# 5 FORMAL SPECIFICATION OF TEMPORAL CONSTRAINTS

"And then the clock collected in the tower/
Its strength and struck."

[A. E. Housman, Eight O'Clock, Last Poems, 1922]

## 5.1 Introduction

In this chapter the formal resources employed in our approach for specifying temporal aspects of TCS are presented. Because various sorts of TCS behaviour can be described using the archetypal constraints identified some sixteen years ago by Dasarathy, we start by reviewing this author's classification [Dasarathy85], wrapping the original classes of constraints in the garments of a simple notation introduced for manipulation purposes. These classes of constraints will be used later (in Chapter 8) to illustrate our approach for capturing temporal properties of systems. Then, we emphasise the need for formality in describing timing properties of the systems and, because our Z++ formalism partially relies on Jahanian and Mok's Real-Time Logic (RTL) and its underlying event-action model [Jahanian86], we briefly survey the model and the RTL notation. Finally, we present the extensions proposed by Lano for employing RTL within the frame of Z++ [Lano95]. Throughout the chapter the concepts and notations are illustrated by short examples of daily-life extraction.

## 5.2 Dasarathy's Classification of Temporal Constraints

Since our modelling approach is aimed at TCS, particular attention is paid to specifying temporal restrictions placed on such systems. Dasarathy's landmark paper [Dasarathy85] on constructs for expressing timing constraints of RTS provides the reference for our way of dealing with time. The original classification introduced by Dasarathy was widely accepted by the researchers in the field because it covers in a simple yet extensive manner the various types of temporal constraints that can be imposed on systems. The basic notions on which the classification was built are those of stimulus (S), response (R), and event (E). The latter, as indicated by the author, can be either a stimulus received by the system from its environment or an externally observable response issued by the system.

However, in order to unify the terminology and subsequently make the transition to the event-action model underlying Jahanian and Mok's RTL [Jahanian86], we had to make some alterations to the original concepts of Dasarathy. Specifically, following Jahanian and Mok's approach and as opposed to Dasarathy's, we consider the events instantaneous and make use of the additional notion of action to describe an operation that has a non-zero duration (details about actions and events are given in Subsection 5.4.1). Consequently, the duration class of timing constraints identified by Dasarathy will no longer apply to events, but to actions, because in our approach events have no duration. This has however only a minor impact on the original classification of Dasarathy, since it affects only one of the nine classes of contraints ("classes of constraints" is our terminology). And, as in the original Dasarathy paper, both stimuli and responses continue to be considered events.

In the following, Dasarathy's classification of timing constraints, presented in our own notation, is briefly reviewed. The examples given by Dasarathy for the classes he proposed were from the field of telephony; in this section, we employ a microwave oven device to provide short illustrations for each class of temporal constraints. The classes of constraints are given in an informal manner, some implicit assumptions being made about the occurrences

of stimuli and responses. However, as discussed in the next section, a more rigorous specification of the constraints is necessary for developing reliable models of TCS.

Before reviewing the possible types of temporal constraints it is useful to note that, as Dasarathy points out, the timing restrictions placed on a system can be either performance constraints, which impose limits on the system's response time, or behavioural constraints, which specify restrictions on the rates of stimuli applied to the system. Both types of constraints can be described using three broad categories of timing constraints: maximum, minimum, and durational. A constraint of type maximum specifies an upper limit placed on the interval of time between two occurrences of events, a constraint of type minimum specifies a lower limit for the interval between two such occurrences, and a durational constraint indicates the amount of time required for the duration of an action. These three categories of temporal restrictions are not exclusive, in a more complex case being possible to have constraints of all three kinds placed on a particular behaviour of the system.

When possible combinations involving stimuli and responses come into consideration, the maximum and minimum categories expand in four subcategories (or classes) each. Because the duration category needs no further partitioning, a total of nine classes of temporal constraints are hence possible (the notation DCx means "Dasarathy contraint class x", where x is a number we provide for easier referencing):

[DC1] MaxSS($S_1$, $S_2$, t), specifies the maximum time t allowed between the occurrence of stimulus $S_1$ and the occurrence of the subsequent stimulus $S_2$.

> Example: After the power level has been set, the microwave oven's start button should be pressed no later than 60 seconds, otherwise the attempt to use the heating feature of the microwave oven will be considered abandoned. This timing condition can be expressed as MaxSS(setPowerLevelCmd, startHeatingCmd, 60.0).
>
> (Note that for documentation purposes the Cmd postfix is used in this chapter to indicate a stimulus event, as opposed to a response event, which has no specific postfix);

[DC2] MinSS($S_1$, $S_2$, t), specifies the minimum time t allowed between the occurrence of stimulus $S_1$ and the occurrence of the subsequent stimulus $S_2$.

Example: After the current date and time has been set, the microwave oven's heating feature should not be started for at least 1 second. This can be expressed as MinSS(setDateTimeCmd, startHeatingCmd, 1.0);

[DC3] MaxRS(R, S, t), specifies the maximum time t allowed between the occurrence of response R and the occurrence of the subsequent stimulus S.

Example: After the countdown chronometer has been paused, the user should press the Resume button no later than 1800 seconds (otherwise the Countdown mode of operation will be considered abandoned). This timing constraint can be specified as MaxRS(chronometerPaused, resumeCountdownCmd, 1800.0);

[DC4] MinRS(R, S, t), specifies the minimum time t allowed between the occurrence of response R and the occurrence of the subsequent stimulus S.

Example: After the heating process has been completed, the user should wait at least one second before opening the door. This timing constraint can be expressed as MinRS(stopHeating, openDoorCmd, 1.0).

[DC5] MaxSR(S, R, t), specifies the maximum time t allowed between the occurrence of stimulus S and the occurrence of the subsequent response R.

Example: After the user has pressed the Pause button during a heating operation, the actual stopping of the heating process should occur no later than 0.5 seconds. This condition can be expressed as MaxSR(pauseHeatingCmd, stopHeating, 0.5);

[DC6] MinSR(S, R, t), specifies the minimum time t allowed between the occurrence of stimulus S and the occurrence of the subsequent response R.

Example: After the user has pressed the Start button for a heating operation, the actual heating process could start immediately. This can be expressed as MinSR(startHeatingCmd, startHeating, 0.0);

[DC7] MaxRR($R_1$, $R_2$, t), specifies the maximum time t allowed between the occurrence of response $R_1$ and the occurrence of the subsequent response $R_2$.

Example: When the heating process has been completed, the completion should be indicated by three successive beeps, the time interval separating every two consecutive beeps not exceeding 1.5 seconds. The timing condition imposed on the beeps can be specified broadly as MaxRR(endBeep, startBeep, 1.5).

[DC8] MinRR($R_1$, $R_2$, t), specifies the minimum time t allowed between the occurrence of response $R_1$ and the occurrence of the subsequent response $R_2$.

Example: When the heating process has been completed, the completion should be indicated by three successive beeps, the time interval that separates every two consecutive beeps being not less than 1.0 second. This timing condition imposed on the beeps can be specified as MinRR(endBeep, startBeep, 1.0);

[DC9] Duration(A, $t_1$, $t_2$), specifies the minimum time $t_1$ and the maximum time $t_2$ required for action A to last ($t_1$ may be 0, and $t_2$ may be omitted, in which case it will be interpreted as $+\infty$).

Example: The audio signals emitted by the microwave oven to indicate the completion of some operations (such as "done heating," or "chronometer reached zero") should be in the form of beeps whose duration, per beep, should be no less than 1.0 seconds and no more than 2.0 seconds). The last part of this requirement can be described by the expression Duration(Beep, 1.0, 2.0).

Of course, in the S-S cases $S_1$ and $S_2$ may be the same type of stimulus, and in the R-R cases the responses $R_1$ and $R_2$ may be of the same nature. Another observation is that in classes [DC1] to [DC4] the timing constraints are imposed on the system's users, and therefore it is necessary to specify the actions the system must take when these constraints are not satisfied. In such cases, Dasarathy proposes the use of an artificial stimulus, a timer to signal situations in which the user fails to apply the second stimulus within the requirements of classes [DC1] to [DC4]. If a maximum-time constraint is not satisfied the timer will signal the absence of the stimulus within the prescribed deadline and the system will be able to transition to a new state and/or issue a specific response. Similarly, if a minimum-time constraint is disobeyed (that is, the user applies the stimulus too soon) then the armed timer will not go off and this will be considered an undesirable situation, which requires specific treatment. Classes [DC5]

to [DC8] are conditions imposed on the system's performance, rather than on the user's behaviour, and therefore the use of a timer is not necessary.

## 5.3 On the Rigorous Specification of Temporal Constraints

Although essentially simple, Dasarathy's categories of temporal constraints are archetypal for they can be succesfully used to specify of large variety of conditions involving time (more precisely, such conditions can be "reduced", or "translated," to combinations of Dasarathy constraints). However, the way the constraints have been described previously leaves room for interpretations. For instance, the constraint [DC6], defined as MinSR(S, R, t) does not specify whether the occurrence of R is actually required (that is, should R always follow S, or it is possible to have instances of S without subsequent response R?). In addition, as observed from the short examples provided in the previous section (e.g., for [DC4] and [DC8]), it is necessary to associate some temporal markings with the beginning and the end of actions and to take in consideration the actual number of occurrences of a stimulus or response. Moreover, parallel execution of actions is difficult to describe precisely without resorting to additional constructs.

For these reasons, while taking the Dasarathy constraints as a reference basis for formulating the requirements of TCS, we resort in our approach to a formal language, RTL, that unambiguously describes the temporal restrictions placed on such systems. To give only an example, in RTL the constraint [DC6] can be expressed in a more precise way, for instance as

$$\forall i \in \mathbb{N}_1 \quad @(s,i) + d \leq @(\uparrow R, i)$$

meaning that each event s (stimulus) is followed by the start of response R after at least d units of time and no response R can occur without being triggered by s (the notation of the Dasarathy constraint has been adapted for RTL). Other detailed predicates related to [DC6] are possible, for instance the response R can be allowed to occur without being triggered by s.

In fact, one of the reasons for using Z++ as counterpart of UML in our integrated modelling approach was its inclusion of RTL, a precise and easy to comprehend language for expressing time-related properties of the systems.

## 5.4 Real-Time Logic (RTL)

The dynamic aspects of systems are formally expressed in Z++ using statements written in an extended version of RTL. We introduce below the specification language Real-Time Logic, originally proposed by Jahanian and Mok [Jahanian86, Jahanian94] and in the next section indicate the extensions brought by Lano to the language. Because RTL is based on the event-action model, a brief presentation of the major components of the model is given first, followed by a summary overview of the notation.

### 5.4.1 The Event-Action Model

RTL, as described by its inventors, provides a uniform way for specifying both relative and absolute timing of events. The computational model on which RTL relies is centred around two key elements: the first is action, and the other is event. Additionally, the concepts of state predicates and timing constraints complete this model that allows the capturing of data dependencies and of temporal ordering of computations performed by the system in response to external and internal events.

An action is an operation that requires a bounded amount of system resources and is delimited by two events, one denoting its initiation, the other its completion (notational details are given in the next Subsection). An event is a temporal marker that has attached a time value, its time of occurrence, and imposes no requirements on the system's resources. Actions may be either primitive or composite. The former have atomic implementations, while the latter consist of two or more subactions, whose order of precedence can be specified using the sequential or parallel operators. Events can be classified in four categories:

- External events, stimuli received by the system from its surrounding environment, for instance the user pushes the microwave oven's Start button;
- Start events, marking the initiation of some action, for instance the beginning of the Heating operation;
- Stop events, marking the termination of some action, for instance, the end of the Heating operation;
- Transition events, signaling a change in the state of the system, for instance speedLimitReached, indicating the fact that a locomotive has reached the maximum allowed speed under some given conditions (e.g., 60 km/h on a bridge).

### 5.4.2 RTL Concepts and Notations

After the introduction of the two most important concepts of RTL, event and action, an overview of the notation is presented in the following. For practical reasons, we introduce some minor alterations to the notation. Specifically, we use combination of words for longer action names and capitalise each word in the combination, as opposed to Jahanian and Mok's original uppercase only convention. Also, when denoting events we use lowercase single-word identifiers or multiple-word identifiers with all the words of the combination except the first capitalised.

- Actions

  - are denoted by capital letters such as `A`, `B`, etc., capitalised words or combination of capitalised words such as Heating, MoveToNextFloor, or abbreviations such as TCD ("Timer Counting Down");
  - `A.B` denotes the subaction `B` of composite action `A`;
  - $A.B_i$ signifies the `i`-th appearance of subaction `B` within composite action `A`;
  - `B||C` means that subactions `B` and `C` execute in parallel;
  - `B;C` signifies that subactions `B` and `C` execute in sequence, `B` followed by `C`;
  - `!N` indicates a synchronisation point `N`, and `A!N` together with `!NB` specifies that action `A` should be completed before action `B` starts its execution;

- Events

  - external events are denoted using the convention for event identifiers described above and are prefixed by the symbol Ω. For instance, ΩpushStartButton is the event corresponding to the user pressing the button Start;
  - start events are indicated by the symbol ↑, for instance ↑A represents the event associated to the start of action A;
  - stop events are indicated by the symbol ↓, for instance ↓A represents the event associated to the completion of action A;
  - transition events indicate a modification in one of the system's state variables. The notation (S := true) denotes the event corresponding to the transition that makes the state variable S true and (S := false) denotes the transition event that makes the state variable S false.

- The occurrence function, denoted @, is introduced to capture the notion of real time:

  @(E,i) = time of the i-th occurrence of the event E, where $i \in \mathbb{N}_1$

  Note that within Z++ the alternative symbol ♣ is used, as described in Subsection 5.5.3. Therefore, ♣(E,i) is employed in the following chapters of the thesis.

- State predicates, assertions about the state of the system. The value of a state predicate can change over time, as a result of external events and/or system responses. Depending on the boundary conditions, nine forms of state predicates are possible, from $S<t_1,t_2>$, through $S(t_1,t_2)$, to $S[t_1,t_2]$. Informally "<t" means before time t, "(t" means "before or at t", "[t" signifies "at time t", etc. For instance, $S<t_1,t_2]$ specifies that the state predicate S is true before time $t_1$ and remains so until exactly at time $t_2$. An example of state predicate is DoorIsClosed <↑Heating,↓Heating>.

- RTL predicates are formed using arithmetical relations (=, ≠, <, ≤, >, ≥) and algebraic expressions containing integer constants, variables, addition, subtraction, multiplication by constants, and the occurrence function.

- RTL formulae can be constructed using universal and existential quantifiers, equality and inequality predicates, and first order logical connectives. An example of an RTL formula is:

$$\forall i, @(\Omega \text{MailReceived}, i) < \uparrow(\text{DispatchMail}, i) \wedge \downarrow(\text{DispatchMail}, i) < @(\Omega \text{MailReceived}, i) + 60$$

  The above can be interpreted as "action DispatchMail must be executed after the event MailReceived each time the event occurs and must be completed within 60 time units of the occurrence of the MailReceived event."

- Timing constraints complete RTL's underlying model by providing assertional statements about the absolute timing of events that characterise the system's behaviour. Four types of constraints are considered of particular importance in RTL:

  - sequential constraints, constraints on the sequential execution of actions. For instance to indicate that subaction `B` always precedes subaction `C` in the composite action `A` one can write $\forall i\ @(\downarrow A.B, i) \leq @(\uparrow A.C, i);$
  - parallel constraints, constraints on the parallel execution of actions. For instance to indicate that subaction `B` precedes the parallel execution of `C` and `D` within composite action `A` one can write $\forall i\ @(\downarrow A.B, i) \leq @(\uparrow A.C, i) \wedge @(\downarrow A.B, i) \leq @(\uparrow A.D, i);$
  - sporadic timing constraints, given as a requirement for action `A` to complete its execution within a deadline `d` after the occurrence of the event `E`, event for which a separation `p` between occurrences is required;
  - periodical timing constraints, in the form "while `S` is true execute `A` with period `p` and deadline `d`" where `S` is a state variable and `A` an action (the longer RTL formulae for the last two categories of constraints can be found in [Jahanian86]).

## 5.5 Using RTL in Z++

The key idea of Lano's approach for expressing temporal properties of systems is to include in the HISTORY clause of Z++ classes an extended RTL predicate that defines the behaviour

of the objects of the class. This behaviour is seen as a continuous and infinite series of states segmented by occurrences of events, and the RTL predicate holds at all times.

In Z++, the domain TIME of time-valued terms is totally ordered, meaning that in additions to satisfying the axioms for partial order, the following properties also hold: (a) there is a designated element 0, such that $0 \leq t$, for each element $t \in TIME$; and (b) for every pair of elements $(t_1, t_2) \in TIME \times TIME$, $(t_1 < t_2) \vee (t_1 = t_2) \vee (t_1 > t_2)$. The time domain satisfies the axioms of a set of non-negative elements of a totally ordered topological ring, with operation + and *, and units 0 and, respectively, 1. It can be considered that $\mathbb{N} \subseteq TIME$.

The following are summarised from [Lano95], only the concepts and notations needed later in the thesis being presented. Compared with Lano's description, we use the term operation instead of method, this choice being maintained throughout the entire thesis.

### 5.5.1 Lano's Key Extensions to RTL

The key concepts of Lano's extension of RTL are:

- invocation instance, which comprises the initiation, the execution, and the termination of operation op;
- request event, in the form $\rightarrow$ op, denoting the arrival at the current object of a request for the execution of operation op;
- the temporal operators $^{\tau}$ "at all future times", $^{\tau}$ "at some future time", and $\odot$ "holds at";
- counters for operation events #req(op), #act(op), and #fin(op), as defined in 5.5.3.

### 5.5.2 Events

Each operation op of class C has associated the following events:

- $\uparrow$op(x), the initialisation of an invocation instance of op(x), $x \in X$, where X is the set of the operation's possible inputs;

- ↓op(x), the termination of the operation's invocation instance;
- →op(x), the arrival at the object of a request for the invocation of the operation;

Other events are events of the form φ := true or φ := false, where φ is a predicate without modal operators or occurrences of now, denoting that the events of this predicate are true (or, respectively, false), and events for a supplier object s of class S, in the form ↑(ops(x),s), ↓(ops(x),s), and →(ops(x),s). In addition, ←(ops(x),s)signifies the sending from the current object of a request for s to execute the operation ops with input x.

### 5.5.3 Terms

The following terms can appear in a class C's associated RTL formulae:

- variables $v_i$, $i \in \mathbb{N}$;
- attributes of the class, its ancestors, and supertypes;
- n-ary functions in the form $f(e_1, e_2, \ldots, e_n)$;
- ♣e, denoting the time at which e occurs, where e is an event occurrence (E,i), $i \in \mathbb{N}_1$;
- →(op(x),i), ↑(op(x),i), and ↓(op(x),i), where op is an operation of class C, x its input, and $i \in \mathbb{N}_1$;
- event occurrences in the form ←((ops(x),s),i), where s is an object of C's supplier class S and ops an operation of S;
- self;
- now;
- e@t, which indicates the value of e at time t, where e is a term and t a time-valued term;
- ○e, which denotes the value of term e at the next operation initiation time;
- #act(op), the number of initiations of op's execution, up to the present time;

- `#req(op)`, the number of requests for `op's` execution, received by the object up to the present time;
- `#fin(op)`, the number of terminations of `op's` execution, up to the present time.

### 5.5.4 Formulae

Considering a class `C`, the following are RTL formulae related to it:

- $P(e_1, \ldots, e_n)$ for an n-ary predicate symbol `P` and terms $e_1, e_2, \ldots e_n$;
- $\varphi \wedge \psi$, $\varphi \vee \psi$, $\varphi \Rightarrow \psi$, and $\neg \varphi$, for formulae $\varphi$ and $\psi$;
- $\varphi \odot t$, which indicates that $\varphi$ holds at time `t`, where $\varphi$ is a formula and `t` a time-valued term;
- $\forall D \bullet \varphi$ and $\exists D \bullet \varphi$, for declarations `D` and formulae $\varphi$;
- $^{\tau}\varphi$, which denotes that $\varphi$ holds at all future times (not related to `C`); $\varphi$ which means that $\varphi$ holds at each initiation time of an operation from the class; and $\bigcirc\varphi$, which is the value of $\varphi$ at the next operation initiation time;
- $\Diamond^{\tau}\varphi$, which means that eventually $\varphi$ will hold in the future, and $\Diamond\varphi$, which indicates that $\varphi$ will eventually hold at the initiation time of an operation from the class;
- `enabled(op)` and `enabled(op(x))`, where `op` is a method of class `C` and `x` an expression in the input type of `op`, indicating the condition that must hold at the operation's initiation.

### 5.5.5 Abbreviations

Lano also introduces a number of abbreviations, including:

- `#active(op)`, for the number of execution instances of `op` that are currently executing; it abbreviates `#act(op) - #fin(op)`;
- `delay(op,i)` = $\uparrow$`(op,i) -` $\rightarrow$`(op,i)`, the delay between the `i`-th request for the execution of operation `op` and the actual `i`-th initialisation of the operation;

- duration(op,i) = ↓op(i) - ↑op(i), the duration of operation op's i-th execution;
- mutex({$op_1$, ..., $op_n$}), meaning that at any given moment a method $op_k$ of the set {$op_1$, ..., $op_n$} has a number of active instances that is equal to the total number of active instances of all the operations in the set;
- self_mutex({$op_1$, ..., $op_n$}), meaning that each operation $op_k$ in the set {$op_1$,..., $op_n$} has at most one active instance at any given moment;
- op, which is an abbreviation for #active(op) > 0;

### 5.5.6 Axioms

A comprehensive set of axioms is included in Lano's book. For illustration purposes two are given below, but for full details we refer the reader to Appendix A of [Lano95]:

(a) At any given time, there cannot be more terminations of op(x) than activations:

$$\forall i \in \mathbb{N}_1 \bullet \clubsuit(\uparrow op(x), i) \leq \clubsuit(\downarrow op(x), i)$$

(b) Event occurrences are indexed ordered on their time of occurrence:

$$\forall i,j \in \mathbb{N}_1 \bullet i \leq j \Rightarrow \clubsuit(E,i) \leq \clubsuit(E,j)$$

## 5.6 Chapter Summary

In this chapter the formal basis for expressing temporal properties of the systems has been presented. Since requirements on the behaviour of TCS can be described using the classes of constraints proposed by Dasarathy, a review and respecification of these classes using a simple notation and small examples related to a microwave oven application have been presented. Also, since, in our modelling technique, the capturing of timing constraints is performed using extended RTL formulae, an overview of the notational elements of RTL as well as a brief description of its underlying action-event model has been provided. Lano's extensions of RTL have also been presented. As a result, the preparation for the formalisation of UML models, including timing restrictions on the behaviour of systems, has been completed. The following chapter provides additional details on the specific use of RTL in our approach.