyword `endshy`. In this example, the module is called `ROOM&SENSOR` and contains two classes: a class `Room` of rooms endowed with a thermostat control which can handle stochastic changes in room temperature, and a class `Sensor` of sensor objects that collect temperature information from rooms for statistical purposes. A module may import other modules, such as the `NAT` natural number module importation declared with the `protecting` keyword. Different types, called *sorts*, can also be declared; here we declare a sort `RMode` of room modes with two constants (`heat` and `cool`) introduced with the `ops` keyword. Similarly, several constants of sort `Real` are declared. Inside such a module, several stochastic hybrid object *classes* may be declared, each beginning with the `class` keyword followed by its name, and ending with `endclass`. After the class name, the discrete (`disc`) and continuous (`cont`) variables of objects in the class are declared, with the separation between both sets of variables marked by a vertical bar. For each discrete variable its corresponding sort is specified. `Room` objects have just one discrete variable (`mode`) holding the current mode, and one continuous variable (`temp`) holding the current temperature.

The invariant, SDE dynamics, and jump kernel of the class are declared with the `inv`, `dyn`, and `jump` keywords, with each declaration finished by a respective end keyword. The invariants are specified by equations identifying each invariant with a boolean predicate. The SDE dynamics is specified by a finite set of (in general parametric on the discrete state) SDE's with self-explanatory notation. Here we have just two discrete states for each object; therefore and SDE is specified for the temperature changes in each case. The Jump kernel specification is specified by probabilistic rewrite rules whose lefthand sides specify object states that have reached the boundary of their invariant, and with the corresponding righthand side specifying the state to which the object jumps according to a certain probability measure. Since for this class the jump outcomes are deterministic, the rules in this case are ordinary rewrite rules and the righthand side states are reached with probability 1.

Before specifying the `Sensor` class, its two `sleep` and `wait` modes are declared as constants of sort `SMode`. This class is very simple. It has three discrete variables: its mode, the oid of the room object it is sensing, and a counter, its only continuous variable (a temperature average) has no SDE dynamics, no invariants, and no jumps. The syntax for messages and for their contents are specified with operators of sorts `Msg` and `Contents`. Here a sensor `S` can send to its room `R` a message asking to report its temperature, and the room answers back with a temperature report. These message exchanges are specified by the first two rules. We assume that the sensor and the room are contiguous, so the reply message sent back by the room in the first rule has no delay. In the second rule, the sensor, upon receiving a temperature report, schedules a `check` message with delay chosen according to an exponential probability measure. The third rule shows how the sensor wakes up upon receiving a `check` message and queries the room again. Note that, by convention, variables not modified by a rule need not be mentioned. Note also that such message reception rules are *implicitly conditional* to the corresponding object being inside its current invariant. For example, the first rule must satisfy the implicit condition `Inv(mode : M, temp : T) = true` for `M` the current mode. Note finally, that the parallel composition operator (_||_) is denoted here with empty (juxtaposition) syntax (_ _).

```
shymod ROOM&SENSOR is protecting NAT . sort RMode .  ops heat cool
: -> RMode . ops T_max T_min intensity epsilon : -> Real .

class Room is disc mode : RMode | cont temp .

inv
  Inv(mode : heat, temp : T) = T < T_max .
  Inv(mode : cool, temp : T) = T > T_min .
endinv

dyn
  sde(mode : heat) d(temp) = intensity * dt + epsilon * dW(t) .
  sde(mode : cool) d(temp) = - intensity * dt + epsilon * dW(t) .
enddyn

jump
  rl < O : Room | mode : heat, temp : T_max > => < O : Room | mode: cool, temp : T_max > .
  rl < O : Room | mode : cool, temp : T_min > => < O : Room | mode :heat, temp : T_min > .
endjump endclass

sort SMode .  ops sleep wait : -> SMode .

class Sensor is disc mode : SMode, room : Oid, count : Nat | cont
average . endclass

op  < _ <- _ > : Oid Contents -> Msg . op report : Oid -> Contents
. op temp :_ : Real -> Contents . op check : -> Contents .

rl  < R : Room | temp : T > < R <- report(S) > => < R : Room |
temp : T > < S <- temp : T > .

rl  < S : Sensor | mode : wait, average : A, count : N > < S <-
temp : T > =>
    < S : Sensor | mode : sleep, average : (A * N + T)/(N + 1), count : N + 1 >
    [D, < S <- check >] with probability D := Exponential(1) .

rl  < S : Sensor | mode : sleep, room : R > < S <- check > =>
    < S : Sensor | mode : wait, room : R > < R <- report(S) > .
endshy
```

A SHYMaude specification can be desugared into a corresponding PMaude specification. Since probabilistic rewrite rules have extra variables in their right-hand sides, they are not directly executable. However, as explained in [1], provided sampling functions for the corresponding probability measures have been implemented in the underlying Maude, a PMaude specification can be *simulated* by an ordinary rewrite theory in Maude, in which the probabilistic choice is realized by the corresponding sampling function. In this way, SHYMaude specifications can be simulated in the underlying Maude system. The Maude translation of a SHYMaude module approximates the SDE dynamics using the Euler-Maruyama method. In this way, Monte Carlo simulations of the OBSHS specification can be performed. Furthermore, such simulations can be input to a statistical model checker like VeStA [29] to formally verify system properties.

## 5 Statistical Model Checking of OBSHS's

Developing formal verification methods for stochastic hybrid system properties that go beyond current simulation methods is an important research issue. Probabilistic temporal logics are possible candidates to state properties; but they are somewhat restrictive: they give a true or false answer when one would often be interested in a *quantitative* answer. For this reason, we use the QuaTEx language of *Quantitative Temporal Expressions* proposed in [1]. This language is supported by the VeStA tool [29], which has an interface to Maude allowing PMaude and SHYMaude specifications to be model checked with respect to QuaTEx properties using their Maude translations.

The key idea of QuaTEx is to generalize probabilistic temporal logic formulas from Boolean-valued expressions to real-valued expressions. The Boolean interpretation is preserved as a special case using the real numbers 0 and 1. As usual, QuaTEx has *state expressions*, evaluated on states, and (real-valued) *path expressions* evaluated on computation paths. The notion of state predicates is now generalized to that of *state functions*, which can evaluate quantitative properties of a state. QuaTEx is particularly expressive because of the possibility of defining recursive expressions. In this way, only the next operator $\bigcirc$ (represented as `#` in the VeStA syntax) and conditional branching *if Bexp then Pexp then Pexp′ fi*, with *Bexp* Boolean and *Pexp,Pexp′* path expressions, are needed to define more complex operators like the until $\mathcal{U}$ of probabilistic computational tree logic (PCTL) and of continuous stochastic logic (CSL) [5], and the CSL bounded until $\mathcal{U}^{\leq t}$. We refer to [1] for a detailed account of QuaTEx and its semantics. We give a flavor for it here by means of one of the QuaTEx expressions that we have evaluated in one of the case studies described below (an auction). The expression in question, `numToGet(n,id)`, computes the number of times that a bidder named `id` has to compete in a repeated auction to get the auctioned item `n` times. This is an expression evaluated on computation paths. The two auxiliary state functions in this case are `won(id)`, counting the number of times that `id` has already won a bid, and `numberOfAuctions(id)`, counting the number of auctions `id` has participated in. The corresponding QuaTEx expression is

```
numToGet(n,id) = if won(id) = n then numberOfAuctions(id) else #numToGet(n,id) fi;
```

VeStA performs *statistical model checking* on a probabilistic system by evaluating QuaTEx expressions on computation paths obtained by Monte Carlo simulations. For the above formula the VeStA command is `E[numToGet(n,id)]`. Two other parameters $\alpha$ and $\delta$ are also provided to the tool. VeStA then responds with a real number $v$, which is the estimated value of the expression with a $(1-\alpha)100\%$ confidence interval bounded by $\delta$. Depending on the tightness of the parameters, VeStA may need a greater or smaller number of sample runs to compute such a value.

### 5.1 Repeated Second-Price Auction

We have specified and analyzed a model of consecutive second-price online auctions repeated on similar items inspired by [27]. During each auction a similar item is on sale and throughout the auction the second highest bid is posted. To overbid the current bid a bidder must submit a bid higher then the current winning bid (which is not public). The winner pays the second highest bid. We enrich the model with the assumption that the bidders reside in different countries and hence the exchange rates (which fluctuate over time and whose dynamics we can model using SDE's [32]) must be used. The specification of the system consists of several object classes: a class of auctioneer who receives bids and updates the current state of the auction, the agency who provides the exchange rate modeling them using the appropriate SDE's, and two classes of bidders one being a class of "normal" bidders who bid throughout the auction with a certain probability and the other being a class of "experts" who bid at the very end of the auction with the price which they consider to be appropriate.

***VeStA Analysis.*** We have used VeStA to estimate a quantitative property of the system. In the analysis we considered a system consisting of 1 auctioneer, 3 early bidders with 1 domestic and 2 in different foreign countries, and one foreign sniper.

The quantitative property was the expected number of auctions for a bidder to get `N` items. The QuaTEx query for this quantitative property was explained above. The results of estimating this quantitative property for $\texttt{n} = 2$ with 95% confidence were: (i) for the domestic bidder the interval $[6.8, 7.5]$, (ii) for a foreign bidder $[8.2, 9.1]$, and (iii) for the sniper $[7.8, 8.6]$.

### 5.2 Thermostat

We have specified and analyzed a system consisting of $N$ rooms, each equipped with a thermostat and a central server unit controlling them. Each thermostat can be in one of three modes: heating, cooling, and idle. The temperature in each room changes randomly according to the SDE $dT = Idt + I_n dW_t$, where $I$ depends on the thermostat mode and is either the rate of heating $I_h$ , the rate of cooling $I_c$, or equal to zero in the idle mode. The server checks the temperature in each room at random times and sends commands to the thermostat according to three rules: (i) if the temperature $T$ in a room is $T > T_{max}$, then it sends a messages to to start cooling, (ii) if $T < T_{min}$, then it sends a message to start heating, (iii) if the temperature is in the region close to $T_0$ with $T_0 = (T_{max} + T_{min})/2$, then it sends a message to go to the idle mode.

***VeStA Analysis.*** We used VeStA to estimate a quantitative property of the thermostat system. The property was the probability that if during a run the temperature goes out of some desired interval $I_D = [T_0-\Delta, T_0+\Delta] \subseteq [T_{min}, T_{max}]$ in a specific room it will return to the desired interval in a specified amount of time. The QuaTEx query for this probability is

```
Eventually(t,id) = if time() >= T then 0
                    else if time() > t + I then 0
                           else if InInterval(id) = 1 then #Always(id)
                                                      else #Eventually(t,id) fi fi ;
Always(id) = if time() > T then 1
              else if InInterval(id) = 1 then #Always(id)
                                         else #Eventually(time(),id) fi fi ;
eval E[Always(id)];
```

where T bounds the lengths of paths, and I defines the interval during which the temperature has to return to the desired interval after leaving it. The function InInterval returns 1 if the temperature of the room with the identifier id is in the desired interval $I_D$ and 0 otherwise. The function Always returns 1 if the path satisfies the property stated above and 0 otherwise. Thus the expectation of this function is the desired quantitative property [1]. We used T = 20 minutes and I = 60 seconds. As in the previous property we used $T_{max} = 74$, $T_{min} = 70$, $\Delta = 1$. Using this values we obtained that with 95% confidence the desired probability lies in the confidence interval [0.31,0.34].

## 6 Related Work and Conclusions

Since under Assumption 3 the OBSHS model can be mapped into the GSHS model, the relation to less general models — including PDP [8], SHS [17], and SDP [14] — which are encompassed by GSHS as special cases is then very direct. We refer to [28] for a recent survey of stochastic hybrid system models, and focus instead on the relation of OBSHS to the three other stochastic hybrid system models addressing concurrency and composition that we are aware of, namely: (i) the model presented in [2], with associated Charon specifications; (ii) communicating piecewise deterministic Markov processes (CPDP) [31]; and (iii) stochastically and dynamically colored Petri nets (SDCPN) [12].

Like OBSHS specifications, the Charon specifications in [2] also support stochastic hybrid objects (called agents in Charon) and object composition; however, the forms of composition supported in each model are orthogonal and complementary. Charon objects can be composed out of subobjects that communicate instantaneously with each other by sharing variables. Once a closed object is thus composed, no dynamic object creation is possible; also, no support for non-instantaneous asynchronous message passing is provided. By contrast, in OBSHS objects are composed into distributed configurations by means the $\_\|\_$ operator. We view these composition operations as serving different purposes: Charon object composition is best suited for building a single object out of tightly coupled subobjects that are contiguouous to each other and can communicate instantaneously. OBSHS composition is best suited for asynchronous distributed object composition. We believe that the methods presented in this paper could be generalized to encompass both Charon and OBSHS compositions, essentially by viewing composed Charon objects as “Russian doll” objects [26] that could then communicate asynchronously with other such objects by messages in the OBSHS style.

The CPDP model [31] is a hybrid automaton formalism with two types of synchronization: on shared events, and on active-passive events with complementary labels. Composition is then synchronous parallel automaton composition. As in the PDP model [8], to which CPDP models can be reduced if their nondeterminism is eliminated by means of a scheduler, no diffusion is allowed, and no dynamic process creation is possible. Also, all communication (which can be value-passing) is assumed to be instantaneous. Therefore, CPDP models seem best suited for composing tightly coupled stochastic processes, not involving diffusion, out of simpler subprocesses.

The SDCPN model [12] has some similarities and some differences with respect to OBSHS. Both models map to GSHS. Both have distributed states formed by multiset union. In fact, by using the formalization of Petri nets as rewrite theories presented in [30], SDCPN transitions can be understood as probabilistic rewrite rules that perform multiset rewriting in the current marking multiset. In both models, both instantaneous and delayed interactions are possible (the analogous role of scheduled messages in OBSHS is played by delay transitions in SDCPN). But SDCPN models do not directly support objects and asynchronous message passing. In our view, SDCPN and OBSHS models, while having a comparable expressive power at the level of their GSHS translations, support quite different specification styles. We think that the most fruitful way of relating these models would be by unifying them within a more general model that specifies transitions as probabilistic rewrite rules.

We can summarize the work just presented as the first proposal we are aware of for a formal model and specification language for distributed object-based stochastic hybrid systems that communicate by asynchronous message passing; and for analyzing such systems by statistical model checking. We view explicit modeling of asynchronous communication as essential for many classes of applications in which objects are physically distributed over non-negligible distances. Furthermore, network communication makes message delays unavoidable. Compositionality is supported in OBSHS at two different levels: at the object level by the parallel object composition operator $\_\|\_$, and at the class level by multiple inheritance.

The work presented here is a first step. Further work is needed in several directions, including: (i) further advancing the design and implementation of our OBSHS specification language; (ii) experimenting with a wider class of applications, including Bioinformatics applications; and (iii) developing, as suggested above, a more general formalism for stochastic hybrid system specification based on probabilistic rewrite rules that can combine the benefits of the OBSHS model with those of other models such as those proposed in [2, 12].

**Acknowledgement.** Research partially supported by grants NSF CNS 05-24516 and ONR N00014-02-1-0715. We cordially thank Koushik Sen for helping us in the use of the VeStA tool, and Rajeev Alur, Mikhail Bernadsky, Manuela

Bujorianu, John Lygeros, Prakash Panangaden, and Richard Sowers for many helpful comments that allowed us to improve the paper.

## References

1. G. Agha, J. Meseguer, and K. Sen. PMaude: Rewrite-based specification language for probabilistic object systems. In *3rd Workshop on Quantitative Aspects of Programming Languages (QALP'05) ENTCS, http://osl.cs.uiuc.edu/~ksen/pub lications.html*, 2005.
2. R. Alur, M. Bernadsky, and R. Sharykin. Structured modeling of concurrent stochastic hybrid systems. In *FORMATS/FTRTFT, Springer LNCS 3253*, pages 309–324, 2004.
3. R. Alur, C. Courcoubetis, N. Halbwachs, T. A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138:3–34, 1995.
4. R. Alur, T. Dang, J. M. Esposito, Y. Hur, F. Ivancic, V. Kumar, I. Lee, P. Mishra, G. J. Pappas, and O. Sokolsky. Hierarchical modeling and analysis of embedded systems. *Proceedings of the IEEE*, 91(1):11–28, 2003.
5. A. Aziz, K. Sanwal, V. Singhal, and R. Brayton. Model-checking continuous-time markov chains. *ACM Trans. Comput. Logic*, 1(1):162–170, 2000.
6. M. L. Bujorianu and J. Lygeros. Toward a general theory of stochastic hybrid systems. *Final Project Report, HYBRIDGE*, 2005.
7. M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and J. Quesada. Maude: specification and programming in rewriting logic. *Theoretical Computer Science*, 285:187–243, 2002.
8. M. H. A. Davis. *Markov Processes and Optimization.* Chapman & Hall, 1993.
9. M. H. A. Davis and M. H. Vellekoop. Permanent health insurance: a case study in piecewise-determenistic markov modelling. *Mitteilungen der Schweiz. Vereiningung der Versicherungsmathematiker*, 2:177–212, 1995.
10. N. Dershowitz and J.-P. Jouannaud. Rewrite systems. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science, Vol. B*, pages 243–320. North-Holland, 1990.
11. S. Eker, M. Knapp, K. Laderoute, P. Lincoln, J. Meseguer, and K. Sonmez. Pathway logic: Symbolic analysis of biological signaling. In *Proceedings of the Pacific Symposium on Biocomputing*, pages 400–412, January 2002.
12. M. H. C. Everdij and H. A. P. Blom. Hybrid Petri nets with diffusion that have into-mappings with generalised stochastic hybrid processes. *Final Project Report, HYBRIDGE*, 2005.
13. M. K. Ghosh, A. Arapostathis, and S. I. Marcus. Optimal control of switching diffusions with application to flexible manufacturing systems. *SIAM Journal on Control Optimization*, 35:1183–1204, 1993.
14. M. K. Ghosh, A. Arapostathis, and S. I. Marcus. Ergodic control of switching diffusions. *SIAM J. Control Optim.*, 35(6):1952–1988, 1997.
15. P. J. Goss and J. Peccoud. Quantitative modeling of stochastic systems in molecular biology using stochastic Petri nets.. *Proc. Natl. Acad. Sci. U.S.A.*, 1998.
16. J. P. Hespanha. Stochastic hybrid systems: Application to communication network. *Hybrid Systems: Computation and Control, number 1790 in LNCS*, pages 160–173, 2000.

17. J. Hu, J. Lygeros, and S. Sastry. Towars a theory of stochastic hybrid systems. In *HSCC, Springer LNCS 1790*, pages 160–173, 2000.
18. I. Hwang, J. Hwang, and C. J. Tomlin. Flight-model-based aircraft conflict detection using a residual-mean interacting multiple model algorithm. *In AIAA Guidance, Navigation, and Control Conference, AIAA-2003-5340*, 2003.
19. HYBRIDGE. Final project report, http://hosted.nlr.nl/public/hosted-sites/hybridge/. 2005.
20. N. Kumar, K. Sen, J. Meseguer, and G. Agha. A rewriting based model of probabilistic distributed object systems. Proc. of Formal Methods for Open Object-Based Distributed Systems, FMOODS 2003, Springer LNCS Vol. 2884, 2003.
21. P. Lincoln and A. Tiwari. Symbolic systems biology: Hybrid modeling and analysis of biological networks. In *HSCC*, pages 660–672, 2004.
22. N. Lynch, R. Segala, and F. Vaandrager. Hybrid i/o automata. *Inf. Comput.*, 185(1):105–157, 2003.
23. O. Maler, Z. Manna, and A. Pnueli. From timed to hybrid systems. In *Proceedings of the Real-Time: Theory in Practice, REX Workshop*, pages 447–484, London, UK, 1992. Springer-Verlag.
24. J. Meseguer. Conditional rewriting logic as a unified model of concurrency. *Theoretical Computer Science*, 96(1):73–155, 1992.
25. J. Meseguer and R. Sharykin. Specification and analysis of distributed object-based stochastic hybrid systems. *UIUCDCS-R-2005-2649, Technical Report, Computer Science, University of Illinois at Urbana-Champaign*, 2005.
26. J. Meseguer and C. Talcott. Semantic models for distributed object reflection. In *Proceedings of ECOOP'02, Málaga, Spain, June 2002*, pages 1–36. Springer LNCS 2374, 2002.
27. H. Mizuta and K. Steiglitz. Agent-based simulation of dynamic online auctions. In *Winter Simulation Conference*, pages 1772–1777, 2000.
28. G. Pola, M. L. Bujorianu, J. Lygeros, and M. D. D. Benedetto. Stochastic hybrid models: an overview. *Proc. of the IFAC Conference on Analysis and Design of Hybrid Systems*, pages 45–50, 2003.
29. K. Sen, M. Viswanathan, and G. Agha. On statistical model checking of stochastic systems. In *CAV*, pages 266–280, 2005.
30. M.-O. Stehr, J. Meseguer, and P. Ölveczky. Rewriting logic as a unifying framework for Petri nets. In *Unifying Petri Nets*, pages 250–303. Springer LNCS 2128, 2001.
31. S. Straube and A. van der Schaft. Communicating piecewise deterministic Markov processes. *Final Project Report, HYBRIDGE*, pages http://hosted.nlr.nl/public/hosted–sites/hybridge/, 2005.
32. H. D. Vinod. Forecasting exchange rate dynamics using gmm, estimating functions and numerical conditional variance methods. Fordham University.