

*Department of Computer & Information Science*

*Technical Reports (CIS)*

---

*University of Pennsylvania*

*Year 2005*

---

Sound Code Generation from Hybrid  
System Models: Some Theoretical  
Results

Madhukar Anand\*

Jesung Kim<sup>†</sup>

Insup Lee<sup>‡</sup>

\*University of Pennsylvania

<sup>†</sup>University of Pennsylvania

<sup>‡</sup>University of Pennsylvania, lee@cis.upenn.edu

University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-05-03.

This paper is posted at ScholarlyCommons.

[http://repository.upenn.edu/cis\\_reports/6](http://repository.upenn.edu/cis_reports/6)

# Sound Code Generation from Hybrid System Models: Some Theoretical Results

Madhukar Anand, Jesung Kim, and Insup Lee  
Department of Computer and Information Science  
University of Pennsylvania  
Philadelphia, PA 19104  
{anandm, jesung, lee}@saul.cis.upenn.edu

## Abstract

Code generation from hybrid system models, a promising approach for producing reliable embedded systems, has been our research focus for some time now. In this report, we summarize the progress made thus far and provide directions for research towards realization of this goal.

## 1 Introduction

Code generation from hybrid systems model has been the focus of our research effort for some time now. A framework for automatic code generation was introduced in [5]. Extensions were proposed in [10] and [6]. Here we make a detailed report of the assumptions and results in the work so far and propose directions for future work towards the goal of model-based code generation for embedded systems.

A model-based design has several benefits, such as,

- Error detection in the early stages of development
- Analysis and formal guarantees of runtime behavior
- Automatic code generation

Incorporation of these advantages into the framework are our primary goals here. Automatic code generation, offers, in addition to the above, several benefits like faster and higher quality code development for complex systems. It also allows the designer to concentrate on high-level design issues.

The broad objective of automatic code generation from hybrid models is to generate platform-specific executable code from a platform independent model. This is briefly illustrated in Figure 1. The hybrid systems model is described in the language CHARON [3]. Another focus of our work is to validate the code

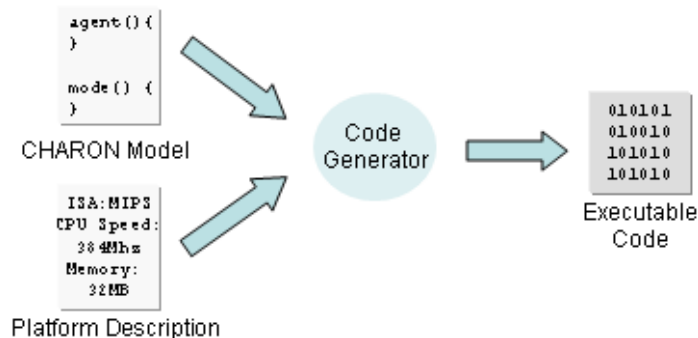


Figure 1: Model for automatic code generation.

against the originating model. This is challenging because the model is defined in the continuous-time domain, whereas the code is executed in a discrete manner. Clearly, certain properties are lost during the translation. Our goal is to identify what kinds of properties can be maintained and to develop necessary techniques.

The rest of the report is organized as follows: We present the model in Section 2. The progress so far including all the underlying assumptions is presented in Section 3. In Section 4 we discuss the possibilities of future work and conclude in Section 5.

## 2 Hybrid Systems Model

Hybrid systems is a formal model that combines continuous dynamics specified in differential equations and finite state machine based discrete control. Formally, a hybrid model consists of a real vector  $x = (x_1, x_2, \dots, x_n)$  denoting the continuous state, a finite set of discrete states  $P$  that associates  $x$  with a differential equation  $\dot{x} = f_p(x)$  for each  $p \in P$ , and a set of transitions  $E \subseteq P \times P$ . The continuous state  $x$  evolves according to the differential equation  $\dot{x} = f_p(x)$  when the current discrete state is  $p$ . When the current discrete state is changed from  $p$  to  $p'$ ,  $x$  is optionally reset to a new value  $R(x, p, p')$  defined by a map  $R : \mathbb{R}^n \times P \times P \rightarrow \mathbb{R}^n$ , and continues evolution in accordance with the differential equation  $\dot{x} = f_{p'}(x)$  associated with  $p'$ . To control the discrete behavior, discrete transitions can be guarded by predicates over  $x$ . That is, a set  $G((p, p')) \subseteq \mathbb{R}^n$  for each  $(p, p') \in E$  specifies the necessary condition on the continuous state that the transition  $(p, p')$  can be taken. Note that a discrete transition is not necessarily taken immediately even if the guard is true. To enforce a transition, an invariant set  $I(p) \subseteq \mathbb{R}^n$  is associated for each  $p \in P$  to specify the condition that the discrete state can stay in  $p$  (that is, the condition that  $x$  will follow  $\dot{x} = f_p(x)$ ). Without loss of generality, we will assume that  $G(p, p') \subset I(p)$ , and  $G(p, p'), I(p) \neq \emptyset$  for all possible  $p$  and  $p'$ .

For example, Figure 2 shows a simple hybrid model for a robot dog panning its head. It consists of two positions, or discrete states, each of which specifies

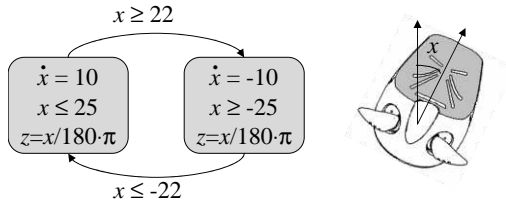


Figure 2: Hybrid model for a robot dog.

constant increase/decrease ( $\pm 10^\circ/s$ ) of variable  $x$ , which represents the angular position of the head. Transitions cause the direction of the movement of the head to be reversed by switching the position (and hence dynamics) when the head is moved beyond a certain position ( $\pm 22^\circ$ ). The transitions can be taken any time while the guard is true (i.e., the exact time when the transition is taken is non-deterministic). The invariant of each location specifies that the switch should occur before the head moves beyond its allowed range ( $x \leq 25$  and  $x \geq -25$ ).

We have been developing the modeling language CHARON, a design environment for specification and analysis of hybrid systems [1]. As a language, CHARON has many object-oriented features to aid design of complex hybrid systems. 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 building block for describing flow of control inside an atomic agent is a *mode*. A mode is basically a hierarchical state machine, that is, a mode can have sub-modes and transitions connecting them. Variables can be declared locally inside any mode with the standard scoping rules for visibility. Modes can be connected to each other only via well-defined entry and exit points. We allow *sharing* of modes so that the same mode definition can be instantiated in multiple contexts. Discrete updates in CHARON are specified by *guarded actions* labeling transitions connecting the modes. Some of the variables in CHARON can be declared *analog*, and they flow continuously during continuous updates that model passage of time. The evolution of analog variables can be constrained in three ways: differential constraints, algebraic constraints, and invariants which limit the allowed durations of flows. CHARON supports compositional trace semantics for both modes and agents [4]. For analysis it supports simulation, and formal verification of safety properties for a restricted subset, namely, models with finite discrete state and linear continuous dynamics in every mode [1, 2].

### 3 Sound code generation for embedded systems

In this section, we will review our previous work ([5, 10, 6]). Our emphasis is establishing a formal relationship between the mathematical semantics of hybrid model and the actual executions of the corresponding code. When considering accuracy of the generated code, a variety of errors can be considered. However, our focus will be on errors due to discretization of a hybrid system. An overview of all kinds of errors is given in [5].

We consider, without loss of generality, a hybrid system consisting of  $n$  concurrent atomic agents  $A_1, A_2, \dots, A_n$ . We denote a atomic agent as a tuple  $\langle M, V \rangle$  where  $M$  denotes the set of modes and  $V$  the set of all variables. That is, we assume that the mode and agent hierarchy is flattened to simplify the discussion. We consider a closed system of agents, assuming without loss of generality that naming conflicts have been resolved. We define the set of globally active modes  $M_A$  for an agent  $A = \langle A_1, \dots, A_n, V \rangle$ , where each agent  $A_i$  is atomic, as the cross-product of the active modes of its sub-agents. The set of all variables of  $A$  denoted by  $V_A$  is the union of all variables of its sub-agents, and the set of valuations of all variables is denoted by  $\Sigma_A$ . A state of the agent  $A$  then consists of a globally active mode and a valuation of all its variables. The set of all states of an agent  $A$  is denoted by  $\chi_A$ , and the set of initial states is denoted by  $\chi_A^0 \subseteq \chi_A$ . The set of globally active constraints  $Cons(M')$ , given a set of globally active mode  $M'$ , is the union of all active constraints of its sub-agents, the set of globally active invariants  $I(M')$  is the union of all active invariants of its sub-agents, and the set of globally active transitions  $T(M')$  is the union of all active transitions of its sub-agents. We call a function  $\Phi : \Sigma_A \times \mathbb{R}_{\geq 0} \rightarrow \Sigma_A$  an admissible flow for the globally active mode  $M'$  if  $\forall v \in \Sigma_A : \Phi(v, 0) = v$  and  $\Phi(v, t)$  is the solution to all globally active constraints in  $M'$ .

#### 3.1 Single threaded code generation

The framework for automatic code generation was presented in [5, 11]. The approach described there was to compile the modeling language CHARON by exploiting the semantics to generate code in a modular fashion. A sufficient condition was also identified to guarantee absence of switching errors.

The framework introduced two phases for translation from the model to the code. The front-end transforms the CHARON code into a high-level language representation. Then, the back-end compiles this code into a binary code suitable for the target platform once the target compiler is given. This is shown in Figure 3.

Here, we present a summary of assumptions and results of that paper.

##### 3.1.1 Model and Assumptions

The model was assumed to be non-blocking, the code to be running in a single thread with all the agents having the same sampling frequency,  $h$ . Further, the

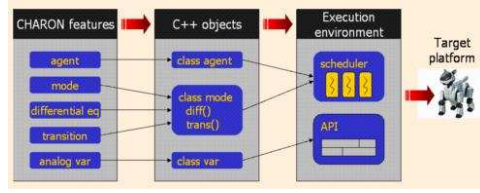


Figure 3: Model for automatic code generation.

tasks are executed in the dependency order with the task-dependency graph assumed acyclic. Only systems where guard and invariant sets are non-disjoint are considered. A transition is assumed to be taken as soon as guard is enabled, a policy called *urgent switching*. Sensing and computation were assumed to be instantaneous in this model.

### 3.1.2 Results

Discretization errors for the case of single-threaded code generation were analyzed in [5]. The possibility of missed transitions was considered and a sufficient condition was derived to guarantee no missed transitions. Before we present this result, a few definitions are in order.

**Definition 1.** (*Fixed step-size simulator*) A fixed step-size simulator with a period  $h$ , given a closed agent  $A = \langle SA, V \rangle$  of atomic sub-agents  $SA = A_1, \dots, A_n$ , computes a potentially partial function  $f_A : \mathbb{N} \rightarrow \chi_A$ . The function  $f_A$  is defined as  $f_A(0) \in \chi_A^0$ , and  $f_A(k+1) = f_n(f_A(k))$  with

1.  $f_0(M, v) = (M, \Phi(v, h))$  where  $\Phi$  is an admissible flow in  $M$ , such that  $\forall t \in [0, h] : \Phi(v, t) \in I(M)$ ; and
2. there exists a admissible ordering  $o : \{1, \dots, n\} \rightarrow SA$  that corresponds to a full ordering of the partial order given by the dependency graph of atomic agents based on their active transitions, such that one of the following two evaluations is used for  $1 \leq i \leq n$ ;
  - (a) if the invariants of the active mode  $M_{o(i)}$  of atomic sub-agent  $o(i)$  are not violated, then  $f_i(M, v) = f_{i-1}(M, v)$ ; or
  - (b) if there exists an enabled active transition  $t \in T(M_{o(i)})$ , that is  $\text{guard}_t(v) = \text{true}$ , where  $t$  is switching to the globally active mode  $M'$ , then  $f_i(M, v) = (M', \text{actions}_t(f_{i-1}(M, v)))$   $\square$

The fixed step-size simulator with a period  $h$  for a given CHARON model can be seen as computing approximations of a mathematical hybrid system model. Given a admissible initial state, it evaluates the behavior of an agent at time points  $0, h, 2h, \dots$

**Definition 2.** ( *$\epsilon$ -lookahead agent*) Given a globally active mode  $M$  for a closed agent  $A = \langle SA, V \rangle$  of atomic sub-agents  $SA = A_1, \dots, A_n$  and an admissible

flow  $\Phi$ , define the guard set  $G \subseteq I(M)$  as the set of valuations of  $\nu_A$  such that at least one globally active transition  $t$  is enabled. A globally active mode  $M$  for an agent  $A$  is called an  $\epsilon$ -lookahead mode iff

$$\text{Post}_{\Phi}(I(M) \setminus G, \epsilon) \subseteq I(M) \quad (1)$$

An  $\epsilon$ -lookahead agent  $A = \langle SA, V \rangle$  is a closed agent of atomic sub-agents such that all its globally active modes are  $\epsilon$ -lookahead modes.  $\square$

It can be shown that a  $h$ -lookahead agent can be faithfully simulated by a fixed step-size simulator with period  $h$ . This is summarized in the following theorem.

**Theorem 1.** *A non-blocking  $h$ -lookahead agent  $A$  can be faithfully simulated by a fixed step-size simulator with period  $h$ ; that is for any admissible trace  $r_A : \mathbb{R}_{\geq 0} \rightarrow \chi_A$  of the  $h$ -lookahead agent  $A$  there exists simulation trace  $f$  that can be computed by a fixed step-size simulator, such that  $\forall k \in \mathbb{N} : r(kh) = f(k)$ .  $\square$*

## 3.2 Multithreaded code generation

The model in 3.1 considered all agents to be executing in a single thread at the same frequency. This was extended to consider multiple threads of agents executing at different frequencies in [10] with an emphasis on providing guarantees of no-faulty transitions. By faulty transitions, we mean a transition that is taken in the code at some moment but is not possible in the original model. The key idea to avoid faulty transitions was to shrink the guard set once estimates for numerical and synchronization error were known. In other words, the transition would be taken conservatively. The main assumptions and results are summarized below.

### 3.2.1 Model and Assumptions

A system of agents executing with different sampling periods and having independent dynamics was considered. It was assumed that the dynamics are specified completely by differential equations without algebraic equations and a global bound on the numerical and synchronization errors is known *a priori*. Again, it was assumed that the sensing was instantaneous. A task with the least  $(t + h_T)$  value was assumed to be scheduled first, where  $t$  is the current logical time and  $h_T$  is the period. This leads to an EDF-like scheduling policy.

### 3.2.2 Results

The notion of a system of agents was formalized by defining a *System of Communicating Hybrid Automata (SCHA)*. To avoid faulty transitions, a *System of Instrumented Communicating Hybrid Automata (SICHA)* was defined and finally instrumentation was proved sufficient to ensure the absence of faulty transitions. Here we omit several definitions and instead concentrate on the concept of *instrumentation*. The reader is referred to [10] for all the details.

**Definition 3.** (CHA). A Communicating Hybrid Automaton, CHA, is a tuple  $A = (P, VC, SV, p_0, F, E, I, G, R, INIT)$ , where

- $P$  is a finite set of distinct positions,
- $VC$  is a finite set of continuous real variables, where  $|VC| = n$ ,
- $SV \subseteq VC$  is a non-empty finite set of shared variables which are partitioned into input  $SV|_{in}$  and output  $SV|_{out}$ ,
- $p_0 \in P$  is the initial position,
- $F: P \rightarrow \mathcal{F}$  assigns to  $p \in P$  a function  $F_p \in \mathcal{F}: \mathbb{R}^n \rightarrow \mathbb{R}^n$ , which defines ordinary differential equations satisfying the assumptions for existence and uniqueness of solutions for all variables in  $VC - SV|_{in}$ ,
- $E \subseteq P \times P$  is a finite set of discrete transitions,
- $I: P \rightarrow (2^{\mathbb{R}})^n$  assigns the invariant interval to  $p \in P$  such that  $I(p) \in (2^{\mathbb{R}})^n$  and, for all  $x \in VC$ , we denote the invariant interval of  $x$  at the position  $p$  by  $I_x(p)$ ,
- $G: E \rightarrow (2^{\mathbb{R}})^n$  assigns to  $(p_1, p_2) \in E$  the guard interval such that for all  $x \in VC$ ,  $G_x((p_1, p_2)) \cap I_x(p_1) \neq \emptyset$ , where  $G_x((p_1, p_2))$  denotes the guard interval of  $x$ ,
- $R: E \times VC \rightarrow \mathbb{R}$  assigns a reset value  $R((p_1, p_2), x) \in I_x(p_2)$  to a pair  $(p_1, p_2) \in E$  and  $x \in VC - SV|_{in}$ , and
- $INIT: VC \rightarrow \mathbb{R}$  assigns to a variable the initial value satisfying  $INIT(x) \in I_x(p_0)$ , for all  $x \in VC - SV|_{in}$ . □

**Definition 4.** (State of a CHA). Given a CHA  $A$ , a (time-stamped) state  $s = (p, u, t)$  is an element of  $P_A \times \mathbb{R}^n \times \mathbb{R}$  satisfying the following condition: at time  $t$ , for all  $x \in VC$ ,  $u(x) \in I_x(p)$ , where  $u(x)$  is the valuation of  $x$ . □

**Definition 5.** (System of Communicating Hybrid Automata). Given a finite set of CHAs  $\{(A_0, SV_0), \dots, (A_i, SV_i), \dots, (A_n, SV_n)\}$ , a System of Communicating Hybrid Automata denoted by SCHA  $C$  is a tuple  $((A_0, SV_0), \dots, (A_i, SV_i), \dots, (A_n, SV_n))$ , such that

- $\cup_i SV_i|_{in} \subseteq \cup_i SV_i|_{out}$  and
- $SV_i|_{out} \cap SV_j|_{out} = \emptyset$ , for all  $0 \leq i \neq j \leq n$ . □

**Definition 6.** (State of an SCHA). Given an SCHA  $C = (A_0, A_1, \dots, A_n, SV)$ , a state  $s$  is defined as  $((p^{A_0}, u^{A_0}), \dots, (p^{A_n}, u^{A_n}), t)$ , where  $(p^{A_i}, u^{A_i}, t)$  is a state of CHA  $A$  at time  $t$  satisfying that  $u^{A_i}(x) = u^{A_j}(x)$  if  $x \in SV_i|_{in} \cap SV_j|_{out}$ . □

**Definition 7.** (Run of an SCHA). A run of an SCHA  $C = (A_0, A_1, \dots, A_n, SV)$  is a (possibly infinite) sequence of states  $\langle s_0, s_1, \dots, s_i, s_{i+1}, \dots \rangle$ , where

- $s_0 = ((p_0^{A_0}, INIT_{A_0}), (p_0^{A_1}, INIT_{A_1}), \dots, (p_0^{A_n}, INIT_{A_n}), 0)$ , and
- $(s_i, s_{i+1})$  is either a discrete transition step or a continuous transition step for all  $i \geq 0$ .

A run is called an alternating run if discrete transition steps and continuous transition steps occurs alternately, i.e., if  $(s_i, s_{i+1})$  is a continuous transition step, then  $(s_{i+1}, s_{i+2})$  is a discrete transition step, and vice versa.  $\square$

**Definition 8.** (DCHA) A Discretized Communicating Hybrid Automaton, DCHA is a tuple  $H = (A, T)$ , where

- $A$  is a Communicating Hybrid Automaton,
- $T = \{t_0, t_1, t_2, \dots\}$  is a discrete time domain, where  $t_i \in \mathbb{R}$  and  $t_{i+1} > t_i$  for all  $i$ .  $\square$

**Definition 9.** (State of DCHA). Given a DCHA  $H$ , a (time-stamped) state  $s = (p, u, t)$  is an element of  $P_A \times \mathbb{R}^n \times \mathbb{R}$  satisfying the following condition:  $t \in T$ , and  $s$  is a state of  $A$ .  $\square$

Given a state  $s_i$  of DCHA  $H$ , we denote the components of  $s_i$  as  $s_{i|p}, s_{i|u}$  and  $s_{i|t}$  respectively.

**Definition 10.** ( $\mu$ -insensitivity). Given a CHA  $A$ , the invariant  $I_x(p)$  is said  $\mu$ -insensitive if, for all  $x \in VC - SV|_{in}$ ,  $x(t) \in I_x(p)$  and  $x(t + \mu) = x(t) + \int_t^{t+\mu} F_p(x) dt \in I_x(p)$  implies  $x(t + \delta) = x(t) + \int_t^{t+\delta} F_p(x) dt \in I_x(p)$  for all  $\delta \in [0, \mu]$  where  $x(t)$  denotes valuation of  $x$  at time  $t$ . When all invariants in  $A$  are  $\mu$ -insensitive,  $A$  is said  $\mu$ -insensitive. We say that SCHA  $C$  is  $\mu$ -insensitive when all CHA  $A_k$  of  $C$  is  $\mu$ -insensitive.  $\square$

**Definition 11.** (discretized SCHA). Given an SCHA  $C = (A_0, A_1, \dots, A_n, SV)$  and a discrete time domain  $T$ , a discretized SCHA, denoted by DSCHA, is a tuple  $(H_0, H_1, \dots, H_n, SV)$ , where  $H_i = (A_i, T)$ .  $\square$

To formalize the effect of accumulated numerical errors and synchronization errors of an SCHA, a formalism called a System of Instrumented Communicating Hybrid Automata (SICHA) was proposed. and it was shown that a run of the SICHA is always included in that of the SCHA.

Let the bound of discrepancy at position  $p$  be denoted by  $\gamma_{p_j, y}$  which is computed statically using Equation (2) below. Note that only *exclusive-write/multiple-read* shared variables are allowed.

Given an SCHA  $C = ((A_0, SV_0), \dots, (A_n, SV_n))$  and a shared variable  $y \in SV_j|_{in} \cap SV_k|_{out}$ , the bound of discrepancy due to synchronization error denoted by  $\gamma_{p_j, y}$  at position  $p_j$  is computed as follows:

$$\gamma_{p_j, y} = \begin{cases} |f(t, y) \cdot h(p_k)|_{max} & \text{if } h(p_k) \geq h(p_j), \\ |f(t, y) \cdot h(p_j)|_{max} & \text{if } h(p_k) < h(p_j) \end{cases} \quad (2)$$

where  $f(t, y)$  is the derivative of  $y$  at time  $t$ ,  $p_k \in P_{A_k}$ , and  $p_j \in P_{A_j}$ .

**Definition 12.** (ICHA). Given a CHA  $A$ , an Instrumented Communicating Hybrid Automaton of  $A$ , called ICHA, is defined as a tuple  $B = (A, N, h, \beta, \gamma)$  where

- $N: P_A \rightarrow \text{PROG}$  assigns to  $p \in P_A$  a numerical method program with a stepsize  $h(p)$ ,
- $h: P_A \rightarrow \mathbb{R}^+$  assigns to  $p \in P_A$  a stepsize  $h(p)$ ,
- $\beta: P_A \times VC \rightarrow \mathbb{R}$  assigns to  $x \in VC$  at each  $p \in P_A$ , a maximum accumulated numerical error to calculate  $x$  denoted by  $\beta_{p,x}$ ,
- $\gamma: P_A \times SV|_{in} \rightarrow \mathbb{R}$  assigns to  $x \in SV|_{in}$  at each  $p \in P_A$ , a maximum difference of the value of  $x$  from time  $t$  to time  $t + h(p)$  denoted by  $\gamma_{p,x}$ ,
- for each  $p \in P_A$  and the invariant interval  $I_{A,x}(p)$ ,  
 $l(I_{B,x}(p)) = l(I_{A,x}(p)) + \alpha_{p,x}$ ,  $r(I_{B,x}(p)) = r(I_{A,x}(p)) - \alpha_{p,x}$ , for all  $x \in VC_A$ , and
- for each  $e \in E_A$  and the guard interval  $G_{A,x}(e)$ ,  
 $l(G_{B,x}(e)) = l(G_{A,x}(e)) + \alpha_{p,x}$ ,  $r(G_{B,x}(e)) = r(G_{A,x}(e)) - \alpha_{p,x}$ , for all  $x \in VC_A$ ,

where

$$\alpha_{p,x} = \begin{cases} \beta_{p,x} & \text{if variable } x \notin SV|_{in} \text{ at position } p \\ \beta_{p,x} + \gamma_{p,x} & \text{otherwise.} \end{cases}$$

□

**Definition 13.** (System of Instrumented Communicating Hybrid Automata) Let  $C = (A_0, A_1, \dots, A_n, SV)$  be an SCHA. A System of ICHA (SICHA)  $D$  of an SCHA  $C$  is defined by a tuple  $(B_0, B_1, \dots, B_n, SV)$  where  $B_i$  is an ICHA of  $A_i$ . □

**Definition 14.** (Run of an SICHA). A run of an SICHA  $D = (B_0, B_1, \dots, B_n, SV)$  is a sequence of states  $\langle s_0, s_1, \dots \rangle$  such that

- $s_0 = ((p_0^{B_0}, \text{INIT}_{B_0}), (p_0^{B_1}, \text{INIT}_{B_1}), \dots, (p_0^{B_n}, \text{INIT}_{B_n}), 0)$ ,
- if  $s_i|_t = s_{i+1}|_t$ ,  $(s_i, s_{i+1})$  is a discrete transition step at time  $s_i|_t$ , and
- if  $s_i|_t < s_{i+1}|_t$ ,  $(s_i, s_{i+1})$  is a unit continuous transition step.

The alternating run of an SICHA is defined similarly to that of an SCHA. A run of an SICHA  $D$  is called an alternating run of  $D$ , if the discrete transition step and the continuous transition step occurs in  $D$  alternately. □

Now, with these definitions, we state the theorem to guarantee no faulty transition:

**Theorem 2.** Given an HA  $A$ , let  $B = (A, N, h, \beta, \gamma)$  be an IHA such that  $A$  is  $\mu_B$ -insensitive. Then, for every alternating run  $\langle s_0^B, \dots, s_i^B, \dots \rangle$ , there exists an alternating run  $\langle s_0^A, \dots, s_i^A, \dots \rangle$  of  $A$  such that

- $s_i^A|_p = s_i^B|_p$ ,
- $s_i^A|_u(x) \in [s_i^B|_u(x) - \beta_x, s_i^B|_u(x) + \beta_x]$  for all  $x \in VC_A - SV|_{in}$ , and
- $s_i^A|_t = s_i^B|_t$ . □

**Theorem 3.** Let  $C$  and  $D$  be an *SCHA*,  $(A_0, \dots, A_j, \dots, A_n, SV)$ , and its *SICHA*,  $(B_0, \dots, B_j, \dots, B_n, SV)$ , respectively. Suppose  $A_j$  is  $\mu_{B_j}$ -insensitive, then, for every alternating run  $\langle s_0^D, s_1^D \dots, s_i^D, \dots \rangle$  in  $D$ , there exists an alternating run  $\langle s_0^C, s_1^C \dots, s_i^C, \dots \rangle$  in  $C$  such that

- $s_i^C|_{A_i,p} = s_i^D|_{B_i,p}$ ,
- $s_i^C|_{A_i,u}(x) \in [s_i^D|_{A_i,u}(x) - \alpha_x, s_i^D|_{B_i,u}(x) + \alpha_x]$ , and
- $s_i^C|_t = s_i^D|_t$ . □

### 3.3 Distributed code generation

Distributed code generation was proposed as an extension to the previous work in [6]. The notion of faithful implementation was introduced with focus on guarantees of no faulty or missed transitions. A sufficient condition to guarantee against missed transitions was also given. As before, we present a summary of the model and assumptions.

#### 3.3.1 Model and Assumptions

The case of the model with constant dynamics was considered here i.e., hybrid systems where the right-hand side of every differential equation is a constant. This is needed to avoid errors in numerical integration and it also makes instrumentation analysis easier. A distributed system of agents having different sampling frequencies was considered here. Also it was assumed that the system is free from clock drift. Further, the times for sensing and communication have been assumed to be small enough to be included in the execution time. Each of the guard sets was assumed to be a rectangular set. This makes the instrumentation easy. The guard sets were assumed to be disjoint with each other. The reason was twofold. First, we could relax the need for an EDF like execution with this assumption since at most one agent has an enabled transition at any moment, thereby eliminating the need for strict task ordering. Secondly, within the same agent, the disjoint condition prevents non-deterministic choice, since at most one transition is enabled at any position.

Now, we present a formal treatment of the work done in [6]

#### 3.3.2 Results

The policy on transition here is that whenever a guard is enabled, the transition is taken eventually. This is called an *eager transition* policy. The  $\mu$ -insensitivity (Definition 10) introduced in Section 3.2 has to be redefined to enforce this policy.

**Definition 15.** ( *$\mu$ -insensitivity*). Given a CHA  $A$ , the invariant  $I_x(p)$  is said  $\mu$ -insensitive if, for all  $x \in VC - SV|_{in}$ ,  $x(t) \in I_x(p)$  and  $x(t + \mu) = x(t) + \int_t^{t+\mu} F_p(x) dt \in I_x(p)$  implies  $x(t + \delta) = x(t) + \int_t^{t+\delta} F_p(x) dt \in I_x(p)$  for all  $\delta \in [0, \mu]$ , and for all  $x \in VC - SV|_{in}$ ,  $x(t) \in G_x(p)$  and  $x(t + \mu) = x(t) + \int_t^{t+\mu} F_p(x) dt \in G_x(p)$  implies  $x(t + \delta) = x(t) + \int_t^{t+\delta} F_p(x) dt \in G_x(p)$  for all  $\delta \in [0, \mu]$ , where  $x(t)$  denotes valuation of  $x$  at time  $t$ . When all invariants and guards in  $A$  are  $\mu$ -insensitive,  $A$  is said  $\mu$ -insensitive. We say that SCHA  $C$  is  $\mu$ -insensitive when all CHA  $A_k$  of  $C$  is  $\mu$ -insensitive.  $\square$

If, from the current position, neither a continuous nor discrete transition is possible, then we conveniently say that the current position goes to a *null position*, denoted by  $\perp$ .

We are considering only those models where the guards do not overlap at any given time, both between transitions in the same agent and between transitions in different agents. This precludes models which have non-deterministic transitions. Since the implementation is deterministic, this is important, as it will allow the implementation to be faithful to the model. There are other advantages in not considering such models, such as testing, evaluating and predicting of positions.

With that important change in the model, we can define *code* more formally as :

**Definition 16.** (*Code Implementing ICHA (CICHA)*) Given an ICHA  $B$ , the code implementing Instrumented communicating hybrid automata called CICHA is defined as a tuple  $K = (B, Id, \omega, \beta, \tau, \pi, \sigma, Clk, \kappa)$  where,

- $Id : B \rightarrow \mathbb{Z}^+$  assigns a unique identifier to the implementation of ICHA  $B$ ,
- $\omega = \{\omega_1, \dots, \omega_n\}$  is a collection of subroutines implementing the ICHA  $B$ ,
- $\beta = \{\beta_1, \dots, \beta_n\}$  represents the region in memory that is associated with all the subroutines  $\omega_i \in \omega$ ,
- $\eta : B \rightarrow \mathbb{R}^+$  assigns to the ICHA  $B$  the Worst Case Execution Time to implement ICHA  $B$ ,
- $\pi \in P$  is the current evaluated position,
- $\sigma = \{(x_i, t_i, id_i, \Theta_i) | x_i \in VC_B\}$  where  $x_i$  is a continuous variable of  $B$ ,  $t_i \in \mathbb{R}^n$  is the timestamp of latest valuation of  $x_i$  by a ICHA with identifier  $id_i$  and  $\Theta_i$  is the set of synchronization mechanisms on  $x_i$ ,
- $Clk : B \rightarrow \mathbb{R}^+$  is a clock variable.
- $\kappa : B \rightarrow \mathbb{R}^+$  is logical time of the latest evaluation.  $\square$

Note that when we talk about a *timestamp*, we are talking of a vector quantity with components from different agents. This is needed as in a distributed

environment where each agent has an independent clock, there are bound to be differences in *time* as kept by each of the clocks. Also note that  $x_i$  represent the *local* copy of variable  $x_i$  and synchronization mechanisms  $L_i$  are needed to keep them updated. It is assumed that the process that writes to a variable  $x_i$ , will broadcast it immediately after the update.  $\kappa$  represents the logical time of the latest execution.

**Definition 17.** (*State of the CICHA*) Given a ICHA  $B$ , a (time-stamped) state of CICHA  $K = (B, Id, \omega, \beta, \eta, \pi, \sigma, Clk, \kappa)$   $q = (p, u, clk, \kappa)$  is an element of  $\pi_B \times \sigma \times \mathbb{R}^+ \times \mathbb{R}^+$  satisfying the following condition: at time  $clk$ , for all  $x \in VC$ ,  $val(x) \in I_x(p)$ , where  $val(x)$  is the value of  $x$ .  $\square$

In defining the state, we timestamp the states with the time from the physical clock  $clk$ . This is different from the vector timestamp associated with the variables. It is possible that the vector timestamps are generated by using the physical clock  $clk$ .

Given a state  $q_i$  of CICHA  $K$ , we denote the components of  $q_i$  as  $q_{i|p}, q_{i|u}, q_{i|t}$  and  $q_{i|l}$  respectively. Note that the code is defined in continuous time and so is the model. However, the difference here is that updates are discrete in the code whereas they are continuous in the model.

**Definition 18.** (*Trace of an CICHA*). A trace of an CICHA  $K = (B, Id, \omega, \beta, \eta, \pi, \sigma, Clk, \kappa)$  is a sequence of states  $\langle q_0, q_1, \dots \rangle$  taken at clock times  $\langle clk_0, clk_1, \dots \rangle$  such that,  $q_0 = (p_0^{B_0}, INIT_{B_0}, 0, 0)$ , where  $INIT_{B_0}$  is the initialization specified in the model. A trace  $\langle q_0, q_1, \dots, q_n \rangle$  is called terminated if  $q_{n|p} = \perp$ .  $\square$

Here, it should be noted that the trace is defined at monotonically increasing discrete timestamps usually at earliest clock times where logical times are the multiples of step-size  $h$ . We denote the earliest clock time with logical time  $\kappa$  to be  $e(\kappa)$ .

**Definition 19.** (*Faithful Implementation*) Given trace of a code  $\langle q_0, q_1, \dots \rangle$  of CICHA  $K$ , with timestamps  $\langle clk_0, clk_1, \dots \rangle$ . If,

1. At all times  $clk$ ,  $\forall x \in VC$ ,  $|val(x_B) - val(x_K)| < \alpha_{p,x}$ ,
2. If  $\langle clk_0, clk_1, \dots \rangle = \langle e(0), e(h), e(2h), e(3h), \dots \rangle$ , then there exists a corresponding run of the ICHA  $B$ ,  $\langle s_0, s_1, \dots \rangle$  taken at same logical times. At any other time  $q_{i|t}, q_{i|p} = q'_{i|p}$  such that  $q'_{i|t} = e(q_{i|t})$  or  $q'_{i|t} = e(q_{i|t} + h)$ .

then the CICHA  $K$  is said to be a faithful implementation of the ICHA  $B$ . If  $K$  satisfies condition 1, then  $K$  is said to be bounded numerical error implementation.  $\square$

It is assumed here that runtime support pushes the execution forward and also is responsible for maintaining physical times map to corresponding logical times in the model. Informally, condition (1) places a bound on the error in variables, and condition (2) gives a error bound on the timing of transitions modulo the sampling error.

**Definition 20.** (*Code Implementing System of Instrumented Communicating Hybrid Automata*) Let  $L = (B_0, B_1, \dots, B_n, SV)$  be an SICHA. The code implementing a System of ICHA (CSICHA)  $H$  of an SICHA  $D$  is defined by a tuple  $(K_0, K_1, \dots, K_n, SV, \phi, \psi, \nu)$  where,

- $K_i$  are CICHAs corresponding to ICHA  $B_i$ .
- $SV$  is the set of shared variables
- $\varphi : K \times K \rightarrow \mathbb{R}^+$  is a function that associates a maximum communication delay with every pair of CICHA  $(K_1, K_2)$ ,
- $\psi = \{\psi_1, \dots, \psi_m\}$  is a collection of subroutines implementing the SICHA  $D$ ,
- $\nu = \{\nu_1, \dots, \nu_n\}$  represents the region in memory that is associated with all the subroutines  $\psi_i \in \psi$  □

**Definition 21.** (*Faithful Implementation*) CSICHA  $L$  is said to be a faithful implementation of the SICHA  $D = (B_0, B_1, \dots, B_n, SV)$  if every CICHA  $K_{B_i}$  is a faithful implementation of ICHA  $B_i$ . □

Now, we are in a position to formally define faulty and missed transitions.

**Definition 22.** (*Faulty Transition*) If  $\langle q_0, q_1, \dots, q_{n-1}, q_n \rangle$  be a trace of the CICHA taken at clock times  $\langle clk_0, clk_1, \dots, clk_n \rangle$ , such that there exists a run of ICHA  $\langle s_0, s_1, \dots, s_{n-1} \rangle$ ,  $\forall i$ ,  $s_{i|p}$  corresponds to  $q_{i|p}$  with  $s_{i|t} = q_{i|l}$  or  $s_{i|t} = q_{i|l} + h$ , but not a run  $\langle s_0, s_1, \dots, s_n \rangle$ , then, the transition  $(q_{n-1|p}, q_{n|p})$  is a faulty transition. □

**Definition 23.** (*Missed Transition*) If  $\langle q_0, q_1, \dots, q_{n-1}, q_n \rangle$  be a terminated trace of the CICHA ( $q_{n|p} = \perp$ ) taken at clock times  $\langle clk_0, clk_1, \dots, clk_n \rangle$ , such that there exists a run of ICHA  $\langle s_0, s_1, \dots, s_{n-1} \rangle$ ,  $\forall i$ ,  $s_{i|p}$  corresponds to  $q_{i|p}$  with  $s_{i|t} = q_{i|l}$  or  $s_{i|t} = q_{i|l} + h$ , but not a run  $\langle s_0, s_1, \dots, s_n \rangle$ , then, we say that there is a missed transition at  $q_{n-1|p}$ . □

Having defined faulty and missed transitions, we now proceed to state how a faithful implementation can help to assure us of no faulty or missed transitions. The following theorems are in place :

**Theorem 4.** Let the system of CICHA  $K$  be implemented on a distributed platform, and let  $K$  correspond to a SICHA  $D = (B_0, B_1, \dots, B_n, SV)$ . If every pair  $(B_i, B_j)$ , are such that their guard sets are disjoint ( $\forall l, m, G_l^{B_i} \cap G_m^{B_j} = \emptyset$ ) and if  $K$  is a bounded numerical error implementation of  $D$ , then, there will be no faulty transitions.

*Proof.* Direct from Theorem 1 in [10]. □

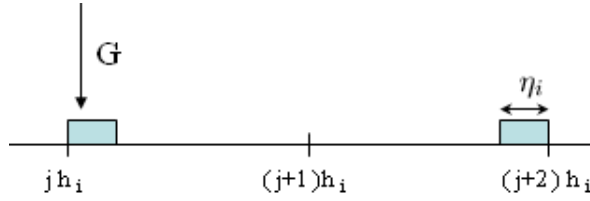


Figure 4: Worst case scheduling possibility.

**Theorem 5.** *Let the system of CICHA  $K$  be implemented on a distributed platform, Let  $K$  correspond to a SICHA  $D = (B_0, B_1, \dots, B_n, SV)$ . Let  $h_i$  be the step size associated with ICHA ( $B_i$ ).*

*Let  $\Phi$  an admissible flow and let the guard set  $G(M) \subseteq I(M)$  be such that an active transition  $t$  is enabled in state  $M$ . If,*

$$\text{Post}_{\Phi}(I(M) \setminus G(M), \Omega) \subseteq I(M) \quad (3)$$

where  $\Omega = 2(h_i + \sum h_j)$ , and the summation is over all frequencies along the longest path in the dependency graph of the guards of  $G$ , and the policy on transitions is eager, then, there will be no missed transitions.  $\square$

*Proof. (sketch)* We assume that task-period set  $\Omega = \{(\tau_i, h_i) \mid 1 \leq i \leq n\}$  is given. Each task  $\tau_i$  will be treated as a periodic task with period  $h_i$  executing in a distributed environment. Let the execution time of  $\tau_i$  be  $\eta_i$  and this is scheduled to run every  $h_i$  time units. Note that  $\eta_i$  here includes both execution time and also perhaps communication delay associated. Also, we speak of time in the reference frame at the processor executing task  $\tau_i$ . Therefore, in the worst case,  $\tau_i$  might be scheduled at time  $jh_i$  and a guard might be enabled (in the code, perhaps on a different processor) immediately after that i.e., at time  $jh_i + \epsilon$ ,  $\epsilon > 0$  and be detected only when  $\tau_i$  is next scheduled to run which may be as late as  $(j+2)h_i - \eta_i$ . Since we assume eager switching, this transition will be taken at  $(j+2)h_i - \eta_i$ . Thus, if a guard is not enabled at  $(j+2)h_i - \eta_i$ , it will go undetected and this will result in a missed transition. Hence, the guard should stay enabled for at least  $((j+2)h_i - \eta_i) - (jh_i + \epsilon) = 2h_i - \eta_i - \epsilon$  time units. Since  $\epsilon$  is arbitrary, to be safe, we can claim that it should stay enabled in the code for  $2h_i$  time units so that the transition is not missed. This is illustrated in Figure 4. However in the model, the guard would have been enabled as early as updates to independent variables. In the worst case this takes twice the sum of all frequencies along the longest path in the dependency graph of the variables ( $= \sum h_i$ ) to be enabled in the code following the same argument as above.  $\square$

We can now state a sufficient condition for a faithful implementation as:

**Theorem 6.** *Let the system of CICHA  $K$  be implemented on a distributed platform, Let  $K$  correspond to a SICHA  $D = (B_0, B_1, \dots, B_n, SV)$  of the SCHA*

*C. Let  $\Phi$  an admissible flow and let the guard set  $G(M) \subseteq I(M)$  be such that an active transition  $t$  is enabled in state  $M$ . If,*

- *Every pair  $(B_i, B_j)$ ,  $\forall l, m, G_l^{B_i} \cap G_m^{B_j} = \emptyset$ ,*
- *If  $Post_{\Phi}(I(M) \setminus G(M), \Omega) \subseteq I(M)$  where  $\Omega = 2(h_i + \sum h_j)$ , and the summation is over all frequencies along the longest path in the dependency graph of the guards of  $G$ ,*
- *$K$  is a bounded numerical error implementation of  $D$*

*then,  $K$  is a faithful implementation of  $C$  with no missed transitions.*

*Proof.* Direct from Definition 19 and Theorems 4 and 5. □

## 4 Pointers for research

### 4.1 Relaxing assumptions

We have made several assumptions that can be relaxed. We have assumed in 3.2 and 3.3 that the dynamics of agents are independent. We can extend this to include certain dependent class of systems based on the numerical method chosen to integrate the differential equations. For example, if forward difference methods can be applied to the underlying system of equations, then no restriction is necessary on the dynamics. However these methods are generally not stable. Backward difference formulas are more stable but internal variables of the integration routine will also have to be shared by the agents.

So far, when considering multithreaded and distributed code generation, we have not considered dynamics specified purely by differential equations. However, when considering real-world applications, physical constraints are often expressed in algebraic form. Thus, we will have to consider such a system. Differential-Algebraic Equations (DAE) ([9]) are then the appropriate formalism to consider. These system of equations, in general, are much harder to solve and are the focus of increasing research effort.

In the model, we have considered delays introduced due to communication. In a distributed execution environment, this could be substantial and certainly greater than the execution times and hence there is a need to incorporate it. This could possibly be done by introducing waits before evaluating guards. Another possibility, which is a more traditional solution, is to model the delays by using delay differential equations [8].

### 4.2 Implementation considerations

We have assumed all along that the code will be instrumented with a constant error bound. However, getting a bound on the numerical error is difficult in practice. Only a small class of differential equations can be solved exactly. For

most cases, numerical methods yield an approximate solution. Hence, obtaining constant bounds on error is often not possible. Errors due to numerical integration of differential equations are thus generally analyzed and represented by the  $O$  notation, and a constant error bound can be rarely analyzed if not impossible. Error estimates can perhaps be used to instrument the code in practical implementations. (for e.g, see [12]). This would mean that the instrumentation should be done dynamically.

Thus far, we have not yet considered the problem of scheduling. A detailed study is needed judging benefits of specific scheduling algorithms.

Also, Theorem 5 gives a theoretical guarantee on no missed transitions. Work has to be done in developing ways to realistically check the sufficient conditions and perhaps an algorithm and a toolset is due. At least, the condition looks to be checkable for systems with linear dynamics.

With the above considerations, we can focus on implementing the entire framework. Reachability analysis required in Theorem 5 can be tested using the tool  $d/dt$  [7].

### 4.3 Tailor-made code

With embedded systems being resource-constrained, another direction for future work would be to identify and implement code optimization. Perhaps the code could be custom generated once the platform details are input.

Finally, there is a need to develop a platform-specification language to tailor the code to specific platform as envisioned in Figure 1.

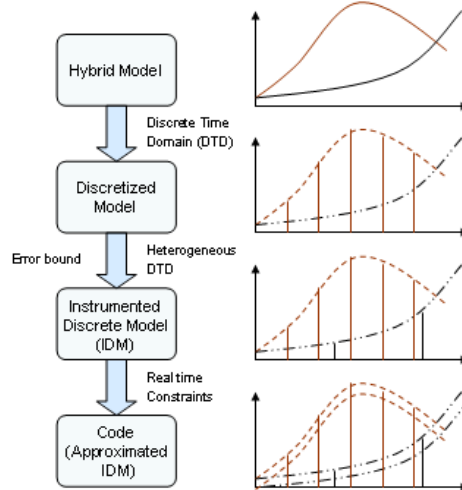


Figure 5: Design Flow.

## 5 Conclusion

In this report, we have summarized the assumptions and results of the work done so far in sound code generation from hybrid system models. Figure 5 gives the design flow from the model to the code.

In particular, we have presented the results of [6] more formally. Asynchronous update and physical time considerations were introduced in this process. We have then proposed several avenues for future work like relaxing assumptions and considerations towards an implementation of the framework.

## References

- [1] R. Alur, T. Dang, J. Esposito, Y. Hur, F. Ivančić, 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.
- [2] R. Alur, T. Dang, and F. Ivančić. Counter-example guided predicate abstraction of hybrid systems. In *Proceedings of the Ninth International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, LNCS 2619, pages 208–223. Springer-Verlag, 2003.
- [3] R. Alur, R. Grosu, Y. Hur, V. Kumar, and I. Lee. CHARON: A language for modular specification of multi-agent hybrid systems. Technical Report MS-CIS-00-01, Dept. of Computer and Information Science, University of Pennsylvania, Jan. 2000.
- [4] R. Alur, R. Grosu, I. Lee, and O. Sokolsky. Compositional refinement of hierarchical hybrid systems. In *Hybrid Systems: Computation and Control, Fourth International Workshop*, LNCS 2034, pages 33–48, 2001.
- [5] R. Alur, F. Ivančić, J. Kim, I. Lee, and O. Sokolsky. Generating embedded software from hierarchical hybrid models. In *Proceedings of ACM Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES)*, 2003.
- [6] M. Anand, J. Kim, and I. Lee. Code generation from hybrid systems models for distributed embedded systems. In *Proceedings of the 8th IEEE International Symposium on Object-oriented Real-time distributed Computing, (to appear)*, 2005.
- [7] Eugene Asarin, Thao Dang, and Oded Maler. The d/dt tool for verification of hybrid systems. In *CAV '02: Proceedings of the 14th International Conference on Computer Aided Verification*, pages 365–370. Springer-Verlag, 2002.
- [8] Uri M. Ascher and Linda R. Petzold. The numerical solution of delay-differential-algebraic equations of retarded and neutral type. *SIAM J. Numer. Anal.*, 32(5):1635–1657, 1995.

- [9] Uri M. Ascher and Linda R. Petzold. *Computer Methods for Ordinary Differential Equations and Differential-Algebraic Equations*. Society for Industrial & Applied Mathematics, 1998.
- [10] Y. Hur, J. Kim, I. Lee, and J.-Y. Choi. Sound code generation from communicating hybrid models. In *Proceedings of HSCC*, LNCS 2993, pages 432–447, 2004.
- [11] Jesung Kim and Insup Lee. Modular code generation from hybrid automata based on data dependency. In *Proceedings of RTAS*, 2003.
- [12] Jan Sieber. Local error control for general index-1 and index-2 differential algebraic equations. In *Preprint series: Institut für Mathematik, Humboldt-Universität zu Berlin (ISSN 0863-0976)*, 1997.

# List of Symbols

Symbol	Description
<b>A</b>	Atomic agent of the hybrid system, Communicating Hybrid Automaton
$\alpha$	Maximum error bound on variables
<b>B</b>	Instrumented Communicating Hybrid Automaton
$\beta$	Regions in memory containing subroutines of CICHA
$\perp$	Null position of the code
<b>C</b>	System of CHA
<b>Clk</b>	A Clock variable
$\chi$	Set of all states of an agent
<b>D</b>	System of ICHA
<b>E</b>	Set of transitions of the hybrid automata.
$\eta$	Worst case execution time of CICHA
<b>F</b>	Function associated with a differential equation
<b>G</b>	Guard set of a state
$\gamma$	Synchronization error
<b>h</b>	Step size of an agent
<b>H</b>	Discretized CHA
<b>I</b>	Invariant set of a state
<b>Id</b>	Integer identifier
<b>INIT</b>	Function assigning initial values
$\mathbb{Z}$	The domain of integers
<b>K</b>	Code implementing ICHA
$\kappa$	Logical time in the CICHA
<b>L</b>	Code implementing SICHA
<b>M</b>	Set of all modes
$\mathbb{N}$	The domain of natural numbers
$\nu$	Regions in memory containing subroutines of CSICHA
<b>o</b>	Admissible ordering of sub-agents
$\omega$	Collection of subroutines of a CICHA
<b>P</b>	Set of discrete states in the hybrid automata
<b>p</b>	Current state of the hybrid automata
$\Phi$	An admissible flow
$\pi$	Current Evaluated position
$\psi$	Collection of subroutines of CSICHA
<b>q</b>	State of the CICHA
<b>R</b>	Reset map
$\mathbb{R}$	The domain of real numbers
<b>s</b>	state of the automata
<b>t</b>	A time variable

<b>T</b>	A discrete time domain
$\tau$	A real-time task
$\Theta$	Set of synchronization mechanisms
$\sigma$	Set of local copies of variables
$\Sigma$	Set of valuations of variables
<b>u</b>	Valuation of a variable
<b>V</b>	Set of all variables of an agent
<b>VC</b>	Set of all continuous variables of an agent
$\varphi$	Bound on communication delay

## List of Acronyms

<b>Acronym</b>	<b>Description</b>
<b>CHA</b>	Communicating Hybrid Automata
<b>CICHA</b>	Code Implementing ICHA
<b>CSICHA</b>	Code Implementing SICHA
<b>DCHA</b>	Discretized CHA
<b>HA</b>	Hybrid Automata
<b>ICHA</b>	Instrumented CHA
<b>SA</b>	Set of sub-agents of an agent
<b>SCHA</b>	System of CHA
<b>SICHA</b>	System of ICHA
<b>SV</b>	Set of shared variables