The International Journal of Time-Critical Computing Systems, 20, 27–49, 2001

Extending the SCR Method for Real-Time Systems

HYOUNG SEOK HONG hshong@salmosa.kaist.ac.kr
EECS Department and Advanced Information Technology Research Center (AITrc)
Korea Advanced Institute of Science and Technology (KAIST)

SEUNG MO CHO seung@salmosa.kaist.ac.kr
EECS Department and Advanced Information Technology Research Center (AITrc)
Korea Advanced Institute of Science and Technology (KAIST)

SUNG DEOK CHA cha@salmosa.kaist.ac.kr
EECS Department and Advanced Information Technology Research Center (AITrc)
Korea Advanced Institute of Science and Technology (KAIST)

YONG RAE KWON kwon@salmosa.kaist.ac.kr
Korea Advanced Institute of Science and Technology (KAIST)

Abstract. We propose timed SCR specifications, which are a generalization of SCR specifications, intended to specify quantitative timing properties of real-time systems. We extend the tabular notation of the SCR method to deal with sporadic and periodic timing constraints. We present a formal semantics for timed SCR specifications by translating them into timed transition systems. A shutdown system in Korean nuclear power plants is used as a case study to illustrate timed SCR specifications.

Keywords: the SCR method, real-time systems, formal specifications, timing constraints

1. Introduction

The Software Cost Reduction (SCR) method was originally developed as a disciplined approach to requirements specification of A-7 aircraft flight program at NRL (Heninger, 1980). The A-7 requirements document introduces the basic features of the SCR method—a set of tabular notation such as mode transition tables, event tables, and condition tables, and high-level constructs such as input and output data items, conditions, events, modes, and mode classes. Since its introduction, the SCR method has been successfully applied to a variety of real-world applications including NRL's A-7 aircraft (Alspaugh et al., 1992), Lockheed's C-130J (Faulk et al., 1994), and the nuclear power plants in Canada (van Schouwen, Parnas, and Madey, 1993), and Belgium (Courtois and Parnas, 1993). Recently, a number of researchers have extended the original SCR method including the four variable model (Parnas and Madey, 1990; van Schouwen, 1990) and the CoRE method (Faulk et al., 1992), and developed formal semantics and analysis methods (Atlee and Gannon, 1993; Atlee and Buckley, 1996; Bharadwaj and Heitmeyer, 1997; Heitmeyer, Jeffords, and Labaw, 1996).

Van Schowen (1990) re-examined the A-7 document, extending the original SCR method to specify both system and software requirements and describe precision and tolerance as well as functional requirements. The four variable model of Parnas and Madey (1990) gives a formal framework for the works of van Schowen (1990) by specifying the required behavior of a system in terms of a set of mathematical relations between four kinds of

variables—monitored and controlled variables, and input and output data items. The CoRE method by Faulk et al. (1992) is an effort to integrate the four variable model with object-oriented methodologies. Atlee et al. (Atlee and Gannon, 1993; Atlee and Buckley, 1996) proposed model checking methods for a subclass of SCR specifications which contain only mode transition tables with boolean input variables. In Atlee and Gannon (1993), an operational semantics is proposed which shows how to generate global reachability graphs from SCR specifications. In Atlee and Buckley (1996), the behavior of each mode transition table is interpreted as a set of temporal formulas and the global behavior is defined as the logical conjunction of these formulas. Heitmeyer et al. (1996) proposed consistency checking methods for SCR specifications. In their approach, each table is represented by a mathematical function and the global behavior is defined as the functional composition of these functions. Based on this semantics, Bharadwaj and Heitmeyer (1997) proposed a model checking method for SCR specifications.

Although these works have greatly strengthened formal foundations of the SCR method, most of them primarily focus on untimed system behavior and do not consider quantitative timing properties of real-time systems. In recent years, several languages have been proposed for specifying real-time systems and a large number of them are timed versions of formalisms, originally designed to specify untimed system behavior. They usually support several language constructs for the specification of two typical real-time features: sporadic and periodic timing constraints (e.g., timed Statecharts (Kesten and Pnueli, 1992) and Modecharts (Jahanian and Mok, 1994)). This paper focuses on how the SCR method can be extended to specify real-time systems. We propose *timed SCR specifications* in which the tabular notation of the SCR method is extended to deal with sporadic and periodic timing constraints. We consider real-time systems over the discrete time domain assuming a system-wide global clock. We choose to use natural numbers for the domain of time because they are often used for specifying computer systems. To illustrate how real-time systems are specified in timed SCR specifications, we take as a case study a shutdown system in Korean nuclear power plants

We present a formal semantics for timed SCR specifications based on a translational approach. The idea of translational semantics is to express the meaning of a language by a translation scheme which, for any program in the language, yields a program in a simpler and, it is hoped, better understood language (Meyer, 1990). In recent years, this approach has been adopted for the semantic definition of specification languages for reactive and concurrent systems (Manna and Pnueli, 1989; Manna and Pnueli, 1993; Petersohn and Urbina, 1997). In these works, a specification language (e.g., Statecharts (Harel, 1987)) is given a semantics by showing how the language can be translated into a generic computational model (e.g., transition systems (Keller, 1976; Pnueli, 1977)). A main benefit from these works is that a number of analysis methods developed for the generic models can be applied to the specification languages. In this paper, a translation scheme is proposed which maps timed SCR specifications into timed transition systems (Henzinger, Manna, and Pnueli, 1992; Manna and Pnueli, 1993). The SCR method provides a set of tabular notation and high-level constructs which make the method easy-to-use, intuitive and scalable to complex applications. On the other hand, timed transition systems are a generic model for real-time systems in the sense that they consist of primitive concepts such as states, transitions and

timing constraints. With the translation, we aim to: (i) capture the intuitive meaning of each high-level concept in timed SCR specifications only using the primitive concepts of timed transition systems. (ii) define the global behavior of timed SCR specifications in terms of computations of the resulting timed transition systems.

The remainder of the paper is organized as follows: Section 2 briefly reviews the SCR method and proposes the abstract syntax of timed SCR specifications. Section 3 introduces a case study and shows how various real-time constraints are specified in timed SCR specifications. Section 4 reviews timed transition systems and gives overview of the semantics. Section 5 describes the behavior of each table in timed SCR specifications. Section 6 describes the global behavior of timed SCR specifications. Finally, we compare our work with the existing semantics and discuss verification methods for timed SCR specifications in Section 7.

2. Timed SCR Specifications

In the SCR method, requirements are specified mainly in two steps. At first, hardware interfaces are described in terms of input data items and output data items. Software is then described as a set of functions associated with output data items. Each function determines the values for one or more output data items and each output data item is given values by exactly one function. The set of entities E used in the SCR method can be partitioned as follows:

- E_I is the set of *input data items*.
- E_O is the set of *output data items*.
- E_T is the set of *terms*. A term is an auxiliary variable used for the concise description of software functions.
- E_M is the set of *mode classes*. Each mode class $MC_i \in E_M$ is a set of modes. A *mode* represents a class of system states. System states are grouped together into a mode according to properties that affect the gross behavior of the system.

We use the naming convention in Heninger (1980): `/name/`, `//name//`, `*name*`, `**name**`, and `!name!` are used for denoting input data items, output data items, modes, mode classes, and terms, respectively.

A *condition* is a predicate over the entities. There are two special conditions, *Inmode* and *Inmode(me)*, where *me* stands for a *mode expression*. These conditions are used to represent the current modes of a system. A mode expression is a set of modes connected with boolean connectives $\wedge$, $\vee$, and $\neg$. For example, if m_1, m_2, and m_3 are modes, then $(m_1 \wedge m_2) \vee m_3$ is a mode expression. The condition $Inmode((m_1 \wedge m_2) \vee m_3))$ intuitively means that a system is either in m_1 and m_2, or in m_3. Mode expressions enable the succinct description of requirements when the behavior is determined by the current modes of two or more mode classes.

An *event* represents a change of values in a condition. Let *Cond1* and *Cond2* be conditions. We denote an event by either @T(*Cond1*) WHEN *Cond2* or @F(*Cond1*) WHEN *Cond2*

whose meaning is that the event occurs when *Cond1* changes its value from false to true (from true to false, resp.) while *Cond2* is true. If an event is the form of @T(*Cond1*) WHEN true, it is abbreviated to @T($Cond1$). We refer to *Cond1* and *Cond2* as *triggering condition* and *WHEN condition*, respectively.

Each mode class is associated a *mode transition table* that describes the changes between modes in that mode class. For each output data item or term, there exists an *event table* or *condition table* that describe the changes of the output data item or the term. We present an abstract syntax for the tabular notation of the SCR method as follows:

- Let MC be a mode class. A *mode transition table* for MC is a set of *mode transitions* $\langle m, e, m' \rangle$ where $m, m' \in MC$ and e is an event.
- Let x be an output data item or a term. An *event table* for x is a set of *actions* $\langle me, e, x := v \rangle$ where me is a mode expression, e is an event, and v is an expression over the entities in E.
- Let x be an output data item or a term. A *condition table* for x is a set of *activities* $\langle me, c, x := v \rangle$ where me is a mode expression, c is a condition, and v is an expression over the entities in E.

We propose *timed SCR specifications* by extending the tabular notation of the SCR method to deal with sporadic and periodic timing constraints. We take a set of non-negative integers N as the time domain and assume that $\infty \geq n$ for all $n \in \mathsf{N}$. Sporadic constraints such as delays, deadlines, and time-outs restrict the times at which the changes of system states can occur. We partition sporadic constraints into two categories: *lower* and *upper bound* requirements. These bounds are used to guarantee that the changes occur neither too early nor too late, respectively. Let l be a *label* defined as

- e, where e is an event or
- $c[l, u]$, where c is a condition, $l \in \mathsf{N}$, and $u \in \mathsf{N} \cup \{\infty\}$.

To specify sporadic constraints we extend mode transition tables and event tables in the SCR method as follows:

- A *mode transition table* for a mode class MC is a set of *mode transitions* $\langle m, l, m' \rangle$ where $m, m' \in MC$ and l is a label.
- An *event table* for an output data item or a term x is a set of *actions* $\langle me, l, x := v \rangle$ where me is a mode expression, l is a label, and v is an expression over the entities in E.

A mode transition or an action is said to be *immediate* if its label is an event. Intuitively, an immediate mode transition occurs at the same time when its triggering event occurs. When the label is of the form $c[l, u]$, mode transitions and actions can occur at an instant between l and u time units only if the condition c has been continuously satisfied. Therefore, we can regard mode transition tables and event tables in the SCR method as a special case in which all the labels are events.

Periodic constraints require some changes of system states to be performed synchronously as the time changes without being requested each time. To specify periodic constraints, we extend condition tables in the SCR method as follows:

- A *condition table* for an output data item or a term x is a set of *activities* $\langle me, c, x := v \rangle$ augmented with the *period* $p \in N$.

Intuitively, we model the behavior of condition tables such that the set of activities describes a mathematical relation which determines the value of x. When a condition table is performed every p time unit, one of the activities in the condition table is selected according to the values of mode expression me and the condition c of each activity.

3. A Case Study

To demonstrate the scope of timed SCR specifications, we take as a case study Shutdown Systems Number 2 (SDS2) of Wolsung nuclear power plants 2/3/4 in Korea (AECL CANDU, 1993). SDS2 is a software-based emergency shutdown system whose purpose is to implement complex functions which are difficult to implement in conventional analog hardware and relay logic. Basically, SDS2 monitors the status of the nuclear reactor (e.g., pressure and power) and generates a trip signal (shutdown command) when the reactor becomes unsafe. The requirements for SDS2 consist of a variety of quantitative timing constraints including immediate trips and delayed trips (sporadic constraints) and periodic computation of output data items and terms (periodic constraints).

3.1. The PPH Trip Parameter

There are six trip parameters in SDS2 and we consider one of them, *PHT High Pressure* (PPH) trip parameter. In the PPH trip, there are four input data items `/PHTD[i]/, i = 1..4` that represent *four PHT core differential pressure* and a term `!FlinC!` represents the *compensated ion chamber linear power*. When the PPH trip is generated, the output data item `//PPHTrip//` is assigned the value of `Open`. Otherwise, it is assigned the value of `Close`. Requirements for the PPH trip mainly consist of two parts: *immediate trip* and *delayed trip*.

We specify the immediate trip using a mode class `**ImdMode**`. Table 1 shows the mode transition table for `**ImdMode**` and its tabular notation. Let `ImdIrr` be a condition defined as $\bigvee_{i=1}^{4}$ `(/PHTD[i]/` $\leq$ `200` $\vee$ `/PHTD[i]/` $\geq$ `4118)`. This condition indicates that the values of four pressure signals are out of normal ranges. The mode transition m_1 states that

> "if the current mode is `*ImdNotTrip*` (the immediate trip is not generated) and the condition `ImdIrr` becomes true, then the mode class changes its mode into `*ImdTrip*` (the immediate trip is generated)."

The delayed trip is determined by `/PHTD[i]/ i=1..4` and `!FlinC!`. We specify the delayed trip using a mode class `**DlyMode**` (Table 2). Let `DlyIrr` and `FlinCIrr` be

Table 1. Mode transition table for `**ImdMode**`.

m_1: ⟨ ∗ `ImdNotTrip`∗, @`T`(`ImdIrr`), ∗`ImdTrip`∗⟩
m_2: ⟨∗`ImdTrip`∗, @`F`(`ImdIrr`), ∗`ImdNotTrip`∗⟩

Current mode	Label	Next mode
`*ImdNotTrip*`	`@T(ImdIrr)`	`*ImdTrip*`
`*ImdTrip*`	`@F(ImdIrr)`	`*ImdNotTrip*`

conditions defined as $\bigvee_{i=1}^{4}$ (`/PHTD[i]/` $\geq$ 3736) and `!FlinC!` $\geq$ 70, respectively. The delayed trip is generated in the modes `*Active*` and `*Wait*`, while the other modes are used to represent the normal status. The delayed trip is determined as follows:

- When the current mode is `*Idle*` and the condition `DlyIrr` and `FlinCIrr` becomes true, continue normal operation during $[l_1, u_1]$, i.e., `*Pend*` becomes the current mode.
- When the interval $[l_1, u_1]$ has expired, i.e., the mode is changed into `*Decide*`, generate the delayed trip (go to `*Active*`) if `FlinCIrr` is still satisfied. Otherwise, do not generate the delayed trip (go to `*Idle*`).
- Once the delayed trip has been generated, i.e., the current mode is `*Active*`, keep it generated for $[l_2, u_2]$ (`*Active*`).
- After the interval $[l_2, u_2]$ has expired, i.e., the current mode is changed into `*Wait*`, do not generate the delayed trip once the condition `(DlyIrr` $\wedge$ `FlinCIrr)` becomes false.

Table 2. Mode transition table for `**DlyMode**`.

m_3: ⟨∗`Idle`∗, `@T(DlyIrr` $\wedge$ `FlinCIrr)`, ∗`Pend`∗⟩
m_4: ⟨∗`Pend`∗, $[l_1, u_1]$, ∗`Decide`∗⟩
m_5: ⟨∗`Decide`∗, ¬`FlinCIrr[0, 0]`, ∗`Idle`∗⟩
m_6: ⟨∗`Decide`∗, `FlinCIrr[0, 0]`, ∗`Active`∗⟩
m_7: ⟨∗`Active`∗, $[l_2, u_2]$, ∗`Wait`∗⟩
m_8: ⟨∗`Wait`∗, @`F`(`DlyIrr`∧`FlinCIrr`), ∗`Idle`∗⟩

Current mode	Labels	Next mode
`*Idle*`	`@T(DlyIrr` $\wedge$ `FlinCIrr)`	`*Pend*`
`*Pend*`	$[l_1, u_1]$	`*Decide*`
`*Decide*`	¬`FlinCIrr[0, 0]`	`*Idle*`
`*Decide*`	`FlinCIrr[0, 0]`	`*Active*`
`*Active*`	$[l_2, u_2]$	`*Wait*`
`*Wait*`	`@F(DlyIrr` $\wedge$ `FlinCIrr)`	`*Idle*`

Table 3. Event table for //PPHTrip//.

a_1: ⟨∗ImdTrip∗ ∨ ∗Active∗ ∨ ∗Wait∗, @T(Inmode), //PPHTrip// := Open⟩
a_2: ⟨∗ImdTrip∗ ∨ ∗Active∗ ∨ ∗Wait∗, @F(Inmode), //PPHTrip// := Close⟩

Modes	Events	
∗ImdTrip∗ ∨ ∗Active∗ ∨ ∗Wait∗	@T(Inmode)	@F(Inmode)
//PPHTrip//	Open	Close

Finally, Table 3 shows the event table for the output data item //PPHTrip//. The event @T(Inmode) means that the system enters into the modes in the action. //PPHTrip// is assigned the value of Open when the immediate trip occurs (**ImdMode** is in *ImdTrip*) or the delayed trip occurs (**DlyMode** is in *Active* or *Wait*). Otherwise, it is assigned the value of Close.

3.2. Periodic Computation of Output Data Items and Terms

In SDS2, several output data items are used for displaying and annunciating the status of the nuclear reactor (e.g., pressure and power) and their values are updated each time unit. In our approach, these periodic functions are specified in terms of condition tables whose period is 1. For another example, consider Table 4 that shows the condition table for the term !FlinC!. The table consists of the activities $\{b_i \mid 1 \leq i \leq 4\}$ with the period 1. The modes *FlinLow*, *FlinMiddle*, and *FlinHigh* belong to a mode class **FlinMode**. The action b_1 states that "if the current mode is *FlinLow* and the condition !Flog! $<$ 2500 is true, !FlinC! := 5 is performed every unit time."

Table 4. Condition table for !FlinC!.

b_1: ⟨∗FlinLow∗, !Flog! $<$ 2500, !FlinC! := 5⟩
b_2: ⟨∗FlinLow∗, !Flog! $\geq$ 2500, !FlinC! := 200⟩
b_3: ⟨∗FlinHigh∗, true, !FlinC! := 200⟩
b_4: ⟨∗FlinMiddle∗, true !FlinC! := basedon/Flin/and/Gds/⟩
$p = 1$

Modes	Conditions		
FlinLow	!Flog! $<$ 2500	!Flog! $\geq$ 2500	—
FlinHigh	—	True	—
FlinMiddle	—	—	True
!FlinC!	5	200	based on /Flin/ and /Gds/

In general, each periodic function in real-time systems can have a different period. For example, consider the buoy controller in Booch (1986), whose requirements include: maintaining current wind, temperature, and location information such that wind speed readings are taken every 30 seconds, temperature readings every 10 seconds, and location every 10 seconds; broadcasting current wind, temperature, and location information every 60 seconds. We can specify them in terms of condition tables with different periods.

4. Overview of the Semantics

4.1. Timed Transition Systems

Transition systems (Keller, 1976; Pnueli, 1977) have been extensively used for specifying and analyzing reactive and concurrent systems. Timed transition systems (Henzinger et al., 1992; Manna and Pnueli,1993) generalize transition systems by imposing timing constraints on the transitions. A *transition system* Λ is a tuple $\langle V, \Sigma, \Theta, \mathcal{T} \rangle$ such that

- V is a set of *state variables*.
- Σ is a set of *states*. Each state $\sigma \in \Sigma$ is an interpretation of V, that is, it assigns to every variable $x \in V$ a value $\sigma(x)$ in its domain.
- $\Theta \subseteq \Sigma$ is the *initial states*.
- $\mathcal{T}$ is a set of *transitions*. Each transition $\tau \in \mathcal{T}$ is a binary relation on Σ, that is, it defines for each state $\sigma \in \Sigma$ a set of τ-successors $\tau(\sigma) \subseteq \Sigma$. We say that a transition τ is *enabled* on σ iff $\tau(\sigma) \neq \emptyset$. Otherwise, τ is *disabled* on σ.

A *timed transition system* Λ is a tuple $\langle V, \Sigma, \Theta, \mathcal{T}, l, u \rangle$ consisting of an underlying transition system $\Lambda^- = \langle V, \Sigma, \Theta, \mathcal{T} \rangle$ as well as

- A *minimal* delay $l_\tau \in \mathsf{N}$ for each transition $\tau \in \mathcal{T}$.
- A *maximal* delay $u_\tau \in \mathsf{N} \cup \{\infty\}$ for each transition $\tau \in \mathcal{T}$. It is required that $l_\tau \leq u_\tau$ for all $\tau \in \mathcal{T}$.

The semantics of timed transition systems is defined in terms of computations that represent interleaved execution of a system with timing constraints. Let T be a *clock variable*. At any point in an execution of a system, T has a value over N representing the current time. Let $\sigma_i \in \Sigma$ and t_i be an interpretation of T, for $i \geq 0$. A *computation* of a timed transition system is an infinite sequence of $\langle \sigma_0, t_0 \rangle, \langle \sigma_1, t_1 \rangle, \langle \sigma_2, t_2 \rangle, \ldots$, satisfying the following requirements:

- *Initiation*: $\sigma_0 \models \Theta$ and $t_0 = 0$.

- *Consecution*: For all $i \geq 0$,
  - either $t_i = t_{i+1}$ and $\sigma_{i+1} \in \tau(\sigma_i)$ for some transition $\tau \in \mathcal{T}$ or
  - $\sigma_i = \sigma_{i+1}$ and $t_i < t_{i+1}$. We refer to this step as a *tick step*, implying that time has progressed.
- *Lower bound*: For every transition $\tau \in \mathcal{T}$ and position $j \geq 0$, if τ is taken at j, there exists a position i, $i \leq j$, such that $t_i + l_\tau \leq t_j$ and τ is enabled on $\sigma_i, \sigma_{i+1}, \ldots, \sigma_j$.
- *Upper bound*: For every transition $\tau \in \mathcal{T}$ and position $i \geq 0$, if τ is enabled at i, there exists some position $j \geq i$ with $t_j \leq t_i + u_\tau$ such that either τ is not enabled on σ_j or τ is taken at j.
- *Progress*: As i increases, t_i grows beyond any bound.

4.2. Identifying States

The values of the entities of input data items, output data items, terms, and mode classes determine the current state of timed SCR specifications. We represent each entity in timed SCR specifications as a state variable in timed transition systems. From the set of entities $E = E_I \cup E_O \cup E_T \cup E_M$, we identify the set of state variables V as follows:

$$V = V_D \cup V_C \cup V_E, \text{ where}$$

- V_D is the set of *data variables* defined as $V_D = E_I \cup E_O \cup E_T$.
- V_C is the set of *control variables* $\{\pi_1, \pi_2, \ldots, \pi_n\}$. Each $\pi_i \in V_C$ indicates the current mode of the mode class $MC_i \in E_M$, for $1 \leq i \leq n$, and ranges over the set of modes of MC_i.
- V_E is the set of *event variables* whose type is Boolean. An event variable is associated with each mode transition and action that is immediate, i.e., its label is an event e. Intuitively, event variables are used to represent the enabledness caused by events. We will discuss event variables later in this section.

For example, we identify the set of state variables for the PPH trip parameter as follows:

- $V_D = \{$/`PHTD[i]`/, `i = 1..4`, `!FlinC!`, //`PPHTrip`//$\}$
- $V_C = \{\pi_1, \pi_2\}$, where π_1 and π_2 are associated with the mode classes `**ImdMode**` and `**DlyMode**`, respectively.
- $V_E = \{y_{m_1}, y_{m_2}, y_{m_3}, y_{m_8}\} \cup \{y_{a_1}, y_{a_2}\}$, where
  - y_{m_i} is associated with the mode transitions m_i, for $i = 1, 2, 3, 8$.
  - y_{a_i} is associated with the actions a_i, for $i = 1, 2$.

4.3. Identifying Transitions

Changes to the entities in timed SCR specifications can be classified external or internal. Changes to input data items are *external* because they are assumed to be external to the system and are not explicitly specified in timed SCR specifications. Changes of mode classes, terms, and output data items are *internal*. These changes are explicitly specified in terms of mode transitions, actions, and activities. From the changes to the entities in timed SCR specifications, we identify the set of transitions as follows:

$$\mathcal{T} = \mathcal{T}_E \cup \mathcal{T}_I, \text{ where}$$

- $\mathcal{T}_E$ is the set of *external transitions* that represent the changes of input data items. We associate an external transition with each input data item.
- $\mathcal{T}_I$ is the set of *internal transitions* that represent the changes of mode classes, terms, and output data items. We associate an internal transition τ_m, τ_a, and τ_b with each mode transition, action, and activity, respectively.

For example, we identify transitions for the PPH trip parameter as follows:

- $\mathcal{T}_E = \{\tau_{c_i} \mid 1 \leq i \leq 4\}$, where τ_{c_i} is associated with `/PHTD[i]/`, for $1 \leq i \leq 4$.
- $\mathcal{T}_I = \{\tau_{m_i} \mid 1 \leq i \leq 8\} \cup \{\tau_{a_i} \mid 1 \leq i \leq 2\}$, where
  - τ_{m_i} is associated with m_i, for $1 \leq i \leq 8$,
  - τ_{a_i} is associated with a_i, for $1 \leq i \leq 2$.

4.4. Definitions

4.4.1. Conditions

We define the meaning of a condition in timed SCR specifications in terms of a state in timed transition systems as follows:

- Let *Cond* be a predicate over the entities. If the value obtained by evaluating *Cond* using the value $\sigma(x)$ for each variable x appearing in *Cond* is true, then we say that σ *satisfies Cond* and write $\sigma \models Cond$.

  For example, if $\sigma(\texttt{!FlinC!}) = 80$, then $\sigma \models \texttt{FlinCIrr}$.
- Let *Cond* be the special condition *Inmode(me)*, where me is a mode expression. We define the relation $\models$ inductively as follows:
  - $\sigma \models Inmode(m)$ iff $\sigma(\pi_i) = m$, where $m \in MC_i$ and π_i is the control variable for MC_i.
  - $\sigma \models Inmode(\neg me)$ iff $\neg(\sigma \models Inmode(me))$.
  - $\sigma \models Inmode(me_1 \wedge me_2)$ iff $\sigma \models Inmode(me_1) \wedge \sigma \models Inmode(me_2)$.

– $\sigma \models Inmode(me_1 \vee me_2)$ iff $\sigma \models Inmode(me_1) \vee \sigma \models Inmode(me_2)$.

For example, $\sigma \models$ `Inmode(`$*$`ImdTrip` $* \vee *$ `Active`$*$`)` iff $\sigma(\pi_1) =$ $*$`ImdTrip`$*$ or $\sigma(\pi_2) =$ $*$`Active`$*$.

4.4.2. Events

We define the meaning of an event as a change between two states. Let e be an event in timed SCR specifications and σ and σ' be states in timed transition systems. We say that the change of states from σ to σ', denoted by $\langle \sigma, \sigma' \rangle$, *represents* an event @T(*Cond1*) WHEN *Cond2* if

- $\sigma \models \neg Cond1$, $\sigma' \models Cond1$,
- $\sigma \models Cond2$, $\sigma' \models Cond2$.

We say that $\langle \sigma, \sigma' \rangle$ *represents* an event @F(*Cond1*) WHEN *Cond2* if

- $\sigma \models Cond1$, $\sigma' \models \neg Cond1$,
- $\sigma \models Cond2$, $\sigma' \models Cond2$.

For example, consider two states σ_1 and σ_2 such that $\sigma_1 \models \neg$`ImdIrr` and $\sigma_2 \models$ `ImdIrr`. We say that $\langle \sigma_1, \sigma_2 \rangle$ represents the event `@T(ImdIrr)`.

4.4.3. The Enabledness Caused by Events

Among internal changes, the enabledness of immediate mode transitions and actions is determined by an occurrence of some event. For example, the mode transition m_1 in Table 1 is enabled only when the event `@T(ImdIrr)` occurs. Let σ and σ' be states in timed transition systems. We say that $\langle \sigma, \sigma' \rangle$ *enables* an immediate mode transition $\langle m, e, m' \rangle$ if

- $\langle \sigma, \sigma' \rangle$ represents e,
- $\sigma \models Inmode(m)$ and $\sigma' \models Inmode(m)$.

Similarly, we say that $\langle \sigma, \sigma' \rangle$ *enables* an immediate action $\langle me, e, x := v \rangle$ if

- $\langle \sigma, \sigma' \rangle$ represents e,
- $\sigma \models Inmode(me)$ and $\sigma' \models Inmode(me)$.

For example, consider σ_1 and σ_2 such that $\sigma_1 \models \neg$`ImdIrr`, $\sigma_2 \models$ `ImdIrr`, and $\sigma_1(\pi_1) = \sigma_2(\pi_1) =$ $*$`ImdNotTrip`$*$. We say that $\langle \sigma_1, \sigma_2 \rangle$ enables the mode transition m_1 because

Table 5. Non-deterministic mode transition table.

	Current mode	Label	Next mode
m_1:	M_1	e_1: @T(x = True)	M_2
m_2:	M_1	e_2: @T(y = True)	M_3

$\langle \sigma_1, \sigma_2 \rangle$ represents `@T(ImdIrr)` while maintaining the current mode as `*ImdNotTrip*` in σ_1 and σ_2.

Since two states are involved in an event, we model an event as a change between two states rather than taking it as a primitive concept. Therefore, the enabledness caused by events are also determined in terms of two states. However, since transitions in timed transition systems are binary relations on the set of states Σ, we should be able to define the enabledness of transitions in terms of one state rather than the changes between states. That is, in every state the enabledness of transitions should be determined without referring to other states. Hence, we cannot directly apply the above definition to determine the enabledness of transitions associated with immediate mode transitions and actions.

We cope with this problem using additional state variables, called *event variables*. Let m be an immediate mode transition and τ_m be the internal transition associated with m. We associate an event variable y_m with m and require that y_m becomes true in σ' if $\langle \sigma, \sigma' \rangle$ enables m. In this way, we can determine the enabledness of τ_m caused by events in terms of one state by referring to the value of y_m in that state. Similarly, we associate an event variable with each immediate action. Each event variable has the initial value of false, which means that no event has occurred before the system initialization. Notice that mode transitions and events are notions of timed SCR specifications, while transitions and event variables are those of timed transition systems.

Finally, we define the disabledness caused by events. We say that $\langle \sigma, \sigma' \rangle$ *disables* an immediate mode transition $\langle m, e, m' \rangle$ if

- $\sigma \models \mathit{Inmode}(m)$ and $\sigma' \models \neg \mathit{Inmode}(m)$.

Similarly, we say that $\langle \sigma, \sigma' \rangle$ *disables* an immediate action $\langle me, e, x := v \rangle$ if

- $\sigma \models \mathit{Inmode}(me)$ and $\sigma' \models \neg \mathit{Inmode}(me)$.

We require that y_i becomes false in σ' if $\langle \sigma, \sigma' \rangle$ disables τ_i. For example, let y_1 and y_2 be the event variables associated with the mode transitions m_1 and m_2 in Table 5. Suppose the current mode is M_1 and the events e_1 and e_2 occur simultaneously. Then both m_1 and m_2 are enabled and hence we change the values of y_1 and y_2 into true. When one of the mode transitions occurs, say m_1, m_2 is disabled and we change the value of y_2 into false, which means that the event e_2 is ignored.

4.4.4. The Special Condition Inmode

In event tables, the special condition *Inmode* can be used as the triggering condition of an event (e.g., in the actions a_1 and a_2 in Table 3). For this case, we change the definition of the enabledness of actions as follows: We say that $\langle \sigma, \sigma' \rangle$ *enables* an immediate action $\langle me, e, x := v \rangle$ if

- when e is the form of @T(*Inmode*) WHEN *Cond2*,
  - $\sigma \models \neg Inmode(me)$, $\sigma' \models Inmode(me)$,
  - $\sigma \models Cond2$, $\sigma' \models Cond2$.
- when e is the form of @F(*Inmode*) WHEN *Cond2*,
  - $\sigma \models Inmode(me)$, $\sigma' \models \neg Inmode(me)$,
  - $\sigma \models Cond2$, $\sigma' \models Cond2$.

For example, consider σ_1 and σ_2 such that $\sigma_1(\pi_1) = *\texttt{ImdNotTrip}*$, $\sigma_2(\pi_1) = *\texttt{ImdTrip}*$, and $\sigma_1(\pi_2) = \sigma_2(\pi_2) = *\texttt{Idle}*$. We say that $\langle \sigma_1, \sigma_2 \rangle$ enables the action a_1, because $\sigma_1 \models \neg Inmode(*\texttt{ImdTrip}* \vee *\texttt{Active}* \vee *\texttt{Wait}*)$ and $\sigma_2 \models Inmode(*\texttt{ImdTrip}* \vee *\texttt{Active}* \vee *\texttt{Wait}*)$.

5. The Behavior of Tables in Timed SCR Specifications

In this section, we show how each table is translated into transitions and describe properties of the resulting transitions in terms of pre-condition, post-condition, and time interval. For all states σ and σ' such that $\sigma' \in \tau(\sigma)$, the pre and post condition of τ are predicates that should be satisfied in σ and σ', respectively.

5.1. External Transitions

In real-time systems, the quantities of the environment which are represented by input data items can be either continuous (e.g., pressure) or discrete (e.g., push-buttons). When quantities are continuous, they are modeled via real-valued input data items. The values of these input data items are expected to change with every time tick, since continuous quantities change synchronously with time in a time-driven way. In contrast, the values of discrete quantities change only at certain instant of time asynchronously in an event-driven way.

Let c be an input data item in timed SCR specifications that represents a continuous quantity. To represent the changes of c, we associate an external transition τ_c with c. The

$$\cdots \longrightarrow \langle \sigma_1, t \rangle \xrightarrow{\tau_{c_1}} \langle \sigma_2, t \rangle \longrightarrow \cdots$$

	σ_1	σ_2
`PHTD[1]`	4117	4119
`ImdIrr`	false	true
π_1	`*ImdNotTrip*`	`*ImdNotTrip*`
y_{m_1}	false	true

Figure 1. A computation for the occurrence of τ_{c_1}.

transition τ_c has the lower and upper bounds of [1, 1] and satisfies that:

$\sigma' \in \tau_c(\sigma)$ iff

1. $\sigma'(x) = \sigma(x)$ for all $x \in V - (\{c\} \cup V_E)$.

2. for each event variable $y_i \in V_E$,

   if $\langle \sigma, \sigma' \rangle$ enables τ_i[1], $\sigma'(y_i) =$ true,
   else if $\langle \sigma, \sigma' \rangle$ disables τ_i, $\sigma'(y_i) =$ false,
   otherwise, $\sigma'(y_i) = \sigma(y_i)$.

The pre-condition for τ_c is true, i.e., τ_c is enabled at all times. The following constitutes the post-condition for τ_c. The input data item c can be assigned any value in the state σ', because τ_c does not put any restriction on c. Variables except c and event variables retain their values over τ_c (item 1). Certain internal transitions can be enabled or disabled by the occurrence of τ_c. If τ_i is enabled (or disabled) by $\langle \sigma, \sigma' \rangle$, we require that the value of its event variable y_i is set as true (false, resp.) in σ' (item 2). Finally, the time interval [1, 1] means that the values of c should be changed with every time tick. For example, Figure 1 shows a computation that represents the occurrence of τ_{c_1} associated with the input data item `/PHTD[1]/`. When τ_{c_1} occurs, the value of `/PHTD[1]/` is changed from 4117 to 4119 and hence $\langle \sigma_1, \sigma_2 \rangle$ represents the event `@T(ImdIrr)`. In addition, since the current mode is maintained as `*ImdNotTrip*` in σ_1 and σ_2, $\langle \sigma_1, \sigma_2 \rangle$ enables the mode transition τ_{m_1} and thus τ_{c_1} changes the value of y_{m_1} into true.

Similarly, we associate an external transition τ_d with each input data item d that represents a discrete quantity. The only difference between τ_c and τ_d is the time interval. The values of d are changed asynchronously only at certain instant of time and hence we define the time interval of τ_d as $[1, \infty]$.

5.2. Internal Transitions

5.2.1. Mode Transitions

We associate an internal transition τ_m with each immediate mode transition $\langle m, e, m' \rangle$. Let y_m be the event variable for τ_m. The transition τ_m has the lower and upper bound of [0, 0] and satisfies that:

$\sigma' \in \tau_m(\sigma)$ iff

1. $\sigma \models \mathit{Inmode}(\{m\})$,
2. $\sigma(y_m) = \text{true}$,
3. $\sigma' \models \mathit{Inmode}(\{m'\})$,
4. $\sigma'(x) = \sigma(x)$ for all $x \in V - (\{\pi_i\} \cup V_E)$, where $m, m' \in MC_i$ and π_i is the control variable for MC_i.
5. $\sigma'(y_m) = \text{false}$,
6. for each event variable $y_i \in V_E$, $y_i \neq y_m$,

   if $\langle \sigma, \sigma' \rangle$ enables τ_i, $\sigma'(y_i) = \text{true}$,
   else if $\langle \sigma, \sigma' \rangle$ disables τ_i, $\sigma'(y_i) = \text{false}$,
   else $\sigma'(y_i) = \sigma(y_i)$.

The first and second items constitute the pre-condition for τ_m. That is, τ_m can occur if the current mode is m and its triggering event has occurred. The post-condition for τ_m is defined by the other items: The current mode is changed into m' (item 3). All the variables except π_i and event variables retain their values over τ_m (item 4). "$\sigma'(y_m) = \text{false}$" enforces that a transition should be executed only one time by the occurrence of its triggering event. Without this condition, the semantics allows two or more executions of a transition by one occurrence of its triggering event. (item 5). The transition τ_m can enable or disable other internal transitions (item 6). The time interval [0, 0] enforces that τ_m should occur immediately when its triggering event occurs. Figure 2 shows an occurrence of the mode transition m_1 in Table 1. The external transition τ_{c_1} represents the triggering event `@T(ImdIrr)` for m_1 and the internal transition τ_{m_1} represents the internal change of the mode class `**ImdMode**` from `*IMdNotTrip*` to `*ImdTrip*`.

We associate an internal transition τ_m with each non-immediate mode transition $\langle m, c[l, u], m' \rangle$. The transition τ_m has the lower and upper bound of $[l, u]$ and satisfies that:

$\sigma' \in \tau_m(\sigma)$ iff

1. $\sigma \models \mathit{Inmode}(\{m\}) \wedge c$,
2. $\sigma' \models \mathit{Inmode}(\{m'\})$,

$$\cdots \longrightarrow \langle \sigma_1, t \rangle \xrightarrow{\tau_{c_1}} \langle \sigma_2, t \rangle \xrightarrow{\tau_{m_1}} \langle \sigma_3, t \rangle \longrightarrow \cdots$$

	σ_1	σ_2	σ_3
`PHTD[1]`	4117	4119	4119
`ImdIrr`	false	true	true
y_{m_1}	false	true	false
π_1	`*ImdNotTrip*`	`*ImdNotTrip*`	`*ImdTrip*`

Figure 2. A computation for the mode transition m_1.

3. $\sigma'(x) = \sigma(x)$ for all $x \in V - (\{\pi_i\} \cup V_E)$, where $m, m' \in MC_i$ and π_i is the control variable for MC_i.

4. for each event variable $y_i \in V_E$,

   if $\langle \sigma, \sigma' \rangle$ enables τ_i, $\sigma'(y_i) = \text{true}$,
   else if $\langle \sigma, \sigma' \rangle$ disables τ_i, $\sigma'(y_i) = \text{false}$,
   else $\sigma'(y_i) = \sigma(y_i)$.

The pre-condition (the first item) and the time bound $[l, u]$ enforce that the condition "*Inmode*$(\{m\}) \wedge c$" has been continuously satisfied between l and u time units. Because the enabledness can be defined in terms of conditions, we do not need to use event variables. Figure 3 shows an occurrence of the mode transition m_4 in Table 2.

5.2.2. Actions

We associate an internal transition τ_a with each immediate action $\langle me, e, x := v \rangle$. Let y_a be the event variable for τ_a. The transition τ_a has the lower and upper bound of [0, 0] and

$$\cdots \longrightarrow \langle \sigma_1, t_1 \rangle \xrightarrow{tick} \langle \sigma_2, t_2 \rangle \xrightarrow{\tau_{m_4}} \langle \sigma_3, t_2 \rangle \longrightarrow \cdots$$

	σ_1	σ_2	σ_3
π_2	`*Pend*`	`*Pend*`	`*Decide*`

$$t_1 + \texttt{PPHDlyT} - \texttt{PPHDltTol} \leq t_2 \leq t_1 + \texttt{PPHDlyT} + \texttt{PPHDltTol}$$

Figure 3. A computation for the mode transition m_4.

$$\cdots \longrightarrow \langle \sigma_1, t \rangle \xrightarrow{\tau_{m_1}} \langle \sigma_2, t \rangle \xrightarrow{\tau_{a_1}} \langle \sigma_3, t \rangle \longrightarrow \cdots$$

	σ_1	σ_2	σ_3
π_1	`*ImdNotTrip*`	`*ImdTrip*`	`*ImdTrip*`
π_2	`*Idle*`	`*Idle*`	`*Idle*`
y_{m_1}	true	false	false
y_{a_1}	false	true	false
`//PPHTrip//`	`Close`	`Close`	`Open`

Figure 4. A computation for the action a_1.

satisfies that:

$\sigma' \in \tau_a(\sigma)$ iff

1. $\sigma \models \mathit{Inmode}(me)$,
2. $\sigma(y_a) = \text{true}$,
3. $\sigma'(x) = \sigma(v)$,
4. $\sigma'(x') = \sigma(x')$ for all $x' \in V - (\{x\} \cup V_E)$,
5. $\sigma'(y_a) = \text{false}$,
6. for each event variable $y_i \in V_E$, $y_i \neq y_a$,

   if $\langle \sigma, \sigma' \rangle$ enables τ_i, $\sigma'(y_i) = \text{true}$,
   else if $\langle \sigma, \sigma' \rangle$ disables τ_i, $\sigma'(y_i) = \text{false}$,
   else $\sigma'(y_i) = \sigma(y_i)$.

The pre-condition for τ_a consists of the first and second items. The third item means that when τ_a occurs, x is assigned the value of v in the state σ'. The other items can be described in a similar way as mode transitions. Figure 4 shows an occurrence of the action a_1 in Table 3. The internal transition τ_{m_1} enables τ_{a_1}, i.e., $\sigma_1 \models \neg \mathit{Inmode}(*\texttt{ImdTrip}* \vee *\texttt{Active}* \vee *\texttt{Wait}*)$ and $\sigma_2 \models \mathit{Inmode}(*\texttt{ImdTrip}* \vee *\texttt{Active}* \vee *\texttt{Wait}*)$. The internal transition τ_{a_1} represents the internal change of the output data item `//PPHTrip//`.

We associate an internal transition τ_a with each non-immediate action $\langle me, c[l, u], x := v \rangle$. The transition τ_a has the lower and upper bound of $[l, u]$ and satisfies that:

$\sigma' \in \tau_a(\sigma)$ iff

1. $\sigma \models \mathit{Inmode}(me) \wedge c$,
2. $\sigma'(x) = \sigma(v)$,

3. $\sigma'(x') = \sigma(x')$ for all $x' \in V - (\{x\} \cup V_E)$,

4. for each event variable $y_i \in V_E$, $y_i \neq y_a$,

   if $\langle \sigma, \sigma' \rangle$ enables τ_i, $\sigma'(y_i) =$ true,
   else if $\langle \sigma, \sigma' \rangle$ disables τ_i, $\sigma'(y_i) =$ false,
   else $\sigma'(y_i) = \sigma(y_i)$.

The internal transition τ_a can be described in a similar way as τ_m for the mode transition $\langle m, c[l, u], m' \rangle$.

5.2.3. *Activities*

Consider a condition table $\{\langle me_i, c_i, x := v_i \rangle \mid 1 \leq i \leq n\}$ with the period p. We associate an internal transition τ_b with each condition table. The transition τ_b has the lower and upper bound of $[p, p]$ and satisfies that:

$\sigma' \in \tau_b(\sigma)$ iff

1. $\sigma'(x) = \begin{cases} \sigma(v_1) & \text{if } \sigma \models \mathit{Inmode}(me_1) \wedge c_1 \\ \sigma(v_2) & \text{if } \sigma \models \mathit{Inmode}(me_2) \wedge c_2 \\ \cdots & \\ \sigma(v_n) & \text{if } \sigma \models \mathit{Inmode}(me_n) \wedge c_n \end{cases}$

2. $\sigma'(x') = \sigma(x')$ for all $x' \in V - (\{x\} \cup V_E)$,

3. for each event variable $y_i \in V_E$,

   if $\langle \sigma, \sigma' \rangle$ enables τ_i, $\sigma'(y_i) =$ true,
   else if $\langle \sigma, \sigma' \rangle$ disables τ_i, $\sigma'(y_i) =$ false,
   else $\sigma'(y_i) = \sigma(y_i)$.

The first item represents the mathematical relation that is described by the condition table. The time interval enforces that τ_b occurs periodically every p time unit. For example, suppose that we associate an internal transition τ_b with Table 4 and a control variable π_3 with the mode class `**FlinMode**`. At each moment that τ_b is executed, one of the four activities in Table 4 is selected based on the current mode of `**FlinMode**` and the value of `!Flog!`[2].

6. The Global Behavior of Timed SCR Specifications

When the translation is completed, the global behavior of timed SCR specifications can be defined in terms of computations of the resulting timed transition systems. Therefore, our semantics is based on interleaving models that sequentialize simultaneous transitions in an arbitrary order so that at most one transition has to be analyzed at any instant of time. When defining the global behavior, the advantage of an interleaving semantics becomes obvious.

$$\cdots \langle \sigma_1, t_1 \rangle \xrightarrow{tick} \langle \sigma_2, t_2 \rangle \xrightarrow{\tau_b} \langle \sigma_3, t_3 \rangle \xrightarrow{tick} \langle \sigma_4, t_4 \rangle \xrightarrow{\tau_b} \langle \sigma_5, t_5 \rangle \cdots$$

	σ_1	σ_2	σ_3	σ_4	σ_5
π_3	`*FL*`	`*FL*`	`*FM*`	`*FM*`	`*FM*`
`!Flog!`	2450	2450	2451	2451	2452
`!FlinC!`	5	5	7	7	8

$t_2 = t_1 + 1$, $t_3 = t_2$, $t_4 = t_3 + 1$, and $t_5 = t_4$

Figure 5. A computation for Table 4.

Without interleaving, we should consider a global transition for every pair of transitions that may be taken simultaneously. The size of reachability graphs can be reduced by interleaving, because we do not need to consider the simultaneous occurrence of transitions. In the timed case, the composition cannot even be defined in this manner, because a single minimal delay and a single maximal delay would have to be assigned to the global transitions (Alur and Henzinger, 1992).

There exist several requirements that should be satisfied in all the computations to guarantee that the computations represent only the proper behavior of a system. One of the main requirements for specification languages of reactive and real-time systems is the *synchrony hypothesis* of Berry (Berry and Gonthier, 1992). This hypothesis assumes that the system is infinitely faster than the environment and hence the response to an external event is always generated at the same time that the event is introduced. In our approach, transitions in timed transition systems are classified into two types: external and internal. To enforce synchrony hypothesis, we require that at each instant of time internal transitions can occur only after all the external transitions have occurred. Formally, for any computation

$$\cdots \xrightarrow{tick} \langle \sigma_i, t \rangle \xrightarrow{\tau_i} \langle \sigma_{i+1}, t \rangle \xrightarrow{\tau_{i+1}} \cdots \xrightarrow{\tau_{j-1}} \langle \sigma_j, t \rangle \xrightarrow{tick} \cdots$$

there exists k, $i \leq k \leq j$, such that each transition τ_l, $i \leq l \leq k$, is an external transition and each transition τ_l, $k < l < j$, is an internal transition.

The computations should be further restricted by the assumptions that constrain the changes of input data items. We can represent these assumptions by strengthening the pre and post condition of external transitions or changing the time interval. In Atlee and Gannon (1993) and Atlee and Buckley (1996), a set of *environmental assumptions* is proposed which describe the relationships between input data items. For example, let $F1$, $F2$, and $F3$ be conditions defined as $/\texttt{Flin}/ \leq \texttt{450}$, $\texttt{450} < /\texttt{Flin}/ < \texttt{4500}$, and $/\texttt{Flin}/ \geq \texttt{4500}$, respectively. The environmental assumption between $F1$, $F2$, and $F3$ is that there is no computation

$$\cdots \langle \sigma, t \rangle \longrightarrow \langle \sigma', t \rangle \cdots$$

such that $\sigma \models F1$ and $\sigma' \models F3$, or $\sigma \models F3$ and $\sigma' \models F1$. Suppose that we associate an external transition τ_f with the input data item `/Flin/`. We incorporate the environmental

assumption into the definition of τ_f by strengthening the pre and post-condition of τ_f as follows:

- if $\sigma \models F1, \sigma' \models F2$,
- if $\sigma \models F2, \sigma' \models F1 \vee F3$,
- if $\sigma \models F3, \sigma' \models F2$.

Another interesting assumption is the *separation parameter* which specifies a lower and upper bound on the length of an interval separating two successive occurrences of external events (Jahanian and Mok, 1986). We can simply represent this assumption by changing the time interval of an external transition.

The global behavior can be also restricted by the assumption in Heitmeyer et al. (1996), called *one input assumption*, which states that exactly one external event occurs at any instant of time. The analysis of timed SCR specifications can be made easier by the assumption, because we can ignore the occurrences of simultaneous external events when defining the global behavior. To enforce one input assumption, we require that if a set of external transitions occur at one instant of time, at most one external can enable internal transitions and other transitions do not affect the enabledness of internal transitions.

7. Conclusion and Discussion

We have presented an approach to extend the SCR method for real-time systems and proposed the syntax and semantics of timed SCR specifications. The syntax is defined by extending the tabular notation of the SCR method to deal with two typical real-time constraints: sporadic and periodic timing constraints. The semantics is based on a translational approach which shows how to identify each of the components of timed transition systems from timed SCR specifications. By the translation, we can formally define the global behavior as well as the meaning of each construct in timed SCR specifications. A main issue in the translation is that timed transition systems are a state-based formalism and do not provide a natural modeling of events. We have showed how to represent events and the enabledness caused by events using additional state variables.

7.1. Comparison to Related Works

In Alspaugh et al. (1992) and van Schouwen (1990), a semantics for each table in the SCR method is proposed while the global behavior is not explicitly discussed. In Alspaugh et al. (1992), three definitions for an event occurrence are proposed. If @T($C1$) occurs at time t, then @T($C1$) WHEN $C2$ occurs at time t if there exists $\epsilon > 0$ such that $C2$ is true throughout the time interval $[t-\epsilon, t]$, $[t, t+\epsilon]$, or $[t-\epsilon, t+\epsilon]$. In van Schouwen (1990), the semantics of events is defined as follows: there exists $\epsilon > 0$ such that $C1$ is false in $[t-\epsilon, t)$, $C2$ is true in $[t-\epsilon, t)$, and $C1$ is true at t. A drawback of these approaches is that two or more transitions cannot occur simultaneously due to the artificial delay ϵ. When

defining the global behavior, there exists some arbitrary amount of time between any two transitions. This delay makes it difficult or even impossible to analyze system properties.

Atlee and Buckley (1996) proposed a semantics for a subclass of SCR specifications which contain only mode transition tables. In their approach, each mode transition is represented as a temporal formula and the global behavior is defined as logical conjunction of each mode transition's formula. Heitmeyer et al. (1996) model SCR specifications as a state machine whose transition relation is a function that maps the current state and an input event into a new state. In their approach, each table is represented by a mathematical function and the global behavior is obtained by the composition of smaller functions which are derived from each table.

We compare our work to the semantics in Atlee and Buckley (1996) and Heitmeyer et al. (1996) as follows. The most significant difference is that we focus on quantitative real-time constraints while the works in Atlee and Buckley (1996) and Heitmeyer et al. (1996) focus on untimed system behavior of SCR specifications. It is not straightforward to incorporate time into SCR specifications when using functional composition or logical conjunction, while our approach provides a natural modeling of time because we represent each table as transitions in timed transition systems and define the global behavior as computations which represent interleaved execution of transitions with timing constraints. Second, we allow both deterministic and non-deterministic behavior while the non-deterministic behavior cannot be allowed in Atlee and Buckley (1996) and Heitmeyer et al. (1996). For example, Table 5 is regarded as inconsistent and cannot be given a semantics by functional composition in Heitmeyer et al. (1996), because the events e_1 and e_2 can occur simultaneously. The approach in Atlee and Buckley (1996) cannot also define a semantics for Table 5. The mode transition m_1 is represented by a formula $f_1 = M_1 \wedge \neg x \wedge n(x) \rightarrow n(M2)$, where n is the next-state operator. Similarly, m_2 is represented by $f_2 = M_1 \wedge \neg y \wedge n(y) \rightarrow n(M3)$. Logical conjunction of f_1 and f_2 leads to a contradiction and the table cannot be given a semantics by logical conjunction. Finally, we discuss how the occurrence of simultaneous events are handled. In Heitmeyer et al. (1996), Heitmeyer et al. use one input assumption in which the events that occur simultaneously are not considered. The assumption makes the analysis of SCR specifications easier but might be restrictive because there exist cases in which we must explicitly model the effect of two events occurring simultaneously. In Atlee and Buckley (1996), all the simultaneous occurrences of mode transitions are considered and a global transition should be generated for every pair of mode transitions that occur simultaneously. In contrast, our semantics is based on interleaving models which allow efficient methods for defining the global behavior while considering all the simultaneous events.

7.2. Verification

We are currently working on how to apply the verification methods in the concurrency literature such as temporal proof and model checking methods to timed SCR specifications. First, our semantics enables us to directly apply the temporal proof methodologies of Henzinger, Manna, and Pnueli, (1994) to timed SCR specifications. We can verify temporal formulas against timed SCR specifications by applying the theorem proving rules in Henzinger et al.

(1994) to the timed transition systems which are translated from timed SCR specifications. The only necessary extension is to incorporate the occurrence of events into temporal formulas, because there is no notion of events in timed transition systems. For example, the following formula states that "after the current mode is changed into `*Active*`, the value of `//PPHTrip//` should be maintained as `Trip` during at least 0.9 seconds."

$$\texttt{@T(Inmode(*Active*))} \rightarrow \Box_{\leq 0.9}(\texttt{//PPHTrip//} = \texttt{Trip})$$

The truth of an event e appearing in temporal formula is defined as follows: we say that a state σ_{i+1} in a computation

$$\cdots \langle \sigma_i, t_i \rangle \longrightarrow \langle \sigma_{i+1}, t_{i+1} \rangle \cdots$$

satisfies the event e if $\langle \sigma_i, \sigma_{i+1} \rangle$ represents e.

Several model checking methods have been proposed for SCR specifications such as Atlee and Gannon (1993), Atlee and Buckley (1996), and Bharadwaj and Heitmeyer (1997). Because the semantics in Atlee and Gannon (1993), Atlee and Buckley (1996), and Bharadwaj and Heitmeyer (1997) are significantly different to our semantics and do not consider quantitative timing properties, we need a new model checking method for timed SCR specifications. We are currently developing a method that generates a form suitable for input to model checkers from timed SCR specifications. We expect that our semantics would be of use when developing a model checking method for timed SCR specifications, because both timed transition systems and the structures used in model checkers are state-based formalisms in the sense that they consist of only states and transitions and they do not provide no natural modeling of events. In particular, event variables used to represent the enabledness caused by events can also be used in model checking.

Acknowledgement

This work was supported by the Korea Science and Engineering Foundation (KOSEF) through the Advanced Information Technology Research Center (AITrc).

Notes

1. For notational convenience, we say that $\langle \sigma, \sigma' \rangle$ enables τ_i if τ_i is associated with an immediate mode transition or an immediate action which is enabled by $\langle \sigma, \sigma' \rangle$.
2. For notational convenience, we abbreviate `*FlinLow*`, `*FlinMiddle*`, and `*FlinHigh*` as `*FL*`, `*FM*`, and `*FH*`.

References

AECL CANDU. 1993. Program functional specification, SDS2 programmable digital comparators. Wolsung Nuclear Power Plant, 86-68330-PFS-000, May.

Alspaugh, T. A., Faulk, S. R., Britton, K. H., Parker, R. A., Parnas, D. L., and Shore, J. E. 1992. Software requirements for the A-7E aircraft. Technical Report NRL-9194, Naval Research Laboratory, Washington, DC.

Alur, R., and Henzinger, T. A. 1992. Logics and models of real time: A survey. *Proceedings of Real Time: Theory in Practice*. Lecture Notes in Computer Science, Vol. 600, Springer-Verlag, pp. 74–106.

Atlee, J. M., and Gannon, J. 1993. State-based model checking of event-driven system requirements. *IEEE Transactions on Software Engineering* 19(1): 24–40.

Atlee, J. M., and Buckley, M. A. 1996. A logic-model semantics for SCR software requirements. In *Proceedings of the 1996 International Symposium on Software Testing and Analysis*, pp. 280–292.

Berry, G., and Gonthier, G. 1992. The ESTEREL synchronous programming languages: Design, semantics, implementation. *Science of Computer Programming* 19: 87–152.

Bharadwaj, R., and Heitmeyer, C. L. 1997. Verifying SCR requirements specifications using state exploration. *Proceedings of First ACM SIGPLAN Workshop on Automatic Analysis of Software*.

Booch, G. 1986. Object-oriented developement. *IEEE Transactions on Software Engineering* 12(2): 211–221.

Courtois, P. -J., and Parnas, D. L. 1993. Documentation for safety critical software. *Proceedings of 15th International Conference on Software Engineering*, pp. 315–323.

Faulk, S. R., Brackett, J., Ward, P., and Kirby, J. 1992. The CoRE method for real-time requirements. *IEEE Software* 9(5): 22–33.

Faulk, S. R., Finneran, L., Kirby, J., Shah, J. S., and Sutton, J. 1994. Experience with applying the CoRE method to the Lockheed C-130J software requirements. *Proceedings of the 9th Annual Conference on Computer Assurance*, pp. 3–8.

Heitmeyer, C. L., Bull, A., Gasarch, C., and Labaw, B. G. 1995. SCR*: A toolset of for specifying and analyzing requirements. *Proceedings of the 10th Annual Conference on Computer Assurance*, pp. 109–122.

Gomma, H. 1993. *Software Design Methods for Concurrent and Real-Time Systems*. Addison-Wesley Publishing Company.

Harel, D. 1987. Statecharts: A visual formalism for complex systems. *Science of Computer Programming* 8: 231–274.

Heitmeyer, C. L., Jeffords, R. D., and Labaw, B. G. 1996. Automated consistency checking of requirements specifications. *ACM Transactions of Software Engineering and Methodology* 5(3): 231–261.

Heninger, K. L. 1980. Specifying software requirements for complex cystems: new techniques and their applications. *IEEE Transactions on Software Engineering* 6(1): 2–13.

Henzinger, T. A., Manna, Z., and Pnueli, A. 1992. Timed transitions systems. *Proceedings of REX Workshop on Real-Time: Theory in Practice*. Lecture Notes in Computer Science, Vol. 600, Springer-Verlag, pp. 226–251.

Henzinger, T. A., Manna, Z., and Pnueli, A. 1994. Temporal proof methodologies for timed transitions systems. *Information and Computation* 112(2): 273–337.

Jahanian, F., and Mok, A. K. 1986. Safety analysis of timing properties in real-time systems. *IEEE Transactions on Software Engineering* 12(9): 890–904.

Jahanian, F., and Mok, A. K. 1994. Modechart: A specification language for real-time systems. *IEEE Transactions on Software Engineering* 20(12): 933–947.

Keller, R. M. 1976. Formal verification of parallel programs. *Communication of the ACM* 19(7): 371–384.

Kesten, Y., and Pnueli, A. 1992. Timed and hybrid statecharts and their textual representation. *Proceedings of Formal Techniques in Real-Time and Fault-Tolerant Systems*. Lecture Notes in Computer Science, Springer-Verlag, Vol. 571, pp. 591–620.

Manna, Z., and Pnueli, A. 1989. The anchored version of the temporal framework. *Proceedings of Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency*. Lecture Notes in Computer Science. Springer-Verlag, Vol. 354, pp. 201–284.

Manna, Z., and Pnueli, A. 1993. Models for reactivity. *Acta Informatica* 30: 609–678.

Meyer, B. 1990. *Introduction to the Theory of Programming Languages*. Prentice Hall.

Parnas, D. L., and Madey, J. 1990. Functional documentation for computer systems engineering. Technical Report 90-287, TRIO, Queen's University, Ontario, Canada.

Petersohn, C., and Urbina, L. 1997. A timed semantics for the STATEMATE implementation of statecharts. *FME '97: Industrial Applications and Strengthened Foundations of Formal Methods*. Lecture Notes in Computer Science. Springer-Verlag, Vol. 1313, pp. 553–572.

Pnueli, A. 1977. The temporal logic of programs. *Proceedings of the 18th Annual Symposium on Foundations of Computer Software*, pp. 46–57.

van Schouwen, A. J. 1990. The A-7 requirements model: Re-examination for real-time systems and an application to monitoring systems. Technical Report 90-276, TRIO, Queen's University, Ontario, Canada.

van Schouwen, A. J., Parnas, D. L., and Madey, J. 1993. Documentation of requirements for computer systems. *Proceedings of RE'93: IEEE International Symposium on Requirements Engineering*, pp. 198–207.