An Approach to Communicating Process Modeling of MARTE

Zhike Wu , JingLiu[*] , Xiaohong Chen, Mingsong Chen
Shanghai Key Laboratory of Trustworthy Computing
East China Normal University
Shanghai, China
wuzhike001@gmail.com, {jliu; xhchen; mschen}@sei.ecnu.edu.cn

ABSTRACT
Precisely describing complicate interaction process is still an open problem in MARTE(Modeling and Analysis of Realtime Embedded System). In this paper, we propose an approach to modeling interaction behaviors to enhance MARTE modeling ability. MARTE is published by OMG(Object Management Group) in Aug, 2010 as a standard modeling language for modeling real time and embedded system. Our approach is based on timed CSP(Communicating Sequential Processes). To describe the multiform time structure in MARTE, we make an extension to timed CSP. The syntax and semantics of the communicating process specification are given and also the laws, the trace model and the failures model are defined. One of the main advantages of our method is to help people to modeling the complicate interaction process with process algebra, thus to simplify the modeling and verification of the interaction and concurrent behaviors in real-time and embedded systems between different processes. The approach is applied to model and analyze a Train Over Speed Protection System for Shanghai Bell Company.

Keywords
MARTE, Communicating Process, Clock Constraints, Multiform time, VOBC

1. INTRODUCTION
Time is a major concern in computer science and software engineering. In real-time systems, response time is very important determinant of correct functioning. They enable the timely delivery of services. As task complexity increase, real-time systems have become distributed[1]. The specification and verification of real-time systems have been one of the major research topics for more than twenty-five years and they also have very important practical significance since there are many real-time requirements in safety-critical systems.

In the domain of real-time and embedded (RTE) systems, the Object Management Group (OMG) has recently adopted the UML Profile for Modeling and Analysis of Real-Time and Embedded systems (MARTE), which is currently in the finalization phase. In its foundations, MARTE defines a broadly expressive Time Model to provide for a generic timed interpretation of UML models. The concrete syntax of MARTE, called CCSL(Clocked Constraint Specification Language) (CCSL)[2]. MARTE Time model deals with both discrete and dense time, physical and logical time. And a clock gives access to a time structure. A clock can be either chronometric or logical. MARTE allows different times to progress in a non-uniform fashion and possibly independently to any reference to physical time which is called multiformed time. A clock constraint is a clock relation that applies to two clock expressions [3].

[*]Corresponding Author

A complicate interaction process required to be modeled in MARTE often has no way to be clearly described, which is still an open problem in MARTE. Our approach is to model interaction behaviors in MARTE. We make an extension to the timed CSP to describe the interaction process in MARTE by means of the multiform time structure and the clock constraints. We give the syntax, semantic, failures model and failures/divergence model of the communicating process specification. Timed CSP is a real-time extension of the process algebra CSP and there are many progress in the development of timed CSP. Timed CSP has a well established refinement theory which proves useful in hierarchical system design. Timed CSP (TCSP) has incorporated temporal logic in its specification language and safety and liveness properties of systems can easily be specified using TCSP[4].

MARTE time model deals with both discrete and dense time, physical and logical time[3]. And MARTE can also deal with multiform time, so that we can express time by kilometers or degree and so on. UML has no support in such kinds of time model. Timed CSP does very good in handling communicating process. So we propose an approach to model communicating process of MARTE based on timed CSP. In our propose, we give the syntax and semantics of the communicating process specification. Also the trace model, failure model, and failures/divergence model are defined. Our approach is applied to model and analyze a Train Over Speed Protection System for Shanghai Bell Company. Our future work will focus on the tool support work.

This paper is structured in the following way: in Section 2 we give the preliminaries about MARTE time structure, clock constraints and communicating Process Modeling. in Section 3 the syntax and semantics of the communicating process specification are given. Also the trace model, failure model, and failures/divergence model are defined in this part. Our approach is applied to model and analyze a Train Over Speed Protection System for Shanghai Bell Company in Section 4. Finally in Section 5, we make a conclusion of our paper and related works and also discuss the future work.

2. PRELIMINARIES

MARTE time model deals with both discrete and dense time. In MARTE, a clock gives access to a time structure. A clock can be either chronometric or logical. The chronometric clock is related to "physical time" while the logical clock is not.

In Figure 1, a time base is a totally ordered set of instants. A set of instants can be discrete or dense. The linear vision of time represented by a single time base is not sufficient for most of the applications, especially in the case of multithreaded or distributed applications. Multiple time bases are then used. A MultipleTimeBase consists of one or many time bases. A time structure contains a tree of multiple time bases[3].

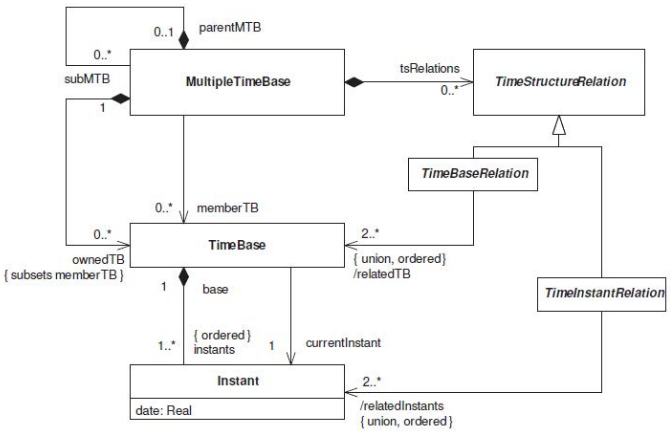


Figure 1: MARTE Time Structure

In MARTE, a *time structure* is a pair (C,R),where C is a set of clocks, R is a binary relation on $\bigcup_{c\in C}(I)_c$, named precedence. R is reflexive and transitive.

Clock is an infinite set of instants, it is not realistic to represent constraints one by one. A Clock C is a 5-tuple $\langle \mathscr{I}, \prec, \mathscr{D}, \lambda, \mu \rangle$ where $\mathscr{I}$ is a set of instants, $\prec$ is a quasi-order relation on $\mathscr{I}$, named *strict precedence*, $\mathscr{D}$ is a set of labels, $\lambda : \mathscr{I} \to \mathscr{D}$ is a labeling function, μ is a symbol, standing for a unit. For logical clocks, μ is often called tick, it can be processor cycle or any other logical activation of a behavior. The ordered set $\langle \mathscr{I}, \prec \rangle$ is the temporal structure associated with the clock. $\prec$ is a total, irreflexive, and transitive binary relation on $\mathscr{I}$[5].

For the multiform time in MARTE, for example, "the computer must shutdown in 10s" and "a car must stop within 50m". Both of the statements express a deadline that "10s" for the former sentence and "50m" for the latter. We can use either "10s" or "50m" as a clock. This is so called Multiform Time.

Clock Constraint: The Time structure defines the relations between different clock instants. Based on instant constraints, it is easy to build more powerful relations that define infinitely many instant relations. We call these relations *clock constraints.* clock constraints are classified into three families: coincident-based, precedence-based, and mixed constraints.

Timed CSP: A program in Timed CSP is a term in the abstract syntax, a language construct such with such as $a \xrightarrow{t} P$, $P_1 \overset{t}{\rhd} P_2$, $P_1 \Box P_2$ and so on. Timed CSP assumes a global clock process to maintain timing information. A process can engage in only a finite number of events in any finite time determined by the global clock. Events are assumed to occur instantaneously.

3. COMMUNICATION PROCESS MODELING FOR MARTE

We model communication process based on timed CSP. We extend the timed CSP with multiform time and clock constraints. The syntax are defined as follows:

$P ::= STOP(STOP)$
$|SKIP(SKIP)$
$|WAIT_{\phi}(WAIT)$
$|a \xrightarrow{\phi} P(timedprefix)$
$|P_1 \Box P_2(externalchoice)$
$|P_1 \sqcap P_2(internalchoice)$
$|P_1 \overset{\phi}{\rhd} P_2(timeout)$
$|f(P)(relabelling)$
$|P_1; P_2(sequentialcomposition)$
$|P_{1A} \|_B P_2(binaryparallelcomposition)$
$|P_1 ||| P_2(asynchronousparallelcomposition)$
$|\mu X \circ F(X)(recursiveprogram)$
$|P \setminus A(hiding)$

where $a \in \sum, \phi \in \mathscr{B}(\mathscr{C}).\mathscr{B}(\mathscr{C})$ represents the set of clock constraints which we will define it in next section. Actions or events will be represented over by a,b,..., clocks by A,B. Also sets of clock are represented by X,Y,... and clock constraints are represented by $\alpha, \beta, \gamma, \phi, ...$

We know that a time base is an ordered set of instants. Instants from different time bases can be bound by relationships(coincidence, precedence or mixed clock constraints). Most logical clock types use a generic time unit called tick. In some case, they may use more specific units: a processor cycle, for instance, or even units for physical quantities such as distance(km or m) or angular degree(°). We will first give the definition of the clock constraints of the ϕ.

3.1 Definition of clock constraints

We say that $\phi \in \mathscr{B}(\mathscr{C})$ and $\mathscr{C}$ represents clocks, the set of $\mathscr{B}(\mathscr{C})$ of clock constraints is generated by the following grammar:

$$\phi ::= \ \ A \subset B | A \prec B | \phi \wedge \phi | \neg \phi$$

for $A, B \in \mathscr{C}, \quad \subset \in \{\subset, \subseteq\}, \quad \prec \in \{\prec, \preceq\}$.

1. A,B represents clocks
2. $\subset \in \{\subset, \subseteq\}$ and $\prec \in \{\prec, \preceq\}$, represent the clock constraints and the binary relation between two clocks.
3. $\subset \in \{\subset, \subseteq\}$, represents coincidence-based clock constraint. You can get the detail from next part.
4. $\prec \in \{\prec, \preceq\}$
, represents precedence-based clock constraint. You can also get the detail from next part.

5. If we want to represent two clocks are identical, such as A = B. We can express it in $A \subset B$ *and* $B \subset A$

We concentrate on two kinds of clock constraints. One is coincidence-based clock constraint and another is precedence-based clock constraint.

We define two kind of clock constraints. One is Subclock relation. It includes $A \subset B$, $A \subseteq B$ and A = B. Another is Precedence relation. It includes $A \prec B$ and $A \preceq B$. In the below part, we will introduce each of them one by one, and before them, we will define some clock-related items.

1. A,B represents clocks instant.
2. offset represent one clock start after period ticks of the other clocks.
3. period means the offset of one clock and another. Such as one clock already go through 4 ticks, but another clock has only 2 ticks, then the offset is 2.
4. N^* represent nature number.
5. $k \in N^*$, A represents a clock. Then A[k] represent the k-th tick of the A clock instant.

3.1.1 Subclock

a): $A \subset B$
The next clock constraints define a subclock(A) from a given superclock(B), such as a clock with less frequent clock. However, as opposed, they can also do the contrary and define a superclock from one or a set of subclock(s). We define it by $\subset$, we give its mathematical declarative way as below.

$$A \subset B \;\; period = P \;\; offset = \delta$$
$$\Leftrightarrow$$
$$(\forall k \in N^*)(A[k] \equiv B[(k-1)*P+\delta]) \;\; (!(period = 1 \;\&\&\; offset = 0))$$

!(period =1&&offset = 0) means that not exist that period =1 and in the meantime offset = 0;

For example in Figure 2, we define two instant sets A and B and we use A $\subset$ B to represent the subclock clock constraint. Also we define a P to define the period and δ to define the offset. For example, in the below picture, we set the period 2 and the offset 3. You can see that B $\subset$ A and in another way you can say that B is a subclock of A.

b): $A \subseteq B$
$A \subseteq B$ is the almost the same with $A \subset B$, but exist the probability that A = B, so we can use the below formula to represent $A \subseteq B$.

$$A \subset B \;\; period = P \;\; offset = \delta$$
$$\Leftrightarrow$$
$$(\forall k \in N^*)(A[k] \equiv B[(k-1)*P+\delta])$$

c): A = B
We define A = B by if and only if $A \subseteq B$ and $B \subseteq A$.

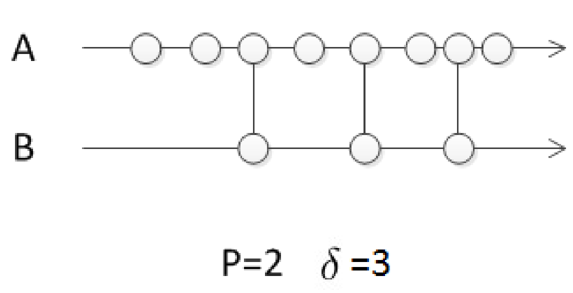


Figure 2: Subclock clock constraint

3.1.2 Precedence

Precedence clock constraint defines many precedence instant relations. They represent asynchronous relations. The most frequently used is **alternatesWith**, which means alternation between two clocks. It can be divided into two forms: the weak form($A \preceq B$) and the strict form($A \prec B$). The weak form of this relation allows the *i*th occurrence of B to be coincident with the *i*th occurrence of A, whereas the strict form doesn't. The strict form requires the two clocks to be disjoint.

a): Weak Precedence($A \preceq B$)
For the weak precedence clock relations we define A $\preceq$ B. A and B are two clock instants set. We use mathematical declarative way to express them separately below. See Figure 3.

$$A \preceq B$$
$$\Leftrightarrow$$
$$(\forall k \in N^*)(A[k] \preceq B[k] \prec A[k+1])$$

The weak precedence clock relations allows the *i*th occurrence of B to be coincident with the *i*th occurrence of A. In the Equation above, $A[k] \preceq B[k] \prec B[k]$ A[k] can be equal to B[k]. That is say, there exist some identical items.

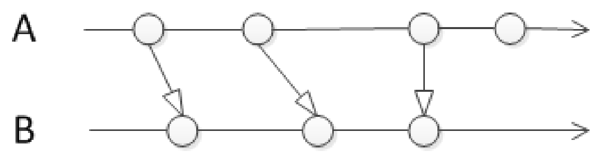


Figure 3: Weak Precedence clock constraint

b): Strict Precedence(A $\prec$ B) For the strict precedence clock relation, we define A $\prec$ B. A and B are two clock instants set. We use mathematical declarative way to express them separately below. See Figure 4.

$$A \prec B$$
$$\Leftrightarrow$$
$$(\forall k \in N^*)(A[k] \prec B[k] \prec A[k+1])$$

The strict precedence clock relations requires the two clocks to be disjoint. You can see $A[k] \prec B[k] \prec A[k+1]$ from the equation above, it mean that no identical items exist.

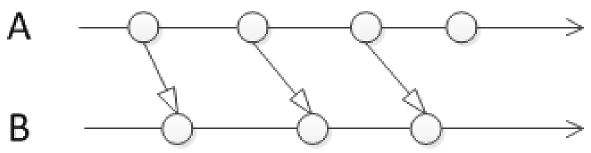


Figure 4: Strict Precedence clock constraint

3.2 Syntax

The event or action a is in the set of all event $\sum$, event set A ranges over the set of subsets of $\sum$.

1. *STOP* means just letting the time pass and it is actually a deadlock process. So we can say that $a \to STOP$ is as simple as just an event or action a.
2. *SKIP* represents terminate successfully and communicates with $\surd$ at anytime.
3. The delay process $WAIT_{\phi}$ will wait until the clock constraints is satisfied and $STOP \overset{\phi}{\rhd} SKIP$ can describe this term.
4. The Process $a \overset{\phi}{\to} P$ behaves as P after the event a is observed and then the clock constraints ϕ is satisfied.
5. There are two forms of choice: external($P_1 \Box P_2$) and internal($P_1 \sqcap P_2$). They have the same syntax with original CSP and also timed CSP.
6. The timeout operator $P_1 \overset{\phi}{\rhd} P_2$ transfers control from P_1 to P_2 if no communications occur after the clock constraints ϕ satisfied.
7. *f(P)* has a similar control structure with P but the observable events will be renamed according to the function f.
8. The sequential composition $P_1; P_2$ represents that it will start the P_2 process after P_1 is finished.
9. Parallel composition of two processes P_1 and P_2 is parameterized by two sets of events.
10. In the construct $P_{1A}\|_B P_2$, P_1 may only perform the events in A and P_2 in B, P_1 and P_2 must cooperate on events from A $\cap$ B.
11. Asynchronous parallel composition is expressed using $|||$ operator. This operator is also known as interleaved parallel composition.
12. The expression $\mu X \circ F(X)$ is used to denote recursive process and the unique fixed point of the semantic domain mapping represented by F.
13. The hiding operator in $P \backslash A$ is used to specify the hiding of events in A from the environment.

3.3 Semantics

In this section, we will define the communicating process specification of MARTE. Since our specification is based on timed CSP, we keep some of the operator from timed CSP and change some of them. In this section, we just define the ones we changed.

Wait The delay process $WAIT_{\phi}$ will wait until the clock constraints is satisfied and then it will change state (via an internal event) to become the terminating process SKIP. It is initially able to perform nothing.

$$init(WAIT_{\phi}) = \{\}$$

It may evolve to a WAIT process with a smaller argument that satisfied during the course of its delay. However, it may not evolve beyond the end of its delay.

$$\frac{}{WAIT_{\phi} \overset{\varphi}{\rightsquigarrow} WAIT_{\phi-\varphi}} \ [\varphi \in \phi]$$

At the end of its delay it performs an internal action to permit termination.

$$\frac{}{WAIT_{\phi} \overset{\phi,\varphi}{\to} SKIP}$$

The term $WAIT_{\phi}$ denotes the system behaves as P after that the system satisfies the clock constraints ϕ defines. And $WAIT_{\phi}$ can also be denoted in another way $STOP \overset{\phi}{\rhd} SKIP$.

Timed Prefixing The process $a \overset{\phi}{\to} P$ is initially prepared to engage in event $a \in \sum$.

$$init(a \overset{\phi}{\to} P) = \{a\}$$

It is prepared to wait for its environment for any clock constraints, without changing its state.

$$\frac{}{a \to P \overset{\varphi}{\rightsquigarrow} a \to P}$$

When it performs its initial event, it will begin to behave as P when it satisfies the clock constraints ϕ. It's operational semantics is as below.

$$\frac{}{a \to P \overset{(\phi,a)}{\to} P}$$

While a $\overset{\phi}{\to}$ P behaves as P when the system satisfies the clock constraints ϕ defines, after the event a is observed.

Timeout The timeout operator is a form of external choice. The process $P_1 \overset{\phi}{\rhd} P_2$ initially executes P_1, but if P_1 is failed to communicate with environment after the system has satisfied the clock constraints ϕ, then a timeout control is passed to P_2. Initially it may perform the same events as P_1.

$$init(P_1 \overset{\phi}{\rhd} P_2) = init(P_1)$$

It may evolve as P_1 evolves, but not beyond clock constraints ϕ.

$$\frac{P_1 \overset{\varphi}{\rightsquigarrow} P_1'}{P_1 \overset{\phi}{\rhd} P_2 \overset{\varphi}{\rightsquigarrow} P_1' \overset{\phi-\varphi}{\rhd} P_2} [\varphi \in \phi]$$

It may perform any events that P_1 may perform before the clock constraint ϕ; external events resolve the choice in favour of P, internal ones do not.

$$\frac{P_1 \overset{(\varphi,a)}{\rightarrow} P_1'}{P_1 \overset{\phi}{\rhd} P_2 \overset{(\varphi,a)}{\rightarrow} P_1'} \ [\varphi \in \phi] \qquad \frac{P_1 \overset{(\varphi,\tau)}{\rightarrow} P_1'}{P_1 \overset{\phi}{\rhd} P_2 \overset{(\varphi,\tau)}{\rightarrow} P_1' \overset{\phi-\varphi}{\rhd} P_2} \ [\varphi \in \phi]$$

Furthermore, the timeout may occur by the clock constraints ϕ.

$$\frac{P_1 \overset{\phi}{\rightsquigarrow} P_1'}{P_1 \overset{\phi}{\rhd} P_2 \overset{(\phi,\tau)}{\rightarrow} P_2}$$

And the timeout operator $P_1 \overset{\phi}{\rhd} P_2$ transfers control from P_1 to P_2 if no communications occur before the system satisfies the clock constraints ϕ define. If there is more than one clock constraints, we can concatenate them, such as

$$a \overset{\phi_1+\phi_2}{\rightarrow} P = a \overset{\phi_1}{\rightarrow} \ WAIT \ \phi_2; P.$$

3.4 Laws

The laws below shows that if there are more than one clock constraint, they can be checked one by one.

$$L_1: \ a \overset{\phi_1+\phi_2}{\rightarrow} P = a \overset{\phi_1}{\rightarrow} \ WAIT \ \phi_2; P$$

$$L_2: \ P_1 \overset{\phi_1+\phi_2}{\rhd} P_2 = P_1 \overset{\phi_1}{\rhd} \ WAIT \ \phi_2; P_2$$

The next laws show that the processes can be performed no matter which order they are and the result are same.

$$L_3: \ P_1 \Box P_2 = P_2 \Box P_1$$

$$L_4: \ P_1 \sqcap P_2 = P_2 \sqcap P_1$$

$$L_5: \ P_{1A} \|_B P_2 = P_{2B} \|_A P_1$$

$$L_6: \ P_1 \underset{C}{\|} P_2 = P_2 \underset{C}{\|} P_1$$

There are also some special laws that when event a communicate with SKIP, we can just skip the process SKIP. Such as:

$$L_7: \ a \overset{\phi}{\rightarrow} SKIP = a$$

$$L_8: \ a \, \| \, SKIP = a$$

$$L_9: \ P; SKIP = P$$

3.5 Trace Model

In this section we show how to calculate the set of traces of any process that using the notations introduced so far. *Stop* and *Skip* has only one trace.

$$traces(Stop) = \{s | s = \langle\rangle\} = \{\langle\rangle\}$$

$$traces(Skip) = \{s | s = \langle\rangle\} = \{\langle\rangle\}$$

For $WAIT_\phi$, no matter the clock constraints ϕ being satisfied or not, there is only one trace which is empty.

$$traces(WAIT_\phi = \{s | s = \langle\rangle\} = \{\langle\rangle\})$$

For $a \overset{\phi}{\rightarrow} P$, if the clock constraints ϕ is satisfied, then the traces contains the trace of P. If not, it only contains the event a.

$$\begin{aligned} traces(a \overset{\phi}{\rightarrow} P) &= \{s | s = \langle\rangle \vee (s_0 = a) \vee (s_0 = a \wedge s' \in traces(P))\} \\ &= \{\langle\rangle\} \cup \{a\} \cup \{(a^\frown s | s \in traces(P))\} \end{aligned}$$

For $P_1 \overset{\phi}{\rhd} P_2$, if the process P_1 has failed to communicate with any visible event and the clock constraints is satisfied, then the traces are the trace of P_1 and P_2. Else, the trace contains only the trace of P_1.

$$\begin{aligned} traces(P_1 \overset{\phi}{\rhd} P_2) &= \{s | s = \langle\rangle \vee (s | s \in traces(P_1)) \vee (s | s \in traces(P_2))\} \\ &= \{\langle\rangle\} \cup \{s | s \in traces(P_1)\} \cup \{(s | s \in traces(P_1)) \wedge (s | s \in traces(P_2))\} \end{aligned}$$

3.6 Failure Model

Refusal: In general, let X be a set of events which are offered initially by the environment of a process P, which in this context we take to have the same alphabet as P. If it is possible for P to deadlock on its first step when placed in this environment, we say that X is a refusal of P. The set of all such refusals of P is denoted as *refusals*(P).

$$refusals(P)\{X | (\langle\rangle, X) \in failures(P)\}$$

The refusals of Wait, Timed Prefixing and Timeout are as below:

$$refusals(WAIT_\phi) = \{\}$$

$$refusals(a \overset{\phi}{\rightarrow} P) = \{X | X \subseteq (refusals(P))\}$$

$$refusals(P_1 \overset{\phi}{\rhd} P_2) = refusals(P_1) \cap refusals(P_2)$$

Divergence: *CHAOS* is a status that the system is do something internal and not communication with external, it's a very bad status. A divergence model of a process is defined to handel the *CHAOS* condition. It will behavior as any trace of the process after which the process behaves chaotically.

$$divergences(P) = \{s | s \in traces(P) \wedge (P \setminus s) = CHAOS_{\alpha P}\}$$

It follows that $divergences(P) \in traces(P)$

The divergence of Wait, Timed Prefixing and Timeout are as below:
$divergences(WAIT_{\phi}) = \{\}$

For $a \xrightarrow{\phi} P$, if the clock constraints ϕ is satisfied, then the divergences is $\{\langle\alpha\rangle^\frown s | \alpha \wedge s \in divergences(P)\}$. If not, the divergences is $\{\}$.

$divergences(a \xrightarrow{\phi} P) = \{\}$
$|\{\langle\alpha\rangle^\frown s | \alpha \wedge s \in divergences(P)\}$

For $P_1 \overset{\phi}{\triangleright} P_2$, if the clock constraints ϕ is satisfied, then the divergences is $divergences(P_1) \cup divergences(P_2)$. If not, the divergences is $divergences(P_1)$.

$divergences(P_1 \overset{\phi}{\triangleright} P_2) = divergences(P_1)$
$|divergences(P_1) \cup divergences(P_2)$

Failures model: The *failures* models extends the traces model with refusal sets, which are set of event $X \subseteq \sum$ that a process can refuse to perform. A failure is a pair (s, X), consisting of a trace s, and a refusal set X which identifies the events that a process may refuse once it has executed the trace s. The observed behavior of a process in the stable failures model is described by the pair (*trace*(P), *failures*(P)).

$$failures(P) = \{(s,X) | s \in traces(P) \wedge (X \in refusals(P \setminus s))\}$$

The failures of Wait, Timed Prefixing and Timeout are as below:
$failures(WAIT_{\phi}) = \{\}$

For $a \xrightarrow{\phi} P$, if the clock constraints ϕ is satisfied, then the failures is $\{(s,X) | s \in traces(P) \wedge X \in refusals(P/s)\}$. If not, the failures is $\{\}$.

$failures(a \xrightarrow{\phi} P) = \{\}$
$|\{(s,X) | s \in traces(P) \wedge X \in refusals(P/s)\}$

For $P_1 \overset{\phi}{\triangleright} P_2$, if the clock constraints ϕ is satisfied, then the failures is $\{(s,X) | s \in traces(P_1) \wedge X \in refusals(P_1/s)\} \cup \{(s,X) | s \in traces(P_2) \wedge X \in refusals(P_2/s)\}$. If not, the failures is $\{(s,X) | s \in traces(P_1) \wedge X \in refusals(P_1/s)\}$.

$failures(P_1 \overset{\phi}{\triangleright} P_2) = \{(s,X) | s \in traces(P_1)$
$\wedge X \in refusals(P_1/s)\}$
$|\{(s,X) | s \in traces(P_1) \wedge X \in refusals(P_1/s)\}$
$\cup \{(s,X) | s \in traces(P_2) \wedge X \in refusals(P_2/s)\}$

Failures/divergence model: The *failures/divergence* model further extends the failure model to handle divergence. The semantics of a process in the failures/divergences model is a pair ($failures_{\perp} P, divergence(P)$) where *divergence(P)* is defined as the set of all traces that can lead to divergent behavior.

$$failures_{\perp} P = failures(P) \cup \{(s,X) | s \in divergences(P)\}.$$

The failures of Wait, Timed Prefixing and Timeout are as below:
$failures_{\perp}(WAIT_{\phi}) = \{\}$

For $a \xrightarrow{\phi} P$, if the clock constraints ϕ is satisfied, then the failures/divergence is $\{(s,X) | s \neq \langle\rangle \wedge (s,X) \in failures(P)\} \cup \{(s,X) | s \in divergence(P)\}$. If not, the failures/divergence is $\{\}$.

$failures_{\perp}(a \xrightarrow{\phi} P) = \{\}$
$|\{(s,X) | s \neq \langle\rangle \wedge (s,X) \in failures(P)\}$
$\cup \{(s,X) | s \in divergence(P)\}\}$

For $P_1 \overset{\phi}{\triangleright} P_2$, if the clock constraints ϕ is satisfied, then the failures/divergence is $\{(s,X) | s \neq \langle\rangle \wedge (s,X) \in failures(P_1) \cup failures(P_2)\} \cup \{(s,X) | s \in divergence(P_1) \wedge divergence(P_2)\}$. If not, the failures/divergence is $\{(s,X) | s \neq \langle\rangle \wedge (s,X) \in failures(P_1)\} \cup \{(s,X) | s \in divergence(P_1)\}$.

$failures_{\perp}(P_1 \overset{\phi}{\triangleright} P_2) = \{(s,X) | s \neq \langle\rangle \wedge (s,X) \in failures(P_1)\}$
$\cup \{(s,X) | s \in divergence(P_1)\}\}$
$|\{(s,X) | s \neq \langle\rangle \wedge (s,X)$
$\in failures(P_1) \cup failures(P_2)\}$
$\cup \{(s,X) | s \in divergence(P_1) \wedge divergence(P_2)\}$

4. TRAIN OVER SPEED PROTECTION SYSTEM

Safety is very important to urban metro system and VOBC(Vehicle OnBoard Controller) system is used to guarantee the safety and control the operation of the subway system. There are several main functions in VOBC, such as Mode Switch, Train Integrity Monitoring, Over Speed Supervision and Protection, Zero Speed Relay and so on. In this section, we take the Over Speed Supervision and Protection system as an case study.

Over Speed Supervision and Protection system is a new type of vehicle equipment. It is of great significance in prevent the the over speed of the train. When the result of train speed plus the tolerance speed is big than the permitted speed, they have to apply common brake or emergency brake to protect the train. The below formula shows how can we determine whether the train is over speed or not.

$$trainSpeed + permittedSpeed > toleranceSpeed$$

1. *trainSpeed* represents the current speed of the train.
2. *permittedSpeed* represents the permitted speed in a mode and is different in different modes.
3. *toleranceSpeed* represents the tolerant speed in a mode. That is a limited speed. It is also different in different modes.

There are several modes in VOBC, they are RMR(Restricted Manual Reverse), RMF(Restricted Manual Forward), WM(Wash mode), OFF and WSP(Wayside Signal Protection). The permitted speed and tolerance speed is different in different modes. In RMR, RMF and WM modes, we don't need the permitted speed. So if $(trainSpeed > toleranceSpeed)$, we say the train is over speed. In OFF mode, if the train is not still, we say the train is over speed. But in WSP mode, there has a permitted speed and it very complicated to get it. We will not discuss how to get the permitted speed, we just to know that there is a permitted speed in this mode and if $(trainSpeed + permittedSpeed > toleranceSpeed)$, then the train is over speed. If the train is over speed, the emergency brake must be applied.

Even if the situation is different in different mode, but we can make an assumption. We assume there is permitted speed in RMR/RMF/WM/OFF modes and the permitted speed is 0. So in OFF mode, the permitted and tolerance speed is both 0. Then we can say that no matter in which mode, if $(trainSpeed + permittedSpeed > toleranceSpeed)$, the train is over speed.

So the whole process becomes that the driver initiate the train using the monitor device and then change the mode of the train. VOBC get the permitted and the tolerance speed by TIM and get the current speed of the train by the speed sensor. After that, VOBC have to determine whether the train is over speed or not. If the train is over speed, VOBC must apply the Emergency Brake. If not, if the train have to stop, Driver can apply common brake. After that, VOBC have to clear the history of the emergency brake. We can see the whole process by the sequence diagram of train over speed protection system in Figure 5: Then we got to know that no matter in which mode, if VOBC is in status of "Not Active" or the Emergency Brake is applied, VOBC must set the tolerance speed to 0. In another words, if the VOBC is not active, if the train runs a distance, we say that's not make sense and we must call an Emergency Brake. In the description of "a distance", that's kind of multiform time. In another circumstance, if the emergency brake is applied, VOBC must set the toleranceSpeed to 0. We change it in another way to say, if there's distance of the train when emergency brake applied, we must set toleranceSpeed to 0. That's very like the condition we say just now.

Train overspeed protection system have four different conditions, which are used to decide whether to call the emergency brake. When VOBC is not active, the train is off and the emergency brake has already been applied, then set the tolerance speed to 0, and when the train speed is bigger than 0, the emergency brake must apply. When the train is not in the conditions above, the driver select mode, and the tolerance speed is different according to the mode and then, if the train speed is large than the tolerance speed, the emergency must be applied then. We can say in brief description:

(1) If the VOBC is not active, toleranceSpeed = 0. So if trainSpeed $>$ 0, then apply Emergency Brake

(2) If the VOBC is OFF, toleranceSpeed = 0, So if trainSpeed $>$ 0, then apply Emergency Brake

(3) If Emergency Brake has already applied, toleranceSpeed = 0, So if trainSpeed $>$ 0, then apply Emergency Brake

(4) If Emergency Brake has already applied, toleranceSpeed is based on which mode, if trainSpeed $>$ toleranceSpeed, then apply Emergency Brake

We define the state diagram of the call emergency brake function below. TS represents the speed of the train and TSM represents the tolerance speed in the selected model. And we also need to say that "$\succ$" does not mean that "bigger than", it mean that one time point is "earlier" than the other time point. And you can see the state transition of train oversoeed protection system with clock constraints in Figure 6.

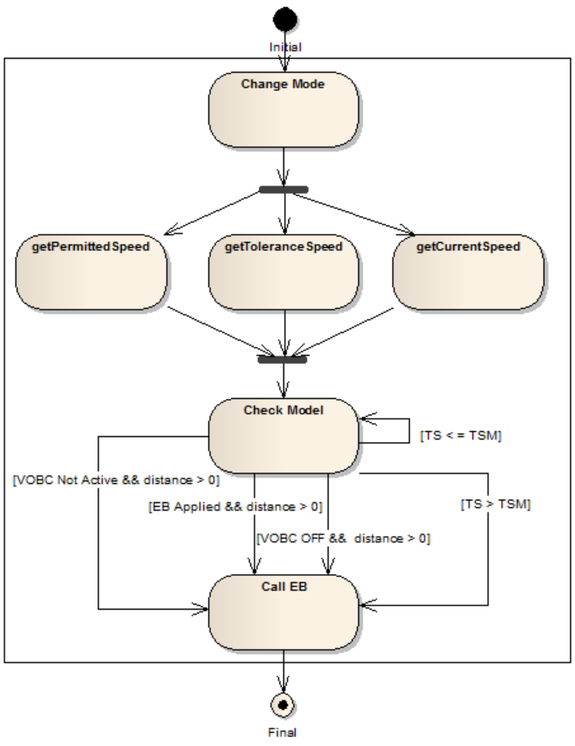


Figure 6: State transition of train overspeed protection system with Clock Constraints

After we analysis the property of the train over speed supervisor and protection system with natural language and also UML chart, now we define the models using our new language: Multiform Time CSP:

Before we define the model, we have to define some process and clock constrains.

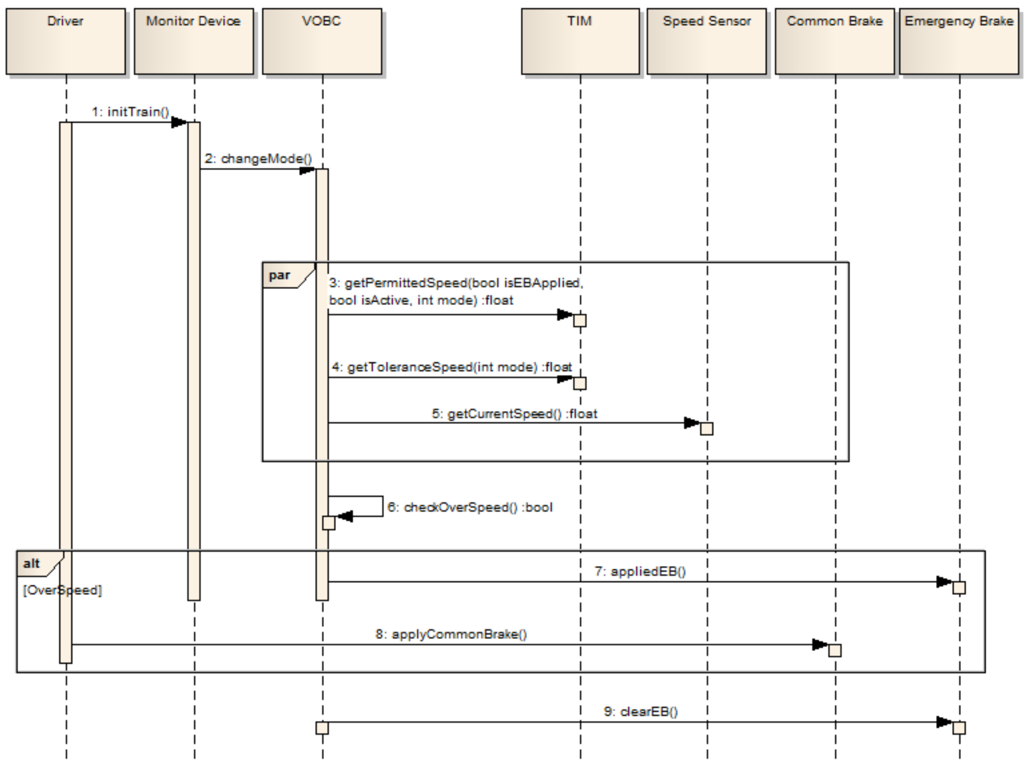
TOSSP: means Train Over Speed Supervisor and Protection system process. It represent the whole process of the system and we will check its safety property then.

ϕ: means a clock constraint which represents the time is over than the time we want to check.

OSC: mean Over Speed Check. It represent to check whether the train is over speed

EB: mean the train applied Emergency Brake.

The model is as below:
$TOSSP = trainInit$
$\rightarrow (checkVOBCActive$



Driver
Monitor Device
VOBC
TIM
Speed Sensor
Common Brake
Emergency Brake
1: initTrain()
2: changeMode()
par
3: getPermittedSpeed(bool isEBApplied, bool isActive, int mode) :float
4: getToleranceSpeed(int mode) :float
5: getCurrentSpeed() :float
6: checkOverSpeed() :bool
alt
[OverSpeed]
7: appliedEB()
8: applyCommonBrake()
9: clearEB()

Figure 5: Sequence Diagram of Train OverSpeed Protection System

$\rightarrow ((VOBCActive \rightarrow OSC)$
$|((VOBCNotActive$
$\rightarrow (WAIT \xrightarrow{distance \succ 0} EB)$
$|(WAIT \overset{\phi}{\triangleright} STOP))))$
$\Box checkEBApplied$
$\rightarrow (Applied \xrightarrow{distance \succ 0} EB)$
$|(NotApplied \rightarrow OSC))$

$OSC = selectMode$
$\rightarrow getPermittedSpeed$
$\rightarrow getToleranceSpeed$
$\rightarrow ((\mu GetTS \bullet GetTS \overset{\phi}{\triangleright} STOP)$
$|(\mu GetTS \bullet GetTS \xrightarrow{trainSpeed+permittedSpeed \succ toleranceSpeed}$
$STOP))$

We define the safety property of the over speed supervisor and protection system. We can only say the train is safe if the train stop without over speed or when it is over speed, the Emergency brake is applied.
Safety Property: $TOSSP = trainInit \rightsquigarrow STOP|EB$

We analysis the Train Over Speed Supervision and Protection system in detail.
$TOSSP = trainInit \rightarrow$

$(checkVOBCActive \rightarrow$
$((VOBCNotActive \rightarrow (WAIT \xrightarrow{distance \succ 0} EB)$

$|(VOBCActive$
$\rightarrow selectMode$
$\rightarrow getPermittedSpeed$
$\rightarrow getToleranceSpeed$
$\rightarrow (\mu GetTS \bullet GetTS \overset{\phi}{\triangleright} STOP)$
$|(\mu GetTS \bullet GetTS \xrightarrow{trainSpeed+permittedSpeed \succ toleranceSpeed}$
$STOP))$

$\Box checkEBApplied \rightarrow$
$(Applied \xrightarrow{distance \succ 0} EB)$

$|(NotApplied$
$\rightarrow selectMode$
$\rightarrow getPermittedSpeed$
$\rightarrow getToleranceSpeed$
$\rightarrow (\mu GetTS \bullet GetTS \overset{\phi}{\triangleright} STOP)$
$|(\mu GetTS \bullet GetTS \xrightarrow{trainSpeed+permittedSpeed \succ toleranceSpeed}$
$STOP))$

We can see that in the TOSSP process, not matter it goes to which branch, it will ends with stop, EB or OSC. And let's see the OSC process. You can easily find that each branch ends with STOP or EB. They we can say that for the process TOSSP, the process will either ends with STOP or applied Emergency Brake. And it satisfies the safety property defined before.

5. RELATED WORK

C. André, F. Mallet, M-A.Peraldi-Frati, defined extensions to the simple time model of UML2. They focus on the specificity of the ablity to take account of multiform time and use an example to analyze behaviors depending on multiform time [5]. This work is very good at modeling of multiform clock and the case study is classic and representive. But our work focus on modeling interaction behaviors in MARTE. We make an extension to the timed CSP to describe the interaction process in MARTE by means of the multiform time structure and the clock constraints to better model and verification the concurrent systems.

Stefano Cattani and Marta Kwiatkowska propose a real-time extension to the process algebra CSP. they handle real time by means of clocks[7]. Based on timed CSP, our work is to work is to describe the interaction process in MARTE using the multiform time and clock constraints in MARTE. JOEL OUAKINE and JAMES WORRELL proposed some mild modifications to the syntax and semantics of Timed CSP which significantly increase expressiveness and able to capture some of the most widely used specifications on timed systems as refinements (reverse inclusion of sets of behaviors) which may then be verified using the model checker FDR[8]. There work does not handle multi-clock problems.

There has been many synchronous languages, such as Esterel, Lustre, SCADE, Signal, etc. Esterel is used to program reactive systems. The advantage of Esterel is that it allows the simple expression of parallelism and preemption[9] SCADE (Safety Critical Application Development Environment) is a tool-suite based on the synchronous paradigm and the Lustre language. It is able to be used in highest criticality applications[10].

Our approach is to model interaction behaviors in MARTE. We make an extension to the timed CSP to describe the interaction process in MARTE by means of the multiform time structure and the clock constraints. We give the syntax, semantic, failures model and failures/divergence model of the communicating process specification. Timed CSP is a realtime extension of the process algebra CSP and there are many progress in the development of timed CSP. Timed CSP has a well established refinement theory which proves useful in hierarchical system design. Timed CSP (TCSP) has incorporated temporal logic in its specification language and safety and liveness properties of systems can easily be specified using TCSP[4].

6. CONCLUSION AND FUTURE WORK

MARTE time model are very suitable for describing discrete and dense time, physical and logical time, multiform time. Timed CSP handles is designed for communicating process modeling.

In this paper, we propose an approach to modeling interaction behaviors in MARTE. We make an extension to the timed CSP to describe the interaction process in MARTE by means of the multiform time structure and the clock constraints. The syntax, semantic as well as the failures model and failures/divergence model of the communicating process specification, are given. Our approach is applied to describe a real-time system, train integrity monitor system of VOBC, to simplify the modeling and verification of the real-time and embedded system for Shanghai Bell Company.

Future work will focus on the application of more areas and extend the classic CSP modeling and verification tools to support the communicating process specification of MARTE.

Acknowledgment

We would like to thank the reviewers of this paper for their valuable comments. This work is partially supported by the projects funded by the 973 program, NSFC project 61170084, NSFC Creative Team 61021004, NSFC for Young 60903021 and 61202104, 863 Project 2011AA010101.

References

[1] Jaipal Singh, Omar Hussain, Elizabeth Chang, Tharam Dillon, *Event Handling for Distributed Real-Time Cyber-Physical Systems*. 2012 15th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing.

[2] Charles André, Frédéric Mallet, *Clock Constraints in UML/MARTE CCSL*. Research Report 6540, INRIA (2008).

[3] Charles André, Frédéric Mallet,Robert de Simone, *Time modeling in MARTE*. I3S, Université de Nice-Sophia Antipolis, CNRS, F-06903 Sophia Antipolis, Aoste Project, I3S/INRIA, 2007.

[4] Ahmet Feyzi Ates, Murat Bilgic, Senro SGto, and Behcet Sarikaya, Senior Member, ZEEE, *Using Timed CSP for Specification Verification and Simulation of Multimedia Synchronization*. IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS, VOL. 14, NO. 1, JANUARY 1996.

[5] Frédéric Mallet, *Clock constraint specification language: specifying clock constraints with UML/MARTE*. http://hal.archives-ouvertes.fr/hal-00677925. 2012

[6] C. André, F. Mallet, M-A. Peraldi-Frati, *A multiform time approach to real-time system modeling*. Int. Symp. on Industrial Embedded Systems, 2007, SIES ąŕ07, pp. 234- 241,06903 VALBONNE, FRANCE .

[7] Stefano Cattani and Marta Kwiatkowska, *CSP + Clocks: a Process Algebra for Timed Automata**. School of Computer Science The University of Birmingham Birmingham B15 2TT,United Kingdom, April 2003

[8] Kun Wei, Jim Woodcock and Alan Burns, *Timed Circus: Timed CSP with the Miracle*. 16th IEEE International Conference on Engineering of Complex Computer Systems, 2011

[9] Wahbi Haribi, *Compiling Esterel for Multi-Core Execution*. Synchrone Sprachen, 2012.

[10] Ning Ge and Marc Pantel, *From Simulink to SCADE/ Lustre to TTA: a Layered Approach for Distributed Embedded Applications**. LCTESąŕ03, June 11-13, 2003, San Diego, California, USA.