Modeling and Specification of Real-time Interfaces with UTP

Dang Van Hung and Hoang Truong*

University of Engineering and Technology
Vietnam National University, Hanoi, Vietnam
{dvh,hoangta}@vnu.edu.vn

Abstract. Interface modeling and specification are central issues of component-based software engineering. How a component will be used is specified in its interface. Real-time interfaces are interfaces with timing constraints relating the time of outputs with the time of inputs. The timing constraint of an interface may depend on the resource availability for the component. In this paper, we propose a general model for real-time interfaces. At a time during execution, an interface behaves according to a contract made with environment about its functionality as well as execution time to fulfill the contract. This contract is specified as a timed design using the UTP notations, and depends on the computation histories of the interface. We model this dependence as a partial function from computation histories of the interface to real-time contracts. How interfaces are composed to form new interfaces, how interfaces are refined, and how to represent interfaces finitely are also considered in this paper. We show that checking the consistency between an environment and an interface and checking the refinement between two interfaces when they are represented by an automaton can be done effectively.

1 Introduction

Component-based design has been an efficient divide-and-conquer technique for the development of complex real-time embedded systems. A key role for this technique is interface modeling and specification. There has been a lot of significant progress towards a comprehensive theory for interfaces, see for example [1–6]. In those works different aspects of interfaces have been modeled and specified: interaction protocols, contracts, concurrency, timing, input-output relations, synchrony and asynchrony, etc. However, none among them can integrate all those aspects which help to analyze the relation between different aspects of the interfaces and find out conditions on which different aspects can be separated to simplify the analysis and verification. For untimed component interfaces, a recent work by Tripakis [7] has proposed a nice and comprehensive theory.

In this work, we extend that theory for real-time interfaces with the use of UTP developed by He and Hoare [8]. In this framework, we consider a component

* This work was supported by the research Project QG.11.29, Vietnam National University, Hanoi

as to provide a set of services, each service is specified by a "timed design". A timed design consists of a precondition on the input variables, and a relation between input variables and output variables together with the time constraint for the availability of the output. This time constraint is of the form $c \leq \ell \leq d$ where c is the best case execution time and d is the worst case execution time to fulfill the computation, and ℓ is a special temporal variable representing excution time. Which service is offered at a time depends on the current state of the component. The current state of the component expresses much more information than a protocol. From the current state, we can see not only how the environment interacted with the component so far, but also how the component has evolved. A state is a timed sequence of rounds, each round consists of an input value together with the time the input is given, and output value responding to that input value together with the actual execution time. So, the set of all states of a component is infinite. We then consider how an interface interacts with its environment, and derive the set of all reachable real-time behaviors for the system which can help to estimate the worst case execution time and to analyze the schedulability for multithreaded environments. We use the plugability as the condition for interface refinement and find that this condition can also be verified syntactically. In practice, an interface can only provide a finite number of services, and since a service is related with a subset of states, we can introduce an equivalence relation to partition the set of states into a finite number of subsets and use a labeled automaton to represent an interface. With that kind of representation, we can easily perform some operations on interfaces like interface compositions and refinement, and the verification of some properties. Note that when timing constraints are ignored, and specified as the interval $[0, 0]$ our model becomes an untimed synchronous interface one.

The paper is organized as follows. The next section presents our general real-time interface model. Section 3 considers some operations on interfaces. The representation of interfaces as automata and checking the plugability is presented in Sec 4. Related work and discussion is presented in Section 5. The last section is the conclusion of our paper.

2 Timed Relational Interface

A static service provided by a real-time component C can be specified as a contract on a signature (X, Y), where X and Y respectively are disjoint sets of input and output variables of the component. The contract can be represented by a so-called "timed design" which was introduced by us in [3] as a timed extension of a design developed by He and Hoare [8]. Let ℓ be a specific temporal variable for execution time, $\ell \notin X \cup Y$.

Definition 1. *(Timed design)*

1. *A timed design on signature (X, Y) is of the form:*

$$\rho = p \vdash (R, c \leq \ell \leq d)$$

where p is a predicate on variables in X called precondition of ρ, R is a relation from X to Y called post-condition of ρ, and c, d are non-negative real numbers satisfying $c \leq d$. Let us denote R as ρ_f and $c \leq \ell \leq d$ as ρ_t for timed design ρ. We call ρ_t the time constraint of ρ[1]*. Timed design ρ simply says that if it is called with the value of inputs satisfying p then it will terminate and give values for output variables after at least c time units and at most d time units, and R is satisfied at the termination.*

2. *For timed designs $\rho = p \vdash (R, c \leq \ell \leq d)$ and $\rho' = p' \vdash (R', c' \leq \ell \leq d')$, ρ is said to be a refinement of ρ', denoted as $\rho \sqsubseteq \rho'$ iff $p' \Rightarrow p$, $R \Rightarrow R'$ and $[c, d] \subseteq [c', d']$. When $\rho \sqsubseteq \rho'$ and $\rho' \sqsubseteq \rho$ we say ρ and ρ' are equivalent.*

Let $\mathbb{R}^+$ denote the set of non-negative real numbers. A computation round (or simply a round) is a pair $(\mathcal{V}, I)$, where $\mathcal{V}$ is a value assignment for variables in $X \cup Y$, and I is a time interval $[b, e]$, $b, e \in \mathbb{R}^+$, $b \leq e$. A round $r = (\mathcal{V}, I)$ is said to satisfy a timed design $\rho = p \vdash (R, c \leq \ell \leq d)$, denoted as $(\mathcal{V}, I) \models \rho$, iff $\mathcal{V}|_X \models p$, $\mathcal{V} \models R$ and $c \leq \mathit{length}(I) \leq d$, where $\mathcal{V}|_X$ is the restriction of $\mathcal{V}$ on the set of variables X, and $\mathit{length}(I)$ is the length of interval I. So, when $p \equiv \mathit{false}$, no round can satisfy ρ. In this definition of satisfiability, only the length of the interval of a round (not the interval itself) and the value assignment play role. Therefore, we says that rounds $(\mathcal{V}, I)$ and $(\mathcal{V}, I')$ are equivalent if $\mathit{length}(I) = \mathit{length}(I')$. For any equivalent rounds r, r' and a timed design ρ it holds that $r \models \rho$ if and only if $r' \models \rho$. A round $r = (\mathcal{V}, [b, e])$ is said to be before (right before) a round $r' = (\mathcal{V}', [b', e'])$, or equivalently, r' is after r, if $e \leq b'$ (resp. $e = b'$). A sequence of rounds $r_1 r_2 \ldots r_n$ such that r_{i+1} is after r_i for all $1 \leq i < n$ is called a state. A sequence of consecutive rounds is a state $r_1 r_2 \ldots r_n$ such that r_{i+1} is right after r_i for all $1 \leq i < n$.

Let $\mathcal{S}(X, Y)$ denote the set of all states, $\mathcal{D}(X, Y)$ denote the set of all timed designs over signature (X, Y). We give the following definition to real-time interfaces.

Definition 2. *(Real-time Interface) A real-time interface is a tuple $\mathcal{I} = (X, Y, \xi)$, where (X, Y) is a signature, ξ is a partial function from $\mathcal{S}(X, Y)$ to $\mathcal{D}(X, Y)$ satisfying:*

- *$\xi(\epsilon) = \rho_0$ is defined, where ϵ is the empty sequence.*
- *If $\xi(r_1 r_2 \ldots r_n)$ is defined, then $\xi(r_1 r_2 \ldots r_{n-1}) = \rho_{n-1}$ is also defined, and $r_n \models \rho_{n-1}$.*

When $\xi(r_1 r_2 \ldots r_n)$ is defined, we call $r_1 r_2 \ldots r_n$ a reachable state of (X, Y, ξ).

So, the set of all reachable states of a real-time interface $\mathcal{I} = (X, Y, \xi)$ forms a prefix-closed subset of $\mathcal{S}(X, Y)$. A reachable state s for which there is no round r such that sr is reachable is called a deadlock state of $\mathcal{I}$. Deadlock states are those reachable states that cannot be expanded. There are two reasons that a given state s is not expandable: either the function ξ is undefined for any extension of

[1] ρ could be written as a formula $p \vdash \rho_f \wedge \rho_t$. However, for the purpose of this paper it is more convenient to write ρ as $p \vdash (\rho_f, \rho_t)$

the state s, or the timed design $\xi(s)$ is not satisfiable. In the sequence, we will assume that for any state s, if $\xi(s)$ is defined then it is a satisfiable timed design. Hence, deadlock states are those reachable states for which the function ξ is not defined for any extension of them.

Example 1. Consider an interface that takes x as its unique input, and outputs y as the average of the values of x that it has received so far. Only the values of x that are in between two thresholds *low* and *high* are permitted to calculate the average. The interface is specified as $(\{x\}, \{y\}, \xi)$, where $\xi((\mathcal{V}_1, I_1) \ldots (\mathcal{V}_n, I_n)) = ((x \leq high \wedge low \leq x) \vdash ((\sum_{j=1}^{n} \mathcal{V}_j(x) + x)/(n+1) = y, 0 \leq \ell \leq 1))$ for any state $(\mathcal{V}_1, I_1) \ldots (\mathcal{V}_n, I_n)$ such that $\mathcal{V}_i(x) \in [low, high]$, $\mathcal{V}_i(y) = \sum_{j=1}^{i} \mathcal{V}_j(x)$ and $length(I_i) \leq 1$.

There are two interesting properties for this interface: it does not have deadlock state, and the range of the function ξ is infinite.

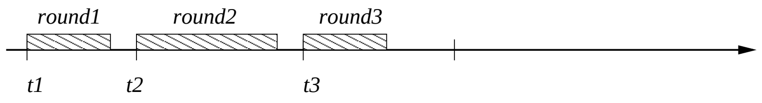

Fig. 1. A state as a timed execution of an interface

Let $\mathcal{R}(\mathcal{I})$ denote the set of reachable states of interface $\mathcal{I}$. In a reachable state, the time interval of a round represents an execution of a service provided by the component of the interface. When the environment invokes a service, it starts a round. So, it is the environment who decides the time to start a round, and the component decides when a round ends. Therefore, it is more practical, if for two states $s = r_1 \ldots r_k$ and $s' = r'_1 \ldots r'_k$ for which r_i and r'_i are equivalent for all $1 \leq i \leq k$, we have $\xi(s) = \xi(s')$. This gives right to the following definition. We say two states $s = r_1 \ldots r_k$ and $s' = r'_1 \ldots r'_n$ are equivalent iff $k = n$ and for all $1 \leq i \leq k$ the rounds r_i and r'_i are equivalent.

Definition 3. *(Timeless Interface) A real-time interface (X, Y, ξ) is timeless (or input-time independent) iff $\xi(s) = \xi(s')$ for any reachable equivalent states s, s'. (X, Y, ξ) is completely timeless (or input/output-time independent) iff $\xi(s) = \xi(s')$ for any reachable states $s = (\mathcal{V}_1, I_1) \ldots (\mathcal{V}_n, I_n)$ and $s' = (\mathcal{V}_1, I'_1) \ldots (\mathcal{V}_n, I'_n)$*

So, the real-time services provided by an interface do not depend on the time of inputs if the interface is timeless, and do not depend on the time of inputs and outputs if the interface is completely timeless. From now on in this paper, we consider only timeless interfaces, and the reachable states with consecutive rounds will be selected to be the representatives of equivalence classes. Those representatives are states that represent the busiest behavior of components, and are used to analyze the real-time capacity of interfaces like for which arrival rate and which deadline can be met for the worst case.

The range of ξ in an interface $\mathcal{I}$ could be infinite like in the example given above, and this makes it difficult to represent interfaces. As we have mentioned in the introduction, in practice, a component can only provide a finite number of services. The interface of this kind of components is called finite.

Definition 4. *A real-time interface* (X, Y, ξ) *is*

- *finite iff* $range(\xi)$ *is finite,*
- *stateless iff* $range(\xi)$ *has only one element, i.e.* ξ *is a constant mapping.*

Now we give a definition of interface environments and study how an interface will be used by an environment. Let us define a timed behavior to be a sequence $(\mathcal{V}_1, t_1)(\mathcal{V}_2, t_2)\ldots(\mathcal{V}_m, t_m)$, where $\mathcal{V}_i$ is a value assignment for variables in $X \cup Y$, $t_i \in \mathbb{R}^+$, $t_i \leq t_{i+1}$, $i = 1, \ldots, m-1$, $t_1 = 0$. Let $\mathcal{W}(X, Y)$ be the set of all timed behaviors over (X, Y). For a set of variables V, let $\mathcal{F}(V)$ denote the set of all predicates with free variables in V.

Definition 5. *(Environment) An environment over the signature* (X, Y) *is a tuple* $E = (X, Y, h)$*, where* h *is a partial function from* $\mathcal{W}(X, Y)$ *to* $\mathcal{D}(X, Y)$*, respectively, satisfying:*

- $h(\epsilon) = f_0 \vdash (g_0, l_0 \leq \ell \leq u_0)$ *is always defined.*
- *If* $h(w_1 w_2 \ldots w_n) = f_n \vdash (g_n, l_n \leq \ell \leq u_n)$ *is defined then*

$$h(w_1 w_2 \ldots w_{n-1}) = f_{n-1} \vdash (g_{n-1}, l_{n-1} \leq \ell \leq u_{n-1})$$

is also defined for which $\mathcal{V}_n \models f_{n-1} \wedge g_{n-1}$ *and* $t_n - t_{n-1} \geq u_{n-1}$ *hold, where* $w_i = (\mathcal{V}_i, t_i)$*,* $i \leq n$*.*

When $h(w_1 w_2 \ldots w_n)$ *is defined, we call* $w_1 w_2 \ldots w_n$ *a reachable timed behavior of environment* E*. Let* $\mathcal{B}(E)$ *denote the set of all reachable timed behaviors of environment* E*.*

As for the function ξ *of interfaces, we assume also that* $h(w)$ *is satisfiable whenever it is defined.*

Note that an environment $E = (X, Y, h)$ is not an interface since h is a partial function on $\mathcal{W}(X, Y)$, while in an interface $\mathcal{I} = (X, Y, \xi)$, the partial function ξ is on $\mathcal{S}(X, Y)$. Intuitively, a reachable timed behavior $w_1 w_2 \ldots w_n$ of E represents an interaction between E and an interface. For any $i = 1, \ldots, n$, at time t_i, E issues an input X that satisfies f_{i-1} to the interface, and expects an output Y at some time in the time interval $[t_i + l_{i-1}, t_i + u_{i-1}]$ from the interface, and that output is related with the issued input by relation g_{i-1}. This is described by $\mathcal{V}_i \models f_{i-1} \wedge g_{i-1}$. The time t_{i+1} at which it issues the next input will be after $t_i + u_{i-1}$.

In order for an interface $\mathcal{I}$ to be plugable to the environment E, the contracts specified by $\mathcal{I}$ should fit the requirement of E at any time during the interaction. This means that t_{i+1} is the time to start a computation round $r_i = (\mathcal{V}_i, I_i)$ for $\mathcal{I}$ with a service specified by a timed design $\rho_i = p_i \vdash (R_i, c_i \leq \ell \leq d_i)$ such that $f_i \Rightarrow p_i$ (i.e. the input received by $\mathcal{I}$ satisfies the precondition of the

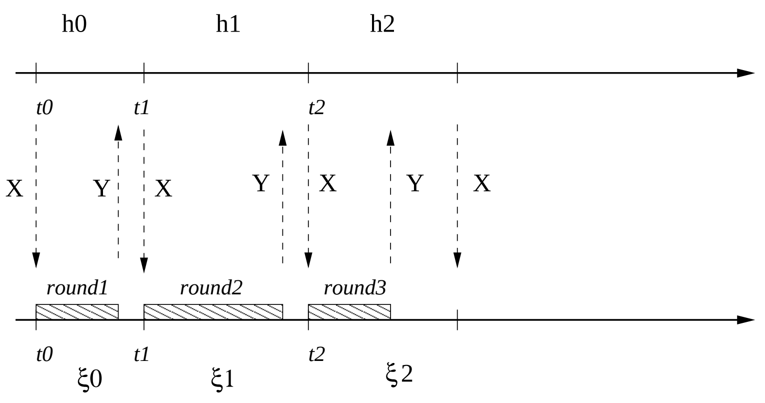

Fig. 2. Environment and interface interaction, contract ξi meets requirement hi at ti

service), and $f_i \wedge R_i \Rightarrow g_i$ (the result from the service is expected by E), and $[t_{i+1}+c_i, t_{i+1}+d_i] \subseteq [t_{i+1}+l_i, t_{i+1}+u_i]$ (the computation time meets the time constraint of E).

Definition 6. *(Plugability) An interface $\mathcal{I} = (X', Y', \xi)$ is plugable to environment $E = (X, Y, h)$, denoted by $E \# \mathcal{I}$ iff $X' = X$, $Y' = Y$ and the following conditions are satisfied.*

1. *Let $h(\epsilon) = f_0 \vdash (g_0, l_0 \le \ell \le u_0)$, and $\xi(\epsilon) = \rho_0 = p_0 \vdash (R_0, c_0 \le \ell \le d_0)$. Then, $f_0 \Rightarrow p_0$, $f_0 \wedge R_0 \Rightarrow g_0$, and $t_0 = 0$, $[t_0+c_0, t_0+d_0] \subseteq [t_0+l_0, t_0+u_0]$. For any $\mathcal{V}_1$ such that $\mathcal{V}_1 \models f_0 \wedge R_0$ and interval $I_1 = [t_0, t_0']$ with $c_0 \le length(I_1) \le d_0$, for any $t_1 \ge t_0 + u_0$ the pair $(\mathcal{V}_1, t_1)$ is called reachable timed behavior of E w.r.t. $\mathcal{I}$, and $(\mathcal{V}_1, I_1)$ is called reachable state of $\mathcal{I}$ w.r.t. $(\mathcal{V}_1, t_1)$.*
2. *Let $w_n = (\mathcal{V}_1, t_1) \dots (\mathcal{V}_n, t_n)$ be reachable timed behavior of E w.r.t. $\mathcal{I}$ such that $h(w_n)$ is defined, and $s_n = (\mathcal{V}_1, I_1) \dots (\mathcal{V}_n, I_n)$ be reachable state of $\mathcal{I}$ w.r.t. w_n. Then, $\xi(s_n)$ is also defined. Furthermore, let $h(w_n) = f_n \vdash (g_n, l_n \le \ell \le u_n)$, and $\xi(s_n) = \rho_n = p_n \vdash (R_n, c_n \le \ell \le d_n)$. Then, $f_n \Rightarrow p_n$, $f_n \wedge R_n \Rightarrow g_n$, and $[t_n + c_n, t_n + d_n] \subseteq [t_n + l_{n-1}, t_n + u_{n-1}]$. For any $\mathcal{V}_{n+1}$ such that $\mathcal{V}_{n+1} \models f_n \wedge R_n$, for any interval $I_n = [t_n, t_n']$ with $c_n \le length(I_n) \le d_n$, and for any t_{n+1} such that $t_n + u_n \le t_{n+1}$, $(\mathcal{V}_1, t_1) \dots (\mathcal{V}_n, t_n)(\mathcal{V}_{n+1}, t_{n+1})$ is called reachable timed behavior of E w.r.t. $\mathcal{I}$, and $(\mathcal{V}_1, I_1) \dots (\mathcal{V}_n, I_n)(\mathcal{V}_{n+1}, I_{n+1})$ is called reachable state of $\mathcal{I}$ w.r.t. $(\mathcal{V}_1, t_1) \dots (\mathcal{V}_n, t_n)(\mathcal{V}_{n+1}, t_{n+1})$.*

Note that in this definition, for a pair $(\mathcal{V}_i, t_i)$ in a reachable behavior w.r.t. $\mathcal{I}$, t_{i-1} is the starting time of a round for which both input and output are given by value assignment $\mathcal{V}_i$. A necessary condition for the plugability is that for any reachable timed behavior w of E w.r.t. $\mathcal{I}$, and reachable state s of $\mathcal{I}$ w.r.t. w, if w is expandable, then s must not be a deadlock state of $\mathcal{I}$.

Given $E\#\mathcal{I}$, let $\mathcal{B}(E\#I)$ be the set of all reachable timed behaviors of E w.r.t. $\mathcal{I}$, and $\mathcal{S}(E\#I)$ be the set of all reachable states of $\mathcal{I}$ w.r.t. E. It is then $\mathcal{S}(E\#I) \subseteq \mathcal{R}(\mathcal{I})$ and $\mathcal{B}(E\#I) \subseteq \mathcal{B}(E)$. So, not all services from $\mathcal{I}$ are used by E, and the outputs from $\mathcal{I}$ may restrict the behaviors of E.

Lemma 1. *Let $\mathcal{I}$ be a real-time interface. There exists an environment $E_{\mathcal{I}}$ such that $E_{\mathcal{I}}\#\mathcal{I}$ and $\mathcal{S}(E_{\mathcal{I}}\#\mathcal{I}) = \mathcal{R}(\mathcal{I})$.*

Proof. Let $\mathcal{I} = (X, Y, \xi)$. For a reachable state $s_n \in \mathcal{R}(\mathcal{I})$ and $s_n = (\mathcal{V}_1, I_1)(\mathcal{V}_2, I_2)\dots(\mathcal{V}_n, I_n)$, we call the sequence $(\mathcal{V}_1, t_1)(\mathcal{V}_2, t_2)\dots(\mathcal{V}_n, t_n)$ a possible timed behavior w.r.t. s_n, where $t_0 = 0$, and for $i \geq 0$, t_{i+1} satisfy the following condition: let $\xi(\mathcal{V}_1, I_1)(\mathcal{V}_2, I_2)\dots(\mathcal{V}_{i-1}, I_{i-1}) = p_{i-1} \vdash (R_{i-1}, c_{i-1} \leq \ell \leq d_{i-1})$, then, t_{i+1} is any time point that satisfies $t_{i+1} \geq t_i + d_{i-1}$, where $\mathcal{V}_1, I_1)(\mathcal{V}_2, I_2)\dots(\mathcal{V}_{i-1}, I_{i-1}) = \epsilon$ when $i = 1$. The partial function h is defined as: if w is a possible behavior w.r.t. s, and $\xi(s) = \rho = p \vdash (R, c \leq \ell \leq d)$ then $h(w) = p \vdash (R, c \leq \ell \leq d)$. Let $E_{\mathcal{I}} = (X, Y, h)$. It is trivial to verify that $E_{\mathcal{I}}\#\mathcal{I}$ and $\mathcal{S}(E_{\mathcal{I}}\#I) = \mathcal{R}(\mathcal{I})$.

It is the plugability that helps to give a more natural definition for the refinement of interfaces. An interface $\mathcal{I}$ is said to be "better" than an interface $\mathcal{I}'$ iff $\mathcal{I}$ can replace $\mathcal{I}'$ in all cases of use of $\mathcal{I}'$.

Definition 7. *(Interface Refinement) An interface $\mathcal{I}$ is said to be a refinement of an interface $\mathcal{I}'$, denoted by $\mathcal{I} \sqsubseteq \mathcal{I}'$, iff for all environment E, if $\mathcal{I}'$ is plugable to E then $\mathcal{I}$ is also plugable to E.*

It turns out that this natural definition is equivalent to the classical one.

Theorem 1. *Let $\mathcal{I} = (X, Y, \xi)$ and $\mathcal{I}' = (X', Y', \xi')$ be real time interfaces. $\mathcal{I} \sqsubseteq \mathcal{I}'$ holds if and only if for all $s \in \mathcal{R}(\mathcal{I}') \cap \mathcal{R}(\mathcal{I})$ either $\xi(s) \sqsubseteq \xi'(s)$ holds or s is a deadlock state of $\mathcal{I}'$.*

Proof. The "only if" part of the theorem follows directly from Lemma 1. Since $E_{\mathcal{I}'}$ is also plugable to $\mathcal{I}$, if $\xi'(s)$ is defined, $\xi(s)$ must be defined and $\xi(s) \sqsubseteq \xi'(s)$ according to the definition of plugability.

The "if part" of the theorem is proved by induction on the length of common reachable states based on the definition of plugability. If $E = (X, Y, h)$ is plugable to $\mathcal{I}'$, the first item of Definition 6 is verified for $\mathcal{I}'$. Let $h(\epsilon) = f_0 \vdash (g_0, l_0 \leq \ell \leq u_0)$, and $\xi'(\epsilon) = \rho'_0 = p'_0 \vdash (R'_0, c'_0 \leq \ell \leq d'_0)$. Then, $f_0 \Rightarrow p'_0$, $f_0 \wedge R'_0 \Rightarrow g_0$, and $t_0 = 0$, $[t_0 + c'_0, t_0 + d'_0] \subseteq [t_0 + l_0, t_0 + u_0]$. Let $\xi(\epsilon) = \rho_0 = p_0 \vdash (R_0, c_0 \leq \ell \leq d_0)$. From the assumption $\xi(\epsilon) \sqsubseteq \xi'(\epsilon)$, it follows that $f_0 \Rightarrow p_0$, $f_0 \wedge R_0 \Rightarrow g_0$, $[t_0 + c_0, t_0 + d_0] \subseteq [t_0 + l_0, t_0 + u_0$. For any $\mathcal{V}_1$ such that $\mathcal{V}_1 \models f_0 \wedge R_0$ and interval $I_1 = [t_0, t'_0]$ with $c_0 \leq \mathit{length}(I_1) \leq d_0$ and for any $t_1 \geq t_0 + u_0$, $(\mathcal{V}_1, t_1)$ is a reachable timed behavior of E w.r.t. $\mathcal{I}$ and $\mathcal{I}'$, and $(\mathcal{V}_1, I_1)$ is a reachable state of both interfaces $\mathcal{I}$ and $\mathcal{I}'$ w.r.t. $(\mathcal{V}_1, t_1)$.

The second item of Definition 6 for E and $\mathcal{I}$ is verified exactly in the same way with the induction hypothesis.

It follows immediately from this theorem that:

Corollary 1. *Let $\mathcal{I}$ and $\mathcal{I}'$ be real time interfaces and $\mathcal{I} \sqsubseteq \mathcal{I}'$. Then, for any environment E such that $E\#\mathcal{I}'$ we have $\mathcal{R}(E\#\mathcal{I}) \subseteq \mathcal{R}(E\#\mathcal{I}')$*

Two interfaces $\mathcal{I}$ and $\mathcal{I}'$ are equivalent, denoted as $\mathcal{I} \equiv \mathcal{I}'$ iff $\mathcal{I} \sqsubseteq \mathcal{I}'$ and $\mathcal{I} \sqsubseteq \mathcal{I}'$.

Corollary 2. *Let $\mathcal{I} = (X, Y, \xi)$ and $\mathcal{I}' = (X', Y', \xi')$ be real time interfaces.*

1. *$\mathcal{I} \equiv \mathcal{I}'$ if and only if for any environment E, $E\#\mathcal{I}$ iff $E\#\mathcal{I}'$.*
2. *$\mathcal{I} \equiv \mathcal{I}'$ if and only if for all $s \in \mathcal{R}(\mathcal{I}) \cap \mathcal{R}(\mathcal{I}')$, $\xi(s) \equiv \xi'(s)$.*

3 Interface Composition

The most important operations on interfaces are composition operations. We consider two kinds of composition: parallel and sequential. Two interfaces $\mathcal{I}$ and $\mathcal{I}'$ can be put together either in parallel or in sequence to achieve a new interface that provide compound services. However, the execution time for the compound services need to considered carefully. From our intention, the time constraint in a timed design serves for the estimation of the execution time. The outputs from a component are supposed to be synchronous, and given at the end of the computation round. Therefore, the computation result coming out earlier must wait for those that have not been available yet. As soon as all the outputs are available, the computation round terminates. This consideration leads to the following definition of parallel and sequential composition of interfaces. Let us denote for a value assignment $\mathcal{V}$ and a set of variables V the restriction of $\mathcal{V}$ on V by $\mathcal{V}|_V$.

Definition 8. *(Parallel Composition)*
Let $\mathcal{I} = (X, Y, \xi)$ and $\mathcal{I}' = (X', Y', \xi')$ be two completely timeless interfaces such that $(X \cup Y) \cap (X' \cup Y') = \emptyset$. The parallel composition $\mathcal{I}||\mathcal{I}'$ is the interface $(X \cup X', Y \cup Y', \xi'')$ where $\xi'' : \mathcal{S}(X \cup X', Y \cup Y') \to \mathcal{D}(X \cup X', Y \cup Y')$ defined as follows. For $s = (\mathcal{V}_1, I''_1) \ldots (\mathcal{V}_n, I''_n) \in \mathcal{S}(X \cup X', Y \cup Y')$, $\xi''(s)$ is defined iff it is defined for all proper prefixes of s, and there exist time intervals $I_1, I'_1, \ldots I_n, I'_n$ such that both

$$\xi((\mathcal{V}_1|_{X \cup Y}, I_1) \ldots (\mathcal{V}_n|_{X \cup Y}, I_n)) = p \vdash (R, c \leq \ell \leq d), \textit{ and}$$
$$\xi'((\mathcal{V}_1|_{X' \cup Y'}, I'_1) \ldots (\mathcal{V}_n|_{X' \cup Y'}, I'_n)) = p' \vdash (R', c' \leq \ell \leq d')$$

are defined, and then

$$\xi''(s) = p \wedge p' \vdash (R \wedge R', \min\{c, c'\} \leq \ell \leq \max\{d, d'\})$$

and I''_n satisfies that $\min\{c, c'\} \leq length(I''_n) \leq \max\{d, d'\}$ and I''_n is after I''_{n-1}.

Note that the definition is well-formed (i.e. ξ'' is well defined) because $\mathcal{I}$ and $\mathcal{I}'$ are completely timeless. It also follows that $\mathcal{I}||\mathcal{I}'$ is also completely timeless.

By sequential composition of two interfaces, we mean that some inputs of the second are connected to some outputs of the first interface. To be defined, an input can be connected to at most one output although several inputs may be connected to the same output. Given two interfaces $\mathcal{I} = (X, Y, \xi)$ and $\mathcal{I}' = (X', Y', \xi')$ such that $(X \cup Y) \cap (X' \cup Y') = \emptyset$. A connection from $\mathcal{I}$ to $\mathcal{I}'$ is a set of pairs $\theta \subseteq Y \times X'$ that satisfy $\forall (y, x), (y', x') \in \theta.(x = x' \Rightarrow y = y')$. Let $X_\theta = \{x \in X' | \exists y \in Y.(y, x) \in \theta\}$. A connection θ converses an assignment a over $((X \cup X') \setminus X_\theta) \cup Y \cup Y'$ to an assignment a^θ over $(X \cup X' \cup Y \cup Y')$ as: $a^\theta|_{((X \cup X') \setminus X_\theta) \cup Y \cup Y'} = a|_{((X \cup X') \setminus X_\theta) \cup Y \cup Y'}$, and for $x \in X_\theta$ we define $a^\theta(x) = a(y)$ where y is the unique element in Y such that $(y, x) \in \theta$. Let us denote $\nu_\theta = \bigwedge_{(y,x) \in \theta} x = y$.

Definition 9. *(Sequential Composition)*
Let $\mathcal{I} = (X, Y, \xi)$ and $\mathcal{I}' = (X', Y', \xi')$ be two completely timeless interfaces. Sequential composition of $\mathcal{I}$ and $\mathcal{I}'$ w.r.t connection θ, denoted by $\mathcal{I}.{}_\theta\mathcal{I}'$ is interface $\mathcal{I}'' = (X'', Y'', \xi'')$, where

- *$X'' = (X \cup X') \setminus X_\theta$, $Y'' = Y \cup Y'$.*
- *For $s = (\mathcal{V}_1, I''_1) \dots (\mathcal{V}_n, I''_n) \in \mathcal{S}(X'', Y'')$, $\xi''(s)$ is defined iff it is defined for all proper prefix of s, and there exist time intervals $I_1, I'_1, \dots I_n, I'_n$ such that both*
$$\xi((a^\theta_1|_{X \cup Y}, I_1) \dots (a^\theta_n|_{X \cup Y}, I_n)) = p \vdash (R, c \leq \ell \leq d), \text{ and}$$
$$\xi'((a^\theta_1|_{X' \cup Y'}, I'_1) \dots (a^\theta_n|_{X' \cup Y'}, I'_n)) = p' \vdash (R', c' \leq \ell \leq d')$$
are defined, and then
$$\xi''(s) = p \wedge \exists Y.(R \wedge p' \wedge \nu_\theta) \vdash (R \wedge R' \wedge \nu_\theta \wedge p', c + c' \leq \ell \leq d + d')$$
and I''_n satisfies that $c + c' \leq length(I''_n) \leq d + d'$ and I''_n is after I''_{n-1}.

In this definition, we assume that the second interface takes the outputs from the first as soon as they are available, and for those outputs that are not in use by the second as inputs the availability is delayed to be the same as the availability of the outputs from the second. Also, only the outputs from the first that satisfy the precondition for the second are produced as outputs. As for parallel composition, $\mathcal{I}.{}_\theta\mathcal{I}'$ is also a completely timeless interface.

Parallel composition and sequential composition of two interfaces $\mathcal{I}$ and $\mathcal{I}'$ are depicted in Fig. 3.

The following theorem follows immediately from Definitions 8 and 9.

Theorem 2. *Let $\mathcal{I}$, $\mathcal{I}'$ and $\mathcal{I}''$ be interfaces, θ and θ' be connections between $\mathcal{I}$ and $\mathcal{I}'$ and between $\mathcal{I}'$ and $\mathcal{I}''$ respectively. Then, the following hold: (1) $\mathcal{I}||\mathcal{I}' \equiv \mathcal{I}'||\mathcal{I}$, (2) $(\mathcal{I}||\mathcal{I}')||\mathcal{I}'' \equiv \mathcal{I}||(\mathcal{I}'||\mathcal{I}'')$, and (3) $(\mathcal{I}.{}_\theta\mathcal{I}').{}_{\theta'}\mathcal{I}'' \equiv \mathcal{I}.{}_\theta(\mathcal{I}'.{}_{\theta'}\mathcal{I}'')$*

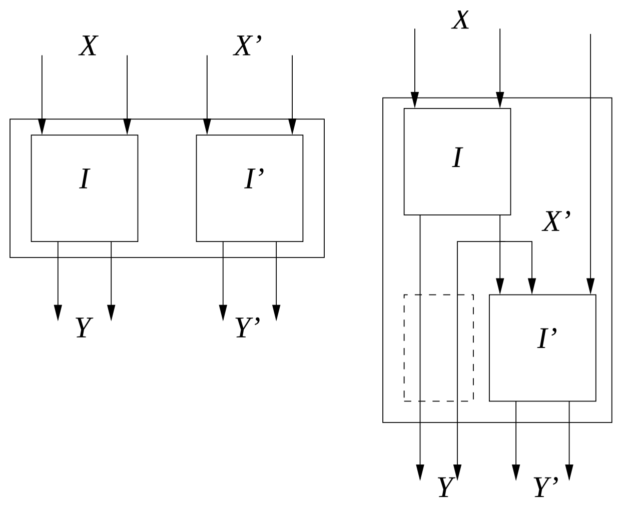

Fig. 3. Parallel and sequential composition

4 Automata Interfaces

There are two issues for general real-time interfaces: how to represent interfaces finitely, and how to check if an interface is plugable to an environment.

In general, a real-time interface $\mathcal{I} = (X, Y, \xi)$ cannot be finitely representable since ξ is a function from an infinite set to an infinite set. When $\mathcal{I} = (X, Y, \xi)$ is finite, the range of ξ is finite. Therefore the coimage of ξ forms a finite partition $\pi = \{S_1, \ldots, S_k\}$ of $\mathbf{dom}(\xi)$. If we can decide if a state is a member of S_i then ξ is finitely representable. For the simplicity of presentation and in order to be practical, we will consider in this section only finite and completely timeless interfaces. With this restriction, automata seem to be among the best representations for finite interfaces.

Definition 10. *(Labeled Automata)*
A labeled automaton M is a tuple $M = (Q, X, Y, q_0, T, l_s, l_t)$, where Q is a finite set of locations, X and Y are sets of input and output variables respectively, $q_0 \in Q$ is an initial state of M, $T \subseteq Q \times Q$ is a set of transitions, and $l_s : Q \to \mathcal{D}(X, Y)$ and $l_t : T \to \mathcal{F}(X \cup Y)$ are labeling functions. l_s associates each location in M with a timed design, and l_t associates each transition in T with a guard formula. To make M deterministic, we assume that $l_t(q, q'') \wedge l_t(q, q') \Rightarrow false$ for any two different transitions (q, q') and (q, q'') with the same source state.

Let $V(X \cup Y)$ be the set of all valuations over the set of variables $X \cup Y$. A labeled automaton M can describe a partial function $g : V^*(X \cup Y) \to \mathcal{D}(X, Y)$ in the following way:

1. $g(\epsilon) = l_s(q_0)$, and ϵ is said to lead M to q_0.

2. For any sequence of valuation $s \in V^*(X \cup Y)$, if $g(s) = p \vdash (R, c \leq \ell \leq d)$, and s leads M to location q, for any computation round $(\mathcal{V}, I')$ and transition (q, q') such that $(\mathcal{V}, I') \models g(s)$ and $\mathcal{V} \models l_t(q, q')$, we have $g(s\mathcal{V}) = l_s(q')$ and $s\mathcal{V}$ is said to lead M to location q'.

The function described by a labeled automaton M is denoted by g_M. It is the definition of g_M that makes the $\mathbf{dom}(g_M)$ prefix closed.

Definition 11. *(Automata Interface)*

1. *Interface $\mathcal{I} = (X, Y, \xi)$ is said to be an automata interface iff there is a labeled automaton M such that for any state $(\mathcal{V}_1, I_1) \ldots (\mathcal{V}_k, I_k)$ ($k \geq 0$, and the case $k = 0$ corresponds to the state ϵ), the ξ function value $\xi((\mathcal{V}_1, I_1) \ldots (\mathcal{V}_k, I_k))$ is defined exactly when $g_M(\mathcal{V}_i \ldots \mathcal{V}_k)$ is defined and $\xi((\mathcal{V}_1, I_1) \ldots (\mathcal{V}_k, I_k)) = g_M(\mathcal{V}_1 \ldots \mathcal{V}_k)$ provided that $c_i \leq length(I_i) \leq d_i$, where $g_M(\mathcal{V}_1, \ldots \mathcal{V}_i) = p_i \vdash (R_i, c_i \leq \ell \leq d_i)$, $i = 1, \ldots, n$. Such an automaton M is said to be a description of $\mathcal{I}$.*
2. *Environment $E = (X, Y, h)$ is said to be an automaton environment iff there is a label automaton M such that for any timed behavior $(\mathcal{V}_1, t_1) \ldots (\mathcal{V}_k, t_k)$, the h function value $h((\mathcal{V}_1, t_1) \ldots (\mathcal{V}_k, t_k))$ is defined exactly when $g_M(\mathcal{V}_i \ldots \mathcal{V}_k)$ is defined and $h((\mathcal{V}_1, t_1) \ldots (\mathcal{V}_k, t_k)) = g_M(\mathcal{V}_i \ldots \mathcal{V}_k)$ provided that $t_{i+1} \geq t_i + d_i$, $t_1 = 0$, where $g_M(\mathcal{V}_1, \ldots \mathcal{V}_i) = p_i \vdash (R_i, c_i \leq \ell \leq d_i)$, $i = 1, \ldots, n$. Such an automaton M is said to be a description of E.*

Example 2. Let us consider a GPS as a real-time interface $\mathcal{I}$, and its user as an environment E. The interface has the input variable set $\{usage, destination\}$, and the output variable set $\{display\}$. The automata representation of $\mathcal{I}$ and E are depicted in Fig 4. The label of states of the interface $\mathcal{I}$ automaton are: $l_s(s1) = true \vdash (display = idle, 0 \leq \ell \leq 1)$, $l_s(s2) = true \vdash (display = current\_position, 0 \leq \ell \leq 1)$, and $l_s(s3) = legal(destination) \vdash (display = route\_to\_destination, 1 \leq \ell \leq 2)$. The label of states of the environment E automaton are: $l_s(r1) = (usage = 1) \vdash (display = idle, 0 \leq \ell \leq 1)$, $l_s(r2) = (usage = 1) \vdash (display = current\_position, 0 \leq \ell \leq 1)$, and $l_s(s3) = (legal(destination) \wedge usage = 0 \vdash (display = route\_to\_destination, 0 \leq \ell \leq 3)$. The user gives the input "$usage = 1$" to the interface to turn on the system, then again gives the input "$usage = 1$" to the interface to get "*current_position*", and then gives the input "$usage = 1$" and a "*destination*" to the interface to get a "*route_to_destination*" to the destination, and finally an input "$usage = 0$" to turn off the system.

In the interface theory, it is important that the plugability of an interface to an environment is decidable. This is possible for automata interfaces and environments.

Theorem 3. *Let $\mathcal{I} = (X, Y, \xi)$ be an automaton interface described by automaton $M = (Q, X, Y, q_0, T, l_s, l_t)$ and $E = (X, Y, h)$ be an automata environment described by automaton $M' = (Q', X, Y, q_0', T', l_s', l_t')$. $E \# \mathcal{I}$ if and only if there is a correspondence f from Q' to Q (one may correspond to many) that satisfies:*

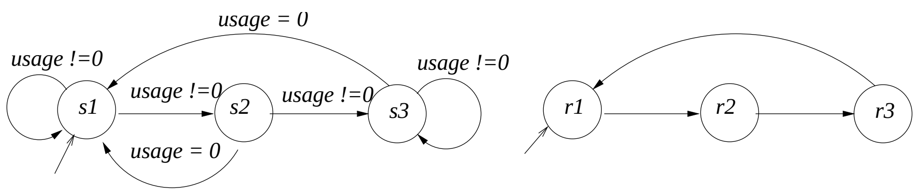

Fig. 4. An automata interface and an automata environment

1. *$f(q'_0) = \{q_0\}$*
2. *For any $q \in f(q')$, $l_s(q) \sqsubseteq l'_s(q')$*
3. *Let $q \in f(q')$ and $l_s(q) = p \vdash (R, c \leq \ell \leq d)$ and $l'_s(q) = p' \vdash (R', c' \leq \ell \leq d')$. For any $r' \in Q'$ such that $F_{(q',r')} \widehat{=} (p' \wedge R \wedge l'_t(q', r'))$ is satisfiable, let $\phi(q) = \{r \mid (q, r) \in T$ and $l_t(q, r) \wedge F_{(q',r')}$ is satisfiable$\}$. Then $F_{(q',r')} \Rightarrow \bigvee_{r\in\phi(q)} l_t(q, r)$ and $\phi(q) \subseteq f(r')$.*

Proof. The "if part" is proved by induction on reachable behavior of E and direct check of the definition of plugability. The "only if" part follows from an inductive construction of a correspondence f from the inductive definition of plugability between E and $\mathcal{I}$. We omit the proof details here.

In Example 2, the correspondence f defined as $f(r_i) = s_i$, $i = 1, 2, 3$. f satisfies the conditions in Theorem 3. Therefore, $\mathcal{I}\#E$.

As a special case of Theorem 3, let $Q' \subseteq Q$, and let $\pi \in X$ be a special distinct input variable, π does not occur in any timed design in the labels of automata M and M', and $\mathbf{dom}(\pi) = Q'$. Let $l_t(q', q) = l'_t(q', q) \widehat{=} (\pi = q)$, $l_s(q) = l'_s(q)$ for all $q', q \in Q'$, let f be the identifying mapping on Q'. Then, by Theorem 3, if the set of all transition sequences (paths) starting from initial state of M' is a subset of the set of all transition sequences starting from initial state of M, $E\#\mathcal{I}$ holds. In this case, the set of all transition sequences starting from initial state of M play the rôle of interaction protocols. If the behavior of E represented by M' "obeys" these protocols, E is plugable to $\mathcal{I}$.

It is easy to design an algorithm for checking the conditions of Theorem 3 to decide the plugability of a given automata interface to a given automata environment E.

Algorithm Checking plugability

Input: Automata interface $\mathcal{I}$ described by automaton
$M = (Q, X, Y, q_0, T, l_s, l_t)$, and an automata environment $E = (X, Y, h)$ described by automaton $M' = (Q', X, Y, q'_0, T', l'_s, l'_t)$.
Output: "Yes" if $E\#\mathcal{I}$, and "no" otherwise.
Method: Let $f \subseteq Q' \times Q$. f is initialized as $f = \{(q'_0, q_0)\}$ and (q'_0, q_0) is unmarked. Carry out following steps.
1. If no unmarked element is found in f, stop with the output "yes". Otherwise, take an unmarked element $(q', q) \in f$ and mark it.

2. If $l_s(q) \not\sqsubseteq l'_s(q')$, stop with the output "no". Otherwise, let $l_s(q) = p \vdash (R, c \leq \ell \leq d)$ and $l'_s(q') = p' \vdash (R', c' \leq \ell \leq d')$. For any $r' \in Q'$ such that $F_{(q',r')} \widehat{=} p' \wedge R \wedge l'_t(q', r')$ is satisfiable, let $\phi(q) \widehat{=} \{r \mid (q, r) \in T$ and $l_t(q, r) \wedge F_{(q',r')}$ is satisfiable$\}$. If $F_{(q',r')} \Rightarrow \bigvee_{r \in \phi(q)} l_t(q, r)$ is false, stop with the output "no". Otherwise, add unmarked pairs (r', r) to f for any $r \in \phi(q)$.
3. Goto Step 1.

In this algorithm we have to check the satisfiability of some logic formulas. When the domain of the variables in the formula are finite, satisfiability can be checked with SAT solvers.

As a corollary of Theorem 1, we have:

Theorem 4. *Let $\mathcal{I} = (X, Y, \xi)$ be an automata interface described by automaton $M = (Q, X, Y, q_0, T, l_s, l_t)$ and $\mathcal{I}' = (X, Y, \xi')$ be an automata interface described by automaton $M' = (Q', X, Y, q'_0, T', l'_s, l'_t)$. $\mathcal{I} \sqsubseteq \mathcal{I}'$ if and only if there is a correspondence f from Q' to Q (one may correspond to many) that satisfies:*

1. *$f(q'_0) = \{q_0\}$*
2. *For any $q \in f(q')$, $l_s(q) \sqsubseteq l'_s(q')$*
3. *Let $q \in f(q')$ and $l_s(q) = p \vdash (R, c \leq \ell \leq d)$ and $l'_s(q) = p' \vdash (R', c' \leq \ell \leq d')$. For any $r' \in Q'$ such that $F_{(q',r')} = p' \wedge R \wedge l'_t(q', r')$ is satisfiable, let $\phi(q) = \{r \mid (q, r) \in T$ and $l_t(q, r) \wedge F_{(q'r')}$ is satisfiable$\}$. Then $F_{(q',r')} \Rightarrow \bigvee_{r \in \phi(q)} l_t(q, r)$ and $\phi(q) \subseteq f(r')$.*

Proof. It follows from Theorem 1 that $\mathcal{I} \sqsubseteq \mathcal{I}'$ if and only if the environment $E_{\mathcal{I}'}$ is plugable to $\mathcal{I}$. Hence, Theorem 4 follows immediately Theorem 3.

5 Discussion and Related Work

As it is said in the introduction of this paper, the model presented in this paper is an extension of the model presented in [7] for real-time interfaces. The main difference is, here we use UTP timed designs to specify services, and services are defined only on reachable states. In addition to time extension, we found that using UTP is much more convenient than using other notations. We focus in this paper on checking the refinement and plugability and finite representations instead of providing a theory of real-time interfaces.

In the literature, there have been several studies on real-time interfaces such as [9–12]. In those papers, the authors have a focus on the analysis of schedulability from the task arrival rates. Our focus in this paper is different. We aim to provide a formal model that can help to do the shedulability analysis. Compared to our previous work [13, 3, 5], the model in this paper can capture the relation between interaction protocols and services. However, in this paper we ignored the relationship between resources and worst case execution times for the simplicity. This model can be easily extended to capture that relation. In our previous work [14], we checked at runtime if an environment follows the interaction protocol specified by a component. In this paper, we provide an algorithm to check at compile time the conformance between an environment and a component in term of both interaction protocols and their related services. Though the complexity of our checking algorithm is a bit high, we believe that it can be implemented with the help of advanced SAT solvers

6 Conclusion

We have presented a real-time interface model using UTP. The services of an interface are specified as a timed design in UTP, and depend on the current states of the interface. The model is general enough to capture all aspects of interface specification such as functionality, non-functionality, interaction protocol and the relation between them. We have also considered some operations on real-time interfaces. The operations are used in the incremental development of component-based embedded systems. We have also considered a finite representation of interfaces by labeled automata. We showed that the syntactical definition of interface refinement is consistent to the semantic one. We have shown that when interfaces and environments can be represented by automata, checking the plugability between interfaces and environments can be done effectively using SAT solvers. We believe that for finite interfaces that are not completely timeless, labeled time automata will be a good candidate for representing them, and similar results for checking the plugability can be obtained.

What we have left out in this paper is the analysis of the timed behaviors of environments when a component is plugged to, and to carry out verification of some real-time properties and the schedulability when there are several threads running in parallel in environments. This will be our future consideration.

References

1. Luca de Alfaro and Thomas A. Henzinger. Interface Automata. In *ACM Symposium on Foundation of Software Engineering (FSE)*, 2001.
2. Laurent Doyen, Thomas A. Henzinger, Barbara Jobstmann, and Tatjana Petrov. Interface theories with component reuse. In *EMSOFT*, pages 79–88, 2008.
3. Dang Van Hung. Toward a formal model for component interfaces for real-time systems. In *Proceedings of the 10th international workshop on Formal methods for industrial critical systems*, FMICS '05, pages 106–114, New York, NY, USA, 2005. ACM.
4. Jifeng He, Zhiming Liu, and Xiaoshan Li. rCOS: A refinement calculus of object systems. *Theor. Comput. Sci.*, 365(1-2):109–142, 2006. UNU-IIST TR 322.
5. Hung Ledang and Dang Van Hung. Timing and concurrency specification in component-based real-time embedded systems development. In *TASE*, pages 293–304. IEEE Computer Society, 2007.
6. Xin Chen, Jifeng He, Zhiming Liu, and Naijun Zhan. A model of component-based programming. In Farhad Arbab and Marjan Sirjani, editors, *FSEN*, volume 4767 of *Lecture Notes in Computer Science*, pages 191–206. Springer, 2007.

7. Stavros Tripakis, Ben Lickly, Thomas A. Henzinger, and Edward A. Lee. A theory of synchronous relational interfaces. *ACM Transactions on Programming Languages and Systems*, 33(4), 2011.
8. C.A.R. Hoare and He Jifeng. *Unifying Theories of Programming.* Prentice Hall Series in Computer Science. Prentice Hall, 1998.
9. Shengquan Wang, Sangig Rho, Zhibin Mai, Riccardo Bettati, and Wei Zhao. Real-time component-based systems. In *Proceedings of the 11th IEEE Real Time and Embedded Technology and Applications Symposium (RTAS05)*, pages 1080–1812. IEEE Computer Society, 2005.
10. Thomas A. Henzinger and Slobodan Matic. An interface algebra for real-time components. In *IEEE Real Time Technology and Applications Symposium*, pages 253–266, 2006.
11. Benot Delahaye, Uli Fahrenberg, Thomas A. Henzinger, Axel Legay, and Dejan Nickovic. Synchronous interface theories and time triggered scheduling. In *FMOODS/FORTE 2012*, pages 203–218, 2012.
12. Jimmie Wiklander, Jens Eliasson, Andrey Kruglyak, Per Lindgren, and Johan Nordlander. Enabling component-based design for embedded real-time software. *JOURNAL OF COMPUTERS*, 4(12):1039–1321, December 2009.
13. Dang Van Hung and Bui Vu Anh. Model checking real-time component based systems with blackbox testing. In *RTCSA*, pages 76–79, 2005.
14. Anh-Hoang Truong, Thanh-Binh Trinh, Dang Van Hung, Viet-Ha Nguyen, Nguyen Thi Thu Trang, and Pham Dinh Hung. Checking interface interaction protocols using aspect-oriented programming. In *Software Engineering and Formal Methods (SEFM 2008)*, pages 382–386, 2008.