11-2014

A hybrid model of connectors in cyber-physical systems

Xiaohong CHEN

Jun SUN
Singapore Management University, junsun@smu.edu.sg

Meng Sun SUN

Citation

CHEN, Xiaohong; SUN, Jun; and SUN, Meng Sun. A hybrid model of connectors in cyber-physical systems. (2014). *Proceedings of the 16th International Conference on Formal Engineering Methods, ICFEM 2014, Luxembourg, November 3–5*. 59-74. Research Collection School Of Information Systems.
Available at: https://ink.library.smu.edu.sg/sis_research/4984

A Hybrid Model of Connectors in Cyber-Physical Systems

Xiaohong Chen[1], Jun Sun[2], and Meng Sun[1]

[1] LMAM & Department of Informatics, School of Mathematical Sciences, Peking University, China
[2] Singapore University of Technology and Design, Singapore
xiaohong.chen@pku.edu.cn, sunjun@sutd.edu.sg, sunmeng@math.pku.edu.cn

Abstract. Compositional coordination models and languages play an important role in cyber-physical systems (CPSs). In this paper, we introduce a formal model for describing hybrid behaviors of connectors in CPSs. We extend the constraint automata model, which is used as the semantic model for the exogenous channel-based coordination language Reo, to capture the dynamic behavior of connectors in CPSs where the discrete and continuous dynamics co-exist and interact with each other. In addition to the formalism, we also provide a theoretical compositional approach for constructing the product automata for a Reo circuit, which is typically obtained by composing several primitive connectors in Reo.

1 Introduction

Cyber-physical systems (CPSs) are systems that integrate computing and communication with monitoring and control of physical entities. The complex interaction with the physical world through *computation*, *communication* and *control* leads to the dynamic behavior of CPSs. CPSs are present everywhere, such as airplanes and space vehicles, hybrid gas-electric vehicles, power grids, oil refineries, medical devices, defense systems, etc. The design of such systems requires understanding the complex interactions among software, hardware, networks and physical components. Coordination models and languages that provide a formalization of the "glue code" that interconnects the components and organizes their interactions in a distributed environment, are extremely important to the success of CPSs [9].

The use of coordination models and languages distinguishes the interaction among components from computing in single component explicitly. This can simplify the development process for complex systems and reasoning and verification of system properties. For example, Reo [2] is a powerful coordination language that offers an approach to express interaction protocols. Such coordination languages provide a proper approach that focuses on the interaction aspects in distributed applications, instead of the behavior models for individual components. However, most of existing coordination models and languages focused only on interactions among software components with discrete behavior. In CPSs, not only software components, but also physical components are coordinated together as well. This makes the integration of discrete and continuous dynamics for coordination an important issue in CPSs.

S. Merz and J. Pang (Eds.): ICFEM 2014, LNCS 8829, pp. 59–74, 2014.

In this paper, we investigated the problem of using Reo to model connectors in CPSs. As channels in CPS have often both discrete and continuous dynamics, existing semantics for Reo [2] is insufficient. Thus, we use *hybrid constraint automata* (HCA) which is an extension of constraint automata (CA), to capture the dynamic behavior of connectors in CPSs where the discrete and continuous dynamics co-exist and interact with each other. The concepts in HCA are borrowed from classical hybrid automata [1,6]. There are three types of transitions in HCA: (1) continuous flow inside one control state captured by some differential equations; (2) discrete jump between two control states representing the execution of some I/O operations; (3) discrete jump between two control states caused by violating the location invariant. Furthermore, a *compositional* approach for construction of HCA from a given Reo connector is provided, where the composition operator on HCA models the *join* operator in Reo to build complex connectors from basic channels.

This work is related to existing semantic models for connectors in CPSs. The time aspects of Reo has been investigated in [3], which uses timed constraint automata (TCA) as the operational semantics for Reo connectors and provides a variant of LTL as a specification formalism for timed Reo connectors. In [7,8] the TCA model has been translated into mCRL2 for model checking timed Reo connectors. The UTP model for timed connectors in Reo has been proposed in [12]. However, both the TCA model and UTP model lack of mechanisms to describe continuous dynamics for connectors. Lynch *et al.* proposed the Hybrid I/O Automata (HIOA) model [11,10] for the hybrid behavior in composition of components, where the input action enabling and input flow enabling axioms should be satisfied, which is not required in the constraint automata (and HCA) model.

The paper is structured as follows. We briefly summarize the coordination language Reo in Section 2. In Section 3 we introduce the hybrid constraint automata model. In Section 4 we show some examples of hybrid Reo circuits and how HCA can serve as their operational model. Finally, Section 5 concludes with further research directions.

2 A Reo Primer

Reo is a channel-based exogenous coordination model wherein complex coordinators, called *connectors*, are compositionally constructed from simpler ones. We summarize only the main concepts in Reo here. Further details about Reo and its semantics can be found in [2,4,5].

A connector provides the protocol that controls and organizes the communication, synchronization and cooperation among the components that they interconnect. Primitive connectors in Reo are channels that have two channel ends. There are two types of channel ends: *source* and *sink*. A source channel end accepts data into its channel, and a sink channel end dispenses data out of its channel. It is possible for the ends of a channel to be both sinks or both sources. Reo places no restriction on the behavior of a channel and thus allows an open-ended set of different channel types to be used simultaneously. Each channel end can be connected to at most one component instance at any given time.

Figure 1 shows the graphical representation of some simple channel types in Reo. A *FIFO1 channel* represents an asynchronous channel with one buffer cell which is empty

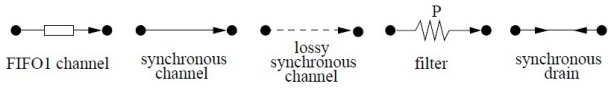

Fig. 1. Some basic channels in Reo

if no data item is shown in the box (this is the case in Figure 1). If a data element d is contained in the buffer of an FIFO1 channel then d is shown inside the box in its graphical representation. A *synchronous channel* has a source and a sink end and no buffer. It accepts a data item through its source end iff it can simultaneously dispense it through its sink. A *lossy synchronous channel* is similar to a synchronous channel except that it always accepts all data items through its source end. The data item is transferred if it is possible for the data item to be dispensed through the sink end, otherwise the data item is lost. For a *filter channel*, its pattern P specifies the type of data items that can be transmitted through the channel. Any value $d \in P$ is accepted through its source end iff its sink end can simultaneously dispense d; all data items $d \notin P$ are always accepted through the source end, but are immediately lost. The *synchronous drain* has two source ends and no sink end. It can accept a data item through one of its ends iff it can simultaneously accept data item through the other end, and all data accepted by the channel are lost.

Complex connectors are constructed by composing simpler ones mainly via the *join* and *hiding* operations. Channels are joined together in a node which consists of a set of channel ends. Nodes are categorized into *source*, *sink* and *mixed nodes*, depending on whether all channel ends that coincide on a node are source ends, sink ends or a combination of the two. The hiding operation is used to hide the internal topology of a component connector. The hidden nodes can no longer be accessed or observed from outside. A complex connector has a graphical representation, called a *Reo circuit*, which is a finite graph where the *nodes* are labeled with pair-wise disjoint, non-empty sets of channel ends, and the *edges* represent the connecting channels. The behavior of a Reo circuit is formalized by means of the data-flow at its sink and source nodes. Intuitively, the source nodes of a circuit are analogous to the input ports, and the sink nodes to the output ports of a component, while mixed nodes are its hidden internal details. Components cannot connect to, read from, or write to mixed nodes. Instead, data-flow through mixed nodes is totally specified by the circuits they belong to.

A component can write data items to a source node that it is connected to. The write operation succeeds only if all (source) channel ends coincident on the node accept the data item, in which case the data item is simultaneously written to every source end coincident on the node. A source node, thus, acts as a replicator. A component can obtain data items, by an input operation, from a sink node that it is connected to. A take operation succeeds only if at least one of the (sink) channel ends coincident on the node offers a suitable data item; if more than one coincident channel end offers suitable data items, one is selected nondeterministically. A sink node, thus, acts as a nondeterministic merger. A mixed node nondeterministically selects and takes a suitable data item offered by one of its coincident sink channel ends and replicates it into all of its coincident source channel ends. A component can not connect to, take from, or write to mixed nodes.

3 Hybrid Constraint Automata

In order to capture both discrete and continuous behaviors of connectors in CPSs, we extend the model of constraint automata as *hybrid constraint automata*. The formal definition of hybrid constraint automata (HCA) arises by combining the concepts of constraint automata and hybrid automata.

3.1 Syntax of HCA

Data Assignments and Data Constraints. Let *Data* be a finite and non-empty set of data items that can be transferred through channels, and $\mathcal{N}$ a finite and non-empty set of node names. A data assignment δ denotes a function $\delta : N \to Data$ where $\varnothing \neq N \subseteq \mathcal{N}$. All possible data assignments on N is denoted as $DA(N)$ or $Data^N$. For a subset $N_0 \subseteq N$, the restriction of δ over N_0 is a data assignment $\delta\restriction_{N_0} \in DA(N_0)$ defined as $\delta\restriction_{N_0}(A) = \delta(A)$ for each $A \in N_0$. We use the notation of $\delta = [A \mapsto d \mid A \in N]$ to specify a data assignment that assigns a value $d \in Data$ to every node $A \in N$. For example, if d_1 is transferred through node A and d_2 is transferred through node B, then the corresponding data assignment is $\delta = [A \mapsto d_1, B \mapsto d_2]$.

Formally, a data constraint g over N is a propositional formula built from the atoms such as "$d_A \in P$" and "$d_A = d_B$" and boolean operators $\wedge, \vee, \neg$, etc. where $A, B \in \mathcal{N}$, $P \subseteq Data$ and d_A is interpreted as $\delta(A)$. For $N \subseteq \mathcal{N}$, $DC(N)$ denotes the set of all data constraints that specify values being transferred on nodes in N. We use $\delta \vDash g$ to denote that the data assignment δ satisfies the data constraint g.

Example 1. Let $N = \{A, B, C\}$ and $Data = \{d_0, d_1\}$. Data assignment $\delta = [A \mapsto d_1, C \mapsto d_0]$ says that d_1 and d_0 are transferred through nodes A and C respectively, while no data item is transferred through B. Let $g_1 = (d_A = d_1)$ and $g_2 = (d_A = d_C)$ be two data constraints, then $\delta \vDash g_1$ and $\delta \nvDash g_2$.

Dynamical Systems and Space Constraints. Dynamical systems can model systems with continuous behaviors. Consider the differential equation:

$$\dot{\xi} = f(\xi) \tag{1}$$

where the dotted variables represent the first derivatives during continuous change and $f : \mathbb{R}^n \to \mathbb{R}^n$ is an infinitely differentiable function. We also call such functions *smooth*. By a trajectory of (1) with initial condition $x \in \mathbb{R}^n$, we mean a smooth curve

$$\xi : [0, \tau) \to \mathbb{R}^n \tag{2}$$

satisfying

- $\tau > 0$;
- $\xi(0) = x$;
- $\dot{\xi}(t) = f(\xi(t))$ for each $t \in (0, \tau)$.

In this case, we say the duration of the trajectory ξ is τ.

Definition 1 (Dynamical system). *An n-dimensional dynamical system* $\Sigma = (\mathbb{R}^n, f)$ *is the real space* $\mathbb{R}^n$ *equipped with differential equation given by a smooth map* $f : \mathbb{R} \to \mathbb{R}^n$. *A trajectory of a dynamical system is a trajectory of the differential equation defined by* f.

We also consider systems $(\mathcal{X}, f)$ with f defined in a subset $\mathcal{X}$ of $\mathbb{R}^n$. A trajectory ξ of the dynamical system $\Sigma = (\mathcal{X}, f)$ with a duration τ and initial condition $x \in \mathcal{X}$ is a solution to (1) satisfying

- $\tau > 0$;
- $\xi(0) = x$;
- $\dot{\xi}(t) = f(\xi(t))$ for each $t \in (0, \tau)$;
- $\xi(t) \in \mathcal{X}$ for each $t \in [0, \tau)$.

Intuitively, an n-dimensional dynamical system $\Sigma = (\mathbb{R}^n, f)$ describes how a point P evolves and flows in space $\mathbb{R}^n$ based on the rules given as differential equations. If at present time $t = t_0$, the coordination of P's location is $x_0 \in \mathbb{R}^n$, then in the near future P follows a trajectory ξ with duration τ and initial condition x_0. At time $t = t_0 + \Delta t$ where $\Delta t < \tau$, P will locate in point $\xi(\Delta t) \in \mathbb{R}^n$. This intuition makes sense in that if there are two trajectories, ξ_1 with duration τ_1 and ξ_2 with duration τ_2 sharing the same initial condition, then $\xi_1(t) = \xi_2(t)$ for each $t \in [0, \min\{\tau_1, \tau_2\})$. This is concluded in Theorem 1 which comes directly from the Peano existence theorem [13], a fundamental theorem in the study of ordinary differential equations that guarantees the existence of solutions to certain initial value problems.

Theorem 1. *Let* $\mathcal{X} \subseteq \mathbb{R}^n$ *be a nonempty subset,* $f : \mathcal{X} \to \mathbb{R}^n$ *a continuously differentiable function, and* $x_0 \in \mathcal{X}$ *an interior point. Then there exists some* $\tau > 0$ *and a unique solution* $\xi : [0, \tau) \to \mathcal{X}$ *of the differential equation* $\dot{\xi} = f(\xi)$ *satisfying* $\xi(0) = x_0$.

A space constraint φ to an n-dimensional dynamical system Σ is defined as a predicate over free variables $\{\#_1, \#_2, \cdots, \#_n\}$ where $\#_i$ is interpreted as the i-th coordinate of a point $x \in \mathbb{R}^n$ for each $i \in \{1, 2, \cdots, n\}$. If the dimension $n = 1$, then we abbreviate $\#_1$ as $\#$. The set of all n-dimensional space constraints is denoted as $SC(n)$ or SC. We use $x \vDash \varphi$ to denote that the point x in space $\mathbb{R}^n$ satisfies the space constraint φ.

Example 2. Let x_1, x_2, x_3 be three points in space $\mathbb{R}^3$ with their coordinations:

$$x_1 = (1, 2, 3)\ ,\ x_2 = (1, 0, -1)\ ,\ x_3 = (0, 0, 0) \tag{3}$$

and φ_1 and φ_2 two space constraints defined as

$$\varphi_1 = (\#_1 = \#_2) \wedge (\#_2 \leq \#_3)\ ,\ \varphi_2 = (\#_1 + \#_2 + \#_3 = 0)\ . \tag{4}$$

Then we have

$$\begin{aligned} &x_1 \nvDash \varphi_1\ ,\ x_1 \nvDash \varphi_2\ , \\ &x_2 \nvDash \varphi_1\ ,\ x_2 \vDash \varphi_2\ , \\ &x_3 \vDash \varphi_1\ ,\ x_3 \vDash \varphi_2\ . \end{aligned} \tag{5}$$

Hybrid Constraint Automata. We now give the formal definition of HCA and some intuitive interpretation on how it operates.

Definition 2 (Hybrid constraint automata). *A hybrid constraint automata (HCA) $\mathcal{T}$ is a tuple $(S, \mathbb{R}^n, \mathcal{N}, \mathcal{E}, IS, \{In_s\}_{s\in S}, \{f_s\}_{s\in S}, \{re_t\}_{t\in\mathcal{E}})$ consisting of*

- *a finite set of control states S and a set of initial control states $IS \subseteq S$;*
- *the dynamical system space $\mathbb{R}^n$;*
- *a finite set of nodes $\mathcal{N}$;*
- *an n-dimensional dynamical system $\Sigma_s = (In_s, f_s)$ for each $s \in S$;*
- *an edge relation $\mathcal{E}$ which is a subset of $S \times 2^{\mathcal{N}} \times DC \times SC \times S$;*
- *a reset function $re_{(s,N,g,\varphi,\bar{s})} : Data^N \times In_s \to In_{\bar{s}}$.*

Instead of writing $(s, N, g, \varphi, \bar{s}) \in \mathcal{E}$, we use $s \xrightarrow{N,g,\varphi} \bar{s}$. If $re_{(s,N,g,\varphi,\bar{s})} = r$ then we say $s \xrightarrow[r]{N,g,\varphi} \bar{s}$.

The intuitive interpretation of how an HCA $\mathcal{T}$ operates is as follows. In the beginning, $\mathcal{T}$ stays in one of the initial control states $s_0 \in IS$ and behaves exactly as the dynamical system Σ_{s_0}, that is, it starts with a point $x_0 \in In_{s_0}$ and then flows based on the differential equation given by f_{s_0}. If $\mathcal{T}$ stays in control state $s \in S$ and locates at point $x \in In_s$, it

- must choose an edge $t = (s, N, g, \varphi, \bar{s})$ from $\mathcal{E}$ such that the data assignment $\delta \vDash g$ and $x \vDash \varphi$, if some I/O operations specified by δ happen on exact those nodes in N. If more than one edges are available, $\mathcal{T}$ chooses one of them nondeterministically. If $\mathcal{T}$ chooses $t = (s, N, g, \varphi, \bar{s})$, it successfully accepts I/O operations and jumps to control state $\bar{s}$ and then behaves exactly as the dynamical system $\Sigma_{\bar{s}}$ with initial condition $re_t(\delta, x) \in In_{\bar{s}}$. If no such edge is available, $\mathcal{T}$ halts;
- must choose an edge $t = (s, N, g, \varphi, \bar{s})$ from $\mathcal{E}$ where $N = \varnothing$ and $g = [\,]$ such that $x \vDash \varphi$, if $\mathcal{T}$ is about to violate the invariant In_s and no I/O operation happens at the time. If more than one edges are available, $\mathcal{T}$ chooses one of them nondeterministically. If $\mathcal{T}$ chooses $t = (s, \varnothing, [\,], \varphi, \bar{s})$, it jumps to control state $\bar{s}$ and then behaves exactly as the dynamical system $\Sigma_{\bar{s}}$ with initial condition $re_t([\,], x) \in In_{\bar{s}}$. If no such edge is available, $\mathcal{T}$ halts;
- may stay in the control state s and behaves exactly as the dynamical system Σ_s as long as it is not forced to make a jump to a new control state.

3.2 The State-Transition Graph of an HCA

So far we described the syntax of HCA and gave some intuitive explanations for their meaning. The following definition formalizes this intuitive behavior by means of a state-transition graph.

Definition 3 (State-transition graph). *Given an HCA $\mathcal{T} = (S, \mathbb{R}^n, \mathcal{N}, \mathcal{E}, IS, \{In_s\}_{s\in S}, \{f_s\}_{s\in S}, \{re_t\}_{t\in\mathcal{E}})$ as above, $\mathcal{T}$ induces a state-transition graph $\mathcal{A}_\mathcal{T} = (Q, \longrightarrow, IQ)$ consisting of*

- *a set of states* $Q = \{\langle s, x\rangle \mid s \in S \wedge x \in In_s\}$;
- *a set of initial states* $IQ = \{\langle s, x\rangle \mid s \in IS \wedge x \in In_s\}$;
- *a transition relation* $\longrightarrow \subseteq Q \times 2^{\mathcal{N}} \times DA \times \mathbb{R}_{\geq 0} \times Q$;

where $\langle s, x\rangle \xrightarrow{N,\delta,\tau} \langle \bar{s}, \bar{x}\rangle$ *if and only if one of the following conditions holds:*

- *(Flows)* $s = \bar{s}$, $N = \varnothing$, $\delta = [\,]$, $\tau > 0$ *and there exists a trajectory* ξ *with duration* τ *and initial condition* x *in the dynamical system* $\Sigma_s = (In_s, f_s)$. *The trajectory heads for the point* $\bar{x}$, *that is*
$$\lim_{t \to \tau^-} \xi(t) = \bar{x}\,; \tag{6}$$
- *(External interactions)* $s \xrightarrow[re]{N,g,\varphi} \bar{s}$, $N \neq \varnothing$, $\delta \in DA(N)$, $\delta \vDash g$, $x \vDash \varphi$, $\tau = 0$ *and* $\bar{x} = re(\delta, x)$;
- *(Internal jumps)* $s \xrightarrow[re]{N,g,\varphi} \bar{s}$, $N = \varnothing$, $\delta = [\,]$, $g = true$, $x \vDash \varphi$, $\tau = 0$ *and* $\bar{x} = re([\,], x)$.

According to Definition 3, transitions in a state-transition graph $\mathcal{A}_{\mathcal{T}}$ are disjointly divided into three categories: flows, external interactions (or briefly interactions) and internal jumps (or briefly jumps). Both interactions and jumps are discrete behavior while flows are continuous. Given a state $q \in Q$, a successor of q is a state $p \in Q$ such that there exists a transition $q \xrightarrow{N,\delta,\tau} p$ in $\mathcal{A}_{\mathcal{T}}$. If this transition is a flow, then p is called a *flow-successor* of q with duration τ. Similarly, we can define *interaction-successor* and *jump-successor*. A state $q = \langle s, x\rangle$ is called *terminal* if and only if it has no outgoing transition.

Given an HCA $\mathcal{T}$ and a state $q = \langle s, x\rangle$ in $\mathcal{A}_{\mathcal{T}}$, a *q-run* (or briefly run) in $\mathcal{T}$ denotes any finite or infinite sequence of successive transitions in $\mathcal{A}_{\mathcal{T}}$ starting in state q. Formally, a q-run has the form

$$\varrho = q_0 \xrightarrow{N_0,\delta_0,\tau_0} q_1 \xrightarrow{N_1,\delta_1,\tau_1} \cdots \tag{7}$$

where $q_0 = q$. It is required that for any sequence segment

$$q_i \xrightarrow{N_i,\delta_i,\tau_i} q_{i+1} \xrightarrow{N_{i+1},\delta_{i+1},\tau_{i+1}} q_{i+2} \tag{8}$$

in ϱ, exactly one of the two transitions is flow for the following reasons. If a run ϱ contains two consecutive flow-transitions, say, $q_i \xrightarrow{\varnothing,[\,],\tau_i} q_{i+1} \xrightarrow{\varnothing,[\,],\tau_{i+1}} q_{i+2}$, then it can be replaced by one flow-transition $q_i \xrightarrow{\varnothing,[\,],\tau_i+\tau_{i+1}} q_{i+2}$ without any change of its behavior. On the other hand, if ϱ contains two consecutive discrete actions (interaction- or jump-transition), then these actions occur at the same time point, which violates the general idea of constraint automata where all observable activities that occur simultaneously are collapsed into a single transition. Therefore, a run ϱ in HCA actually consists of an alternating sequence of continuous transitions (flows) and discrete actions (interactions or jumps).

Let $t = q \xrightarrow{N,\delta,\tau} \bar{q}$ be a transition in $\mathcal{A}_{\mathcal{T}}$, we introduce some abbreviate notations as follows:

- instead of writing $q \xrightarrow{\varnothing,[],\tau} \bar{q}$, we say $q \xrightarrow{\tau} \bar{q}$ if t is a flow-transition. Under this circumstances, $\tau > 0$;
- instead of writing $q \xrightarrow{N,\delta,0} \bar{q}$, we say $q \xrightarrow{N,\delta} \bar{q}$ if t is an interaction-transition. Under this circumstances, $N \neq \varnothing$;
- instead of writing $q \xrightarrow{\varnothing,[],0} \bar{q}$, we say $q \xrightarrow{0} \bar{q}$ if t is a jump-transition.

The q-run ϱ is called initial if $q \in IQ$ and the first transition of ϱ is a flow. The q-run ϱ is called *time divergent* if ϱ is infinite and

$$\lim_{n\to+\infty} \sum_{i=0}^{n} \tau_i = +\infty \ . \tag{9}$$

For an initial run ϱ, instead of using general notation as in (7), we use the following simplified notation:

$$\varrho = q_0 \xrightarrow{\tau_0} q_1 \xrightarrow{N_1,\delta_1} q_2 \xrightarrow{\tau_2} q_3 \xrightarrow{N_3,\delta_3} \cdots \tag{10}$$

where the notation $q_1 \xrightarrow{N_1,\delta_1} q_2$ should be regarded as an interaction-transition if $N_1 \neq \varnothing$ or a jump-transition if $N_1 = \varnothing$ and $\delta_1 = [\,]$. Maximality of a run means that it is either time divergent or finite and ends in a terminal state.

Intuitively, N_i is the set of nodes in state q_i that are scheduled to synchronously perform the next set of I/O operations, while δ_i represents the concrete values that are exchanged through those operations at the nodes $A \in N_i$. The value τ_i stands for the duration time when the system evolves based on differential equations.

We now define the notion of *timed data stream* (TDS) which serves to formalize the observable data flows of the runs in an HCA and thus formally define the behavior of an HCA. A TDS is a sequence of triples (N, δ, t) where N is a non-empty set of nodes, δ is a data assignment over N and t is a time point. The intuitive meaning of (N, δ, t) is that at time t the nodes in N simultaneously perform some I/O-operations specified by the pair (N, δ).

Definition 4 (Timed data stream). *A timed data stream for a node-set $\mathcal{N}$ denotes any finite or infinite sequence*

$$\Theta = (N_0, \delta_0, t_0)\ ,\ (N_1, \delta_1, t_1)\ ,\ \cdots\ \in (2^{\mathcal{N}} \times DA \times \mathbb{R}_{\geq 0})^* \tag{11}$$

such that $N_i \neq \varnothing$, $\delta_i \in DA(N_i)$, $0 < t_0 < t_1 < \cdots$. The empty timed data stream is denoted by the symbol ε. The length $\|\Theta\| \in \mathbb{N} \cup \{\infty\}$ is defined as the number of triples (N, δ, t) in Θ. The execution time

$$\tau(\Theta) = \begin{cases} t_k & \|\Theta\| = k+1 \\ \lim\limits_{i\to+\infty} t_i & \|\Theta\| = \infty \\ 0 & \Theta = \varepsilon \end{cases}$$

Θ is called time divergent if it is infinite and $\tau(\Theta) = +\infty$.

Definition 5 (Timed data stream language). *If ϱ is a run of HCA $\mathcal{T}$ as above then the induced TDS $\Theta(\varrho) = (N_{i_0}, \delta_{i_0}, t_{i_0})\ ,\ (N_{i_1}, \delta_{i_1}, t_{i_1})\ ,\ \cdots$ is obtained by*

1. *removing all flow- and jump-transitions in ϱ;*
2. *building the projection on the transition labels;*
3. *replacing the duration time τ_i by the absolute time points $t_i = \sum_{j=0}^{i} \tau_i$.*

The generated TDS language of a state q in $\mathcal{A}_\mathcal{T}$ is

$$\mathcal{L}(\mathcal{T}, q) = \{\Theta(\varrho) : \varrho \text{ is a maximal } q\text{-run}\} . \tag{12}$$

The language $\mathcal{L}(\mathcal{T})$ consists of all timed data streams $\Theta(\varrho)$ where ϱ is a maximal and initial run.

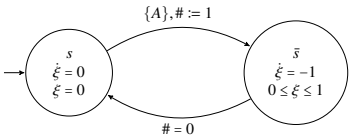

Fig. 2. HCA for a delay channel

Example 3. Let $Data = \{d\}$ and $\mathcal{N} = \{A\}$. Fig. 2 shows an HCA $\mathcal{T}$ with the set of control states $S = \{s, \bar{s}\}$ and the initial control state $s \in S$ [1]. $\mathcal{T}$ has two edges: $s \xrightarrow[\#:=1]{\{A\},true,true} \bar{s}$ and $\bar{s} \xrightarrow[\sigma_x]{\varnothing,true,\#=0} s$. Here, $\# := 1$ is an abbreviation of the function $re : [A \mapsto d] \times \{0\} \mapsto \{1\}$ and $\sigma_x : Data^\mathcal{N} \to \mathbb{R}$ is the projection function satisfying $\sigma_x(\delta, x) = x$ for any $\delta \in DA(\mathcal{N})$ and $x \in \mathbb{R}$. According to Definition 3, we can give the corresponding state-transition graph $\mathcal{A}_\mathcal{T} = (Q, \longrightarrow, IQ)$ of the HCA $\mathcal{T}$, where $Q = \{\langle s, 0\rangle\} \cup \{\langle \bar{s}, x\rangle \mid 0 \le x \le 1\}$, $IQ = \{\langle s, 0\rangle\}$ and $\longrightarrow$ consists of

- flow-transitions in control state s, that is $\langle s, 0\rangle \xrightarrow{\varnothing,[],\tau} \langle s, 0\rangle$ for each $\tau > 0$;
- flow-transitions in control state $\bar{s}$, that is $\langle \bar{s}, x\rangle \xrightarrow{\varnothing,[],\tau} \langle \bar{s}, x - \tau\rangle$ for each $x \in (0, 1]$ and $0 < \tau \le x$;
- an interaction-transition $\langle s, 0\rangle \xrightarrow{\{A\},[A\mapsto d],0} \langle \bar{s}, 1\rangle$;
- a jump-transition $\langle \bar{s}, 0\rangle \xrightarrow{\varnothing,[],0} \langle s, 0\rangle$.

The intuitive interpretation of how $\mathcal{A}_\mathcal{T}$ works is as follows. At the beginning, $\mathcal{A}_\mathcal{T}$ stays in state $\langle s, 0\rangle$, where there are two outgoing transitions: one is an interaction-transition to state $\langle \bar{s}, 1\rangle$ and the other is a self-loop flow-transition to state $\langle s, 0\rangle$ itself. Therefore if no I/O-transition is performed, then $\mathcal{A}_\mathcal{T}$ must stay in state $\langle s, 0\rangle$, until some I/O-operations happen. Because $\mathcal{T}$ has only one node A and the data set $Data$ contains only one data item d, the only I/O-operation that can happen here is the one specified by the data assignment $\delta = [A \mapsto d]$, which triggers $\mathcal{A}_\mathcal{T}$ moving to state $\langle \bar{s}, 1\rangle$ through the only interaction-transition $\langle s, 0\rangle \xrightarrow{\{A\},[A\mapsto d],0} \langle \bar{s}, 1\rangle$. From then on, $\mathcal{A}_\mathcal{T}$ will flow based

[1] To make the graph simple and clear, here we omit all the trivial conditions and labels such as the projection function σ_x, the *true* predicate and empty node-set $\varnothing$.

on the differential equation $\dot{\xi} = -1$. Notice that in any state $\langle \bar{s}, x \rangle$ where $x \in (0, 1]$, the only outgoing transitions for $\mathcal{A}_{\mathcal{T}}$ is flow-transitions, which implies that $\mathcal{A}_{\mathcal{T}}$ will stay in control state $\bar{s}$ and keep flowing until it reaches the state $\langle \bar{s}, 0 \rangle$. As soon as it reaches $\langle \bar{s}, 0 \rangle$, it will choose the only outgoing transition $\langle \bar{s}, 0 \rangle \xrightarrow{\varnothing,[],0} \langle s, 0 \rangle$ and finally come back to the initial state $\langle s, 0 \rangle$.

A typical maximal and initial run ϱ of the HCA $\mathcal{T}$ has the form

$$\varrho = \langle s, 0 \rangle \xrightarrow{\tau_1} \langle s, 0 \rangle \xrightarrow{\{A\}, [A \mapsto d]} \langle \bar{s}, 1 \rangle \xrightarrow{1} \langle \bar{s}, 0 \rangle \xrightarrow{0} \langle s, 0 \rangle \xrightarrow{\tau_2} \cdots \tag{13}$$

where $\tau_1, \tau_2, \ldots$ are positive real numbers and $\tau(\varrho) = \tau_1 + 1 + \tau_2 + 1 + \cdots = +\infty$, which means ϱ is time divergent. The corresponding TDS $\Theta(\varrho)$ induced by $\mathbf{q}$ is

$$\Theta(\varrho) = (\{A\}, [A \mapsto d], \tau_1), (\{A\}, [A \mapsto d], \tau_1 + 1 + \tau_2), \cdots \tag{14}$$

and the TDS-language $\mathcal{L}(\mathcal{T})$ of $\mathcal{T}$ is set of sequences $(\{A\}, [A \mapsto d], t_1), (\{A\}, [A \mapsto d], t_2), \cdots$ where $t_{i+1} - t_i > 1$ for each $i \geq 1$.

4 Hybrid Reo Circuits

This section explains how HCA is able to formalize connectors with hybrid behaviors in Reo in a compositional way.

4.1 Hybrid Primitive Channels

Reo defines what a channel is and how channels can be composed into more complex connectors. Reo places no restrictions on the behavior of channels. This allows an open-ended set of user-defined channel types as primitives for constructing complex connectors (also called circuits in Reo). In the sequel, we introduce a number of common channel types when considering the hybrid behavior of CPSs.

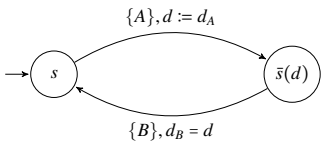

Fig. 3. HCA for FIFO1 channel

FIFO channels. FIFO channels are the most common form of asynchronous channels. The word "asynchronous" here means that there exists some delay after a data item is written into the input port for the data item to be available on the output port. The simplest FIFO channel with discrete behavior only is the FIFO1 channel. A FIFO1 channel is a FIFO channel with one buffer cell, which has a source end and a sink end. The corresponding HCA for the FIFO1 channel is shown in Fig. 3, where all the trivial conditions and labels are omitted intentionally.

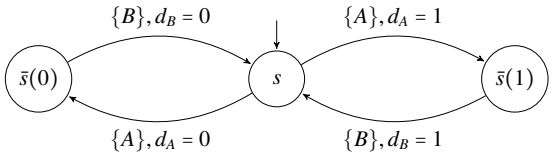

Fig. 4. Non-parametric HCA for FIFO1 channel

Note that in Fig. 3 we use a parametric notation for HCA which can be easily unfolded to a standard HCA as in Definition 2. For example, let $Data = \{0, 1\}$ be the set of data items, the unfolded non-parametric HCA is given in Fig. 4.

There are FIFO channels with some time properties such as the expiring FIFO1 channel, where a data item is lost if it is not taken out from the buffer through the sink end within τ time units after it enters the source end. The HCA for an expiring FIFO1 channel is shown in Fig. 5. The edge from s to $\bar{s}(d)$ models a write action on the source end A, which triggers the HCA moving to control state $\bar{s}(d)$, where the differential equation $\dot{\xi} = -1$ forces the automata to flow from the point $\tau \in \mathbb{R}$ towards $0 \in \mathbb{R}$. If no interaction-transition is available, i.e., the sink end B is not ready for a take operation, then the automata will reach the point $0 \in \mathbb{R}$ finally and immediately jump to the control state s trough $\bar{s}(d) \xrightarrow{\#\,=\,t} s$ in order to avoid violating the invariant predicate $\xi \geq 0$. Under this circumstance, the channel loses the data item in its buffer, which is exactly the behavior as we supposed. It is also possible that when the automaton is in the control state $\bar{s}(d)$ a take operation happens on the sink end B. This will force the automata to accept the I/O-operation and move back to the initial state s through $\bar{s}(d) \xrightarrow[\#\,:=\,0]{\{B\},\, d_B\,=\,d,\, \#\,\geq\,0} s$.

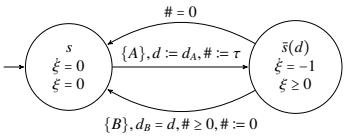

Fig. 5. HCA for expiring FIFO1 channel

A more interesting example is the data-sensitive FIFO1 channel where the behavior is determined by not only the external environment (i.e., the I/O-operations on its channel ends) but also the data items which are transferred through the channel. A typical example is a variant of a standard FIFO1 channel where after a write operation happens on the source end A, the times it takes to "transfer" the data item from A to the sink end B depends on the size of the data item being transferred. Let $size : Data \to \mathbb{R}_{>0}$ be a primitive function where $size(d)$ gives the size of data item $d \in Data$ and a constant data transferring speed $k \in \mathbb{R}_{>0}$. The HCA for such a channel is shown in Fig. 6. The edge $s \xrightarrow[\#\,:=\,size(d)]{\{A\},d:=d_A} \bar{s}(d)$ models a write operation on A which forces the automata to

move to the control state $\bar{s}(d)$ and locates in $size(d) \in \mathbb{R}$. When the automata stays in $\bar{s}(d)$, it flows based on the differential equation $\dot{\xi} = -k$ in the negative direction in $\mathbb{R}$. Notice the invariant set of $\bar{s}(d)$ is the entire space $\mathbb{R}$, therefore the automata is allowed to stay in the control state $\bar{s}(d)$ as long as there is no I/O-operation succeeds on B. If a take operation successfully happens on B, then the automata checks all legal outgoing transitions from $\bar{s}(d)$. Since the only outgoing transition is $\bar{s}(d) \xrightarrow[\#:=0]{\{B\}, d_B = d, \# \leq 0} s$ where the space constraint is $\# \leq 0$, the automata is able to make the transition only when it reaches the non-positive part in $\mathbb{R}$, that is at least $size(d)/k$ time units after the write operation happened on A.

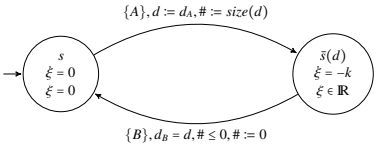

Fig. 6. HCA for constant speed transferring FIFO1 channel

4.2 Join on HCA

In Reo, complex circuits can be composed by primitive channels. We now define the composition operator on HCA that serves to formalize Reo circuits in a compositional way.

Definition 6 (HCA product). *Let $\mathcal{T}_1$ and $\mathcal{T}_2$ be two HCA*

$$\mathcal{T}_i = (S_i, \mathbb{R}^{n_i}, \mathcal{N}_i, \mathcal{E}_i, IS_i, \{In_s\}_{s \in S_i}, \{f_s\}_{s \in S_i}, \{re_t\}_{t \in \mathcal{E}_i}) \tag{15}$$

where $i \in \{1, 2\}$ such that the set of all shared nodes $\mathcal{N}_0 = \mathcal{N}_1 \cap \mathcal{N}_2$. The product $\mathcal{T}_1 \bowtie \mathcal{T}_2$ is defined as an HCA

$$\mathcal{T} = (S, \mathbb{R}^n, \mathcal{N}, \mathcal{E}, IS, \{In_s\}_{s \in S}, \{f_s\}_{s \in S}, \{re_t\}_{t \in \mathcal{E}}) \tag{16}$$

consisting of

- *a set of control states $S = S_1 \times S_2$ and a set of initial control states $IS = IS_1 \times IS_2$;*
- *the dynamical system space $\mathbb{R}^n$ where $n = n_1 + n_2$;*
- *a finite set of nodes $\mathcal{N} = \mathcal{N}_1 \cup \mathcal{N}_2$;*
- *an n-dimensional dynamical system $\Sigma_s = (In_s, f_s)$ for each $s = \langle s_1, s_2 \rangle \in S$, where $In_s = In_{s_1} \times In_{s_2}$, and $f_s : In_s \to \mathbb{R}^n$ is a function defined as $f_s(x_1, x_2) = (f_{s_1}(x_1), f_{s_2}(x_2))$ for each $x_1 \in In_{s_1}$ and $x_2 \in In_{s_2}$;*
- *an edge relation $\mathcal{E}$ which is a subset of $S \times 2^{\mathcal{N}} \times DC \times SC \times S$;*
- *a reset function $re_t : Data^N \times In_s \to In_{\bar{s}}$ for each $t = (s, N, g, \varphi, \bar{s}) \in \mathcal{E}$.*

Intuitively, the compositional product HCA $\mathcal{T}$ behaves exactly as the parallel of $\mathcal{T}_1$ and $\mathcal{T}_2$, with the only constraint that all I/O-operations happen on the shared ports in N should coincide with each other. Therefore, $\langle s_1, s_2\rangle \xrightarrow[re]{N, g, \varphi} \langle \bar{s}_1, \bar{s}_2\rangle \in \mathcal{E}$ is defined by the following rules.

– *The first rule deals with the situation when $\mathcal{T}_1$ and $\mathcal{T}_2$ are about to do some I/O-operations on the nodes in N_1 and N_2 respectively at the same time. This is allowed only when they coincide on the shared nodes, that is*

$$\frac{\begin{array}{c} s_1 \xrightarrow[re_1]{N_1, g_1, \varphi_1} \bar{s}_1 \in \mathcal{E}_1 \\ s_2 \xrightarrow[re_2]{N_2, g_2, \varphi_2} \bar{s}_2 \in \mathcal{E}_2 \\ N_1 \cap \mathcal{N}_0 = N_2 \cap \mathcal{N}_0 \\ g_1 \wedge g_2 \not\equiv false \\ \varphi_1 \wedge \varphi_2 \not\equiv false \end{array}}{\langle s_1, s_2\rangle \xrightarrow[re]{N_1 \cup N_2, g_1 \wedge g_2, \varphi_1 \wedge \varphi_2} \langle \bar{s}_1, \bar{s}_2\rangle \in \mathcal{E}} \tag{17}$$

where $re : Data^N \times In_{\langle s_1, s_2\rangle} \to \langle \bar{s}_1, \bar{s}_2\rangle$ is a function defined as follows. For each $\delta \in Data^N$ and $i \in \{1, 2\}$, let $\delta_i \in Data^{N_i}$ satisfy $\delta_i(A) = \delta(A)$ for each $A \in N_i$. For each $\delta \in Data^N$ and $\langle x_1, x_2\rangle \in \langle In_{s_1}, In_{s_2}\rangle$,

$$re(\delta, \langle x_1, x_2\rangle) = \langle re_1(\delta_1, x_1), re_2(\delta_2, x_2)\rangle.$$

– *The second rule deals with the situation when $\mathcal{T}_1$ is about to make a discrete transition while $\mathcal{T}_2$ continues in flowing. This is allowed if $\mathcal{T}_1$'s transition does not ask $\mathcal{T}_2$ to coordinate with it, that is*

$$\frac{\begin{array}{c} s_1 \xrightarrow[re_1]{N_1, g_1, \varphi_1} \bar{s}_1 \in \mathcal{T}_1 \\ s_2 \in S_2 \\ N_1 \cap \mathcal{N}_0 = \varnothing \end{array}}{\langle s_1, s_2\rangle \xrightarrow[re]{N_1, g_1, \varphi_1} \langle \bar{s}_1, s_2\rangle} \tag{18}$$

where $re : Data^N \times In_{\langle s_1, s_2\rangle} \to \langle \bar{s}_1, \bar{s}_2\rangle$ is a function defined as $re(\delta, \langle x_1, x_2\rangle) = re_1(\delta, x_1)$ which is well defined since $N = N_1$. There is a symmetric rule which deals with the situation when $\mathcal{T}_2$ is about to make a discrete transition while $\mathcal{T}_1$ continues in flowing:

$$\frac{\begin{array}{c} s_2 \xrightarrow[re_2]{N_2, g_2, \varphi_2} \bar{s}_2 \in \mathcal{T}_2 \\ s_1 \in S_1 \\ N_2 \cap \mathcal{N}_0 = \varnothing \end{array}}{\langle s_1, s_2\rangle \xrightarrow[re]{N_2, g_2, \varphi_2} \langle s_1, \bar{s}_2\rangle} \tag{19}$$

Roughly speaking, the product HCA $\mathcal{T}$ needs to deal with three situations:

- both HCA choose to make discrete transitions. This is captured by (17);
- one of the HCA chooses to make a discrete transition while the other chooses to stay in current control state and continues in flowing. This is captured by (18) and (19).
- both HCA choose to stay in their current control states respectively and continue in flowing. This is captured by the composed dynamical systems $\Sigma_{\langle s_1,s_2\rangle}$.

Here we introduce some convenient notations for the join operation. For $s = \langle s_1, s_2\rangle$, we use $s.first$, $s.second$ to denote s_1, s_2 respectively. Similarly, for $x = \langle x_1, x_2\rangle$, we use $x.first$, $x.second$ to denote x_1, x_2 respectively.

The join operator introduced in Definition 6 captures the replicator semantics of source nodes in Reo. Therefore it can serve as the semantic operator for the join of two nodes where at least one of them is a source node. To mimic the merge semantics of sink nodes, we introduce the HCA $\mathcal{T}_{Merger}$ shown in Fig. 7. To join two sink nodes A and B, we first choose a new node named C and then return $\mathcal{T}_{Merger}(A, B, C) \bowtie \mathcal{T}_A \bowtie \mathcal{T}_B$ where $\mathcal{T}_A$ and $\mathcal{T}_B$ are the HCA that model the sub-circuits containing A and B respectively.

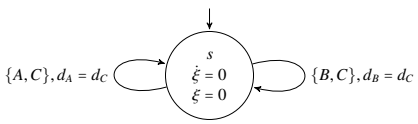

Fig. 7. HCA for merger

The correctness of the join operator on HCA is guaranteed by means of TDS-languages. For this purpose, we define the join operator on TDS-languages and establish a compositionality result in Theorem 2.

Definition 7 (Join on timed data streams and TDS languages). *Let $\Theta = ((N_i, \delta_i, t_i))_i$ and $\Phi = ((M_j, \gamma_j, s_j))_j$ be two TDS over $\mathcal{N}$ and $\mathcal{M}$ respectively. The common node-set is denoted as $\mathcal{N}_0 = \mathcal{N} \cap \mathcal{M}$. We say that Θ and Φ are inconsistent if there exist $i \in \mathbb{N}$ and $j \in \mathbb{N}$ such that $t_i = s_j$, $(N_i \cup M_j) \cap \mathcal{N}_0 \neq \varnothing$ and $\delta_i \restriction_{\mathcal{N}_0} \neq \gamma_j \restriction_{\mathcal{N}_0}$. We say that Θ and Φ are consistent if they are not inconsistent. The join $\Theta \bowtie \Phi$ of two consistent TDS can be inductively defined as a sequence generated by*

- *appending (N_1, δ_1, t_1) to $\Theta' \bowtie \Phi$, if $t_1 < s_1$;*
- *appending (M_1, γ_1, s_1) to $\Theta \bowtie \Phi'$, if $s_1 < t_1$;*
- *appending $(N_1 \cup M_1, \delta_1 \cup \gamma_1, t_1)$ to $\Theta' \bowtie \Phi'$, if $s_1 = t_1$. $\delta_1 \cup \gamma_1$ is well defined since Θ and Φ are consistent.*

Let L_1 and L_2 be two TDS-languages over $\mathcal{N}_1$ and $\mathcal{N}_2$ respectively. The join $L_1 \bowtie L_2$ is a TDS-language over $\mathcal{N}_1 \cup \mathcal{N}_2$ consists of all timed data streams Θ that can be obtained by joining two consistent timed data streams $\Theta_1 \in L_1$ and $\Theta_2 \in L_2$.

Lemma 1. *Let $\mathcal{T}_1$ and $\mathcal{T}_2$ be HCA, then*

$$\mathcal{L}(\mathcal{T}_1 \bowtie \mathcal{T}_2) = \mathcal{L}(\mathcal{T}_1) \bowtie \mathcal{L}(\mathcal{T}_2)\ . \tag{20}$$

Lemma 1 directly leads to the following compositional theorem, which implies the correctness of the product operator on HCA.

Theorem 2. *Let $\mathcal{T}_1$, $\mathcal{T}_2$ and $\mathcal{T}_3$ be HCA, then*

$$\mathcal{L}((\mathcal{T}_1 \bowtie \mathcal{T}_2) \bowtie \mathcal{T}_3) = \mathcal{L}(\mathcal{T}_1 \bowtie (\mathcal{T}_2 \bowtie \mathcal{T}_3)) \tag{21}$$

Example 4. Figure 8 shows a Reo circuit consisting of two expiring FIFO1 channels, with expiring limit τ and ω respectively. The HCA for the two channels and the whole circuit obtained by their join are shown in Fig. 9.

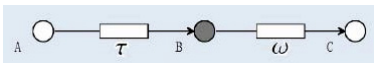

Fig. 8. The Reo circuit obtained by joining two expiring FIFO1 channels

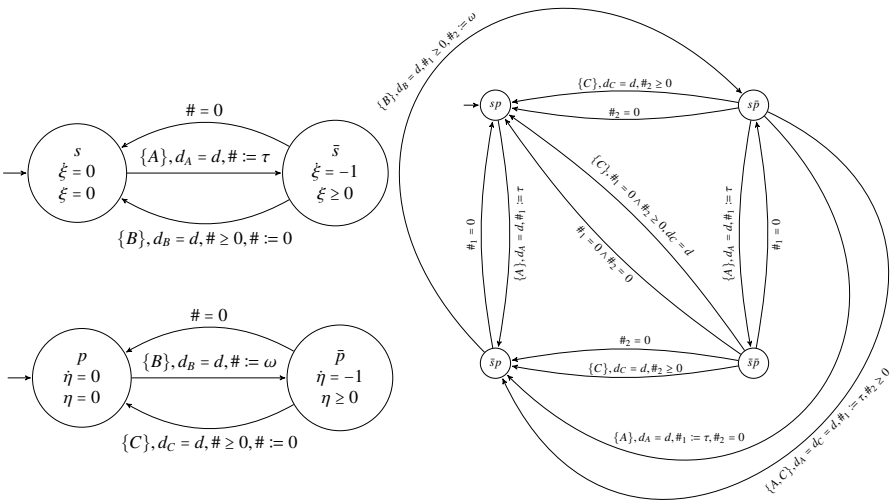

Fig. 9. HCA for two expiring FIFO1 channels and the Reo circuit consisting of them

5 Conclusion

In this paper we introduced hybrid constraint automata (HCA) as a formal model to define hybrid behavior for Reo component connectors. We presented a compositional product operator that can serves as a basis for the automated construction of a hybrid

constraint automaton model from a given Reo circuit, and as a starting point for its formal verification.

In terms of future work, what we would like to do in the next step is to develop proper logics for specifying hybrid properties of Reo connectors. Development of suitable algorithms and model checking tools to verify hybrid properties of connectors in Reo based on the HCA model will also be studied.

Acknowledgement. The work was partially supported by the National Natural Science Foundation of China under grant no. 61202069 and 61272160, project IGDSi1305012 from SUTD, and Research Fund for the Doctoral Program of Higher Education of China under grant no. 20120001120103.

References

1. Alur, R., Courcoubetis, C., Henzinger, T.A., Ho, P.-H.: Hybrid Automata: An Algorithmic Approach to the Specification and Verification of Hybrid Systems. In: Grossman, R.L., Ravn, A.P., Rischel, H., Nerode, A. (eds.) HS 1991 and HS 1992. LNCS, vol. 736, pp. 209–229. Springer, Heidelberg (1993)
2. Arbab, F.: Reo: A Channel-based Coordination Model for Component Composition. Mathematical Structures in Computer Science 14(3), 329–366 (2004)
3. Arbab, F., Baier, C., de Boer, F., Rutten, J.: Models and Temporal Logics for Timed Component Connectors. In: Proceedings of SEFM2004, pp. 198–207. IEEE Computer Society (2004)
4. Arbab, F., Rutten, J.: A coinductive calculus of component connectors. In: Wirsing, M., Pattinson, D., Hennicker, R. (eds.) WADT 2003. LNCS, vol. 2755, pp. 34–55. Springer, Heidelberg (2003)
5. Baier, C., Sirjani, M., Arbab, F., Rutten, J.: Modeling component connectors in Reo by constraint automata. Science of Computer Programming 61, 75–113 (2006)
6. Henzinger, T.A.: The theory of hybrid automata. In: LICS, pp. 278–292. IEEE Computer Society (1996)
7. Kokash, N., Krause, C., de Vink, E.: Time and data aware analysis of graphical service models. In: Proceedings of SEFM 2010, pp. 125–134. IEEE Computer Society (2010)
8. Kokash, N., Krause, C., de Vink, E.: Reo+mCRL2: A framework for model-checking dataflow in service compositions. In: Formal Aspects of Computing, vol. 24, pp. 187–216.
9. Lee, E.A.: Computing Foundations and Practice for Cyber Physical Systems: A Preliminary Report. Technical Report UCB/EECS-2007-72, Department of Electrical Engineering and Computer Sciences, UC Berkeley (2007)
10. Lynch, N.A., Segala, R., Vaandrager, F.W.: Hybrid I/O Automata Revisited. In: Di Benedetto, M.D., Sangiovanni-Vincentelli, A.L. (eds.) HSCC 2001. LNCS, vol. 2034, pp. 403–417. Springer, Heidelberg (2001)
11. Lynch, N., Segala, R., Vaandrager, F., Weinberg, H.: Hybrid I/O Automata. In: Alur, R., Sontag, E.D., Henzinger, T.A. (eds.) HS 1995. LNCS, vol. 1066, pp. 496–510. Springer, Heidelberg (1996)
12. Meng, S.: Connectors as designs: The time dimension. In: Proceedings of TASE 2012, pp. 201–208. IEEE Computer Society (2012)
13. Peano, G.: Demonstration de l'intégrabilité des équations defférentielles ordinaires. Mathematische Annalen 37, 182–228 (1890)