REIHE INFORMATIK
3/96

Real-Time Estelle

Stefan Fischer
Universität Mannheim
Fakultät für Mathematik und Informatik
Seminargebäude A5
D-68131 Mannheim

Real-Time Estelle

Stefan Fischer
Praktische Informatik IV, University of Mannheim
D-68131 Mannheim, GERMANY
Phone: +49 621 292 5053, Fax: +49 621 292 5745
`stefis@pi4.informatik.uni-mannheim.de`

Abstract

Estelle is one of the standardized Formal Description Techniques for the specification of communication protocols and distributed systems. Unfortunately, Estelle is not capable to express real-time requirements or characteristics of services or protocols which is especially important in the context of distributed multimedia systems. In this paper, we introduce an extension to Estelle called Real-Time Estelle that allows to describe real-time systems. We introduce the syntax of the new language and propose both an operational and a hybrid semantics. Examples show the usefulness of the approach. We also discuss ways to implement Real-Time Estelle specifications.

Keywords: Estelle, real-time systems, multimedia systems, quality of service, automatic implementation

1 Introduction

One of the major applications in the framework of the emerging "information superhighway" will be distributed multimedia systems. Compared to traditional applications in data processing, these systems have totally different requirements in terms of data transfer rates, response time etc. All these requirements are typically related to *real-time* and usually expressed by quantitative *Quality of Service* (QoS) parameters.

The use of Formal Description Techniques has proved useful for the specification of traditional protocols and distributed systems. However, most of these languages have no real-time concept. Thus, it is often impossible to express quantitative QoS requirements based on a formal syntax and semantics.

Estelle is one of the standardized Formal Techniques for the specification of protocols and distributed systems. Estelle has many advantages. Since the language is a superset of the programming language Pascal, Estelle specifications are easy to understand, and it is possible to derive simulations and efficient implementations in a straight-forward way.

In [15] however, we showed that Estelle is not very good at specifying real-time constraints. The Estelle model of time is too weak to express real-time requirements. Thus, in this report, we develop a real-time extension of Estelle called *Real-Time Estelle*. We show that it is possible with this extension to express timing constraints in terms of real-time and thus quantitative QoS requirements or characteristics.

The rest of this report is organized as follows: in Section 2, we describe a general formal model of real-time systems. We also give an overview of earlier approaches to formal description of real-time systems. Then, we extend the Estelle transition model to fit into this general formal model. In Sections 3 and 4, we describe the syntax and semantics of Real-Time Estelle which allow us to construct real-time systems in terms of the extended real-time transition model. Section 5 presents examples of how to use the extension for QoS descriptions and modeling of system timing behavior. In Section 6, we show how implementations may be derived from Real-Time Estelle specifications which are capable of guaranteeing specified QoS requirements. Section 7 concludes the report.

2 Real-Time Model

2.1 Formal Definition of a Real-Time Model

In [3], a model for real-time systems is given which concludes many of the approaches developed so far:

Definition 2.1 *A real-time system is modeled as a set of timed state sequences. The domain of time is the number of positive real numbers $\mathbb{R}^+$. In a timed state sequence, intervals on $\mathbb{R}^+$ determine the duration of system states. Formally, a real-time system $S = (\mathcal{S}, \mathcal{P}, \mu, \mathcal{T})$ consists of the following four components:*

- *A set $\mathcal{S}$ of system states.*
- *A set $\mathcal{P}$ of atomic propositions observable in system states.*
- *A function $\mu : \mathcal{S} \rightarrow 2^{\mathcal{P}}$ which determines which propositions are observable in each system state.*
- *A set $\mathcal{T}$ of timed state sequences $s = s_0 I_0 s_1 I_1 ...$, where $s_i \in \mathcal{S}$ and $I = I_0 I_1 ...$ is an interval sequence such that for every $t \in \mathbb{R}^+$, there exists exactly one interval $I_i \subset \mathbb{R}^+$ with $t \in I_i$. Thus, the expression (s_i, I_i) means that at every time instant $t \in I_i$ the system is in state s_i.*

For many real-time systems, this definition may be simplified. Often, it suffices to characterize a system by its timed observation sequences instead of by state sequences. An observation sequence is a sequence $\sigma = \sigma_0 \sigma_1 \sigma_2 ...., \sigma_i \subseteq 2^{\mathcal{P}}$. A *timed observation sequence* $\rho = (\bar{\sigma}, \bar{I})$ consists of an observation sequence $\bar{\sigma}$ and an interval sequence $\bar{I}$ of equal

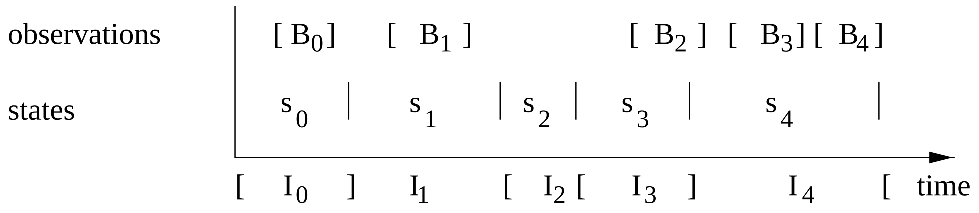

Figure 1: Timed state and timed observation sequences

length. A pair (σ_i, I_i) means that during interval I_i the observation σ_i does not change. The realtionship between timed state and timed observation sequences is shown in Figure 1.

The observations B_0 and B_1 are correct. B_2, however, is not correct since the state changes during the observation interval. Looking at two consecutive observations, the state of the system does not necessarily change (B_3 and B_4). Also. not necessarily all states of a system are observed.

A second important simplification is possible, if only *instantaneous events* occur. In this case, it suffices to record only singular intervals, i.e. the points in time where events happen (which is equivalent to state changes).

If there is no causal relationship between single events (concurrency), one often uses *interleaving semantics*. Here, the occurrence of several events at the same time are mapped to all possible sequences with only one event in each sequence member. The occurrence of events p and q at time 3, also described as $(\{p, q\}, 3)$ will then be mapped to two sequences $(\{p\}, 3) \rightarrow (\{q\}, 3)$ and $(\{q\}, 3) \rightarrow (\{p\}, 3)$ which both model a possible behavior of the system. Interleaving semantics reduces the state space of a system and thus simplifies the analysis.

In the case of instantaneous events, it also suffices to use natural numbers for the representation of time. The time unit has then to be selected accordingly.

2.2 Constructing and Describing Real-Time Systems

We now have a very general formal model for real-time systems. However, it may be quite costly or even impossible to enumerate all timed state sequences describing the system. Therefore, formalisms — formal description techniques — have been developed to describe implicitly those sequences. In this section, we will give a short overview on formalisms for real-time systems.

Generally, the techniques may be divided into two categories: constructive and descriptive approaches. The first group provides an explicit algorithm of how to construct the state sequences, while the second one describes characteristics of the system.

Basically, all constructive approaches are based on *state transition systems*. They may be further subdivided into timed automata-, timed Petri Net- and timed process algebra-based methods.

The work of Alur and Dill [2], Henzinger, Manna and Pnueli [20], Rudin [36] and Lynch and Vandraager [28] is all based on some kind of timed automata. Time restrictions are introduced by labeling transitions or states of finite state machines with time limits. Henzinger et al. [20], e.g., assign to each transition of their Timed Transition System (TTS) an upper bound u and a lower bound l. Once enabled, the transition has to be delayed at least l time units, and it has to be fired before u time units have elapsed. Figure 2 shows a sample TTS.

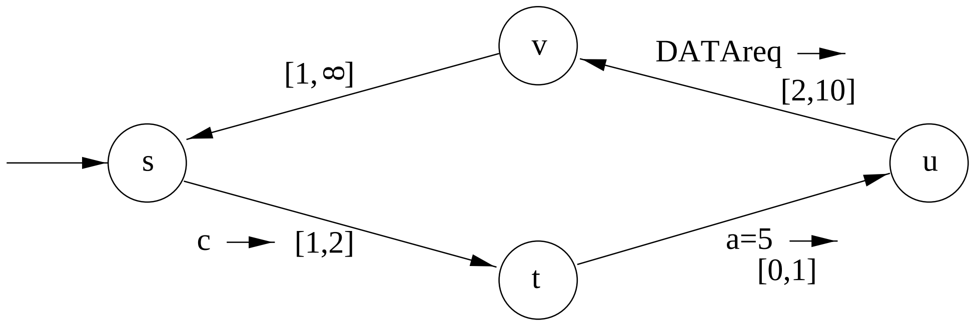

Figure 2: A Timed Transition System

If the system starts in state s, then the execution of the transition has to be delayed for at least one unit time after condition c has become true. At most 2 units time after that, the transition has to be executed, an the system switches to state t. By this TTS, the following timed state sequences (among others) are constructed.

$$(s, 0) \to (t, 1.5) \to (u, 1.7) \to (v, 6.7) \to (s, 9.0) \to ....$$
$$(s, 0) \to (t, 1.8) \to (u, 2.0) \to (v, 12.0) \to (s, 14.0) \to ....$$

Here, the transitions are instantaneous which means that it suffices to record only singular intervals.

There has also been much and early work in the area of timed Petri Nets, e.g. *Time Petri Nets* [31], *Object Composition Petri Nets* [26] and *Time Stream Petri Nets* [38]. Especially Little and Ghafoor [26] made Petri Nets popular for the specification of synchronization relations in multimedia systems.

We do not further elaborate on timed process algebras. The interested reader may find a good starting point in [33], though development has since progressed.

Descriptive techniques are usually based on *temporal logic* (TL), for an overview see [19]). Generally, temporal logic formulas describe temporal relations between states which are characterized by state propositions. The formulas are interpreted over state sequences. All state sequences fulfilling the formulas compose the system. For the description of real-time systems, traditional temporal logic is not sufficient. Therefore, extensions have been introduced to TL to express real-time relations, e.g. timed operators as in Metric Temporal Logic (MTL) [22] or Quality of Service Temporal Logic QTL [6], or explicit timer variables as in Real-Time Temporal Logic (RTTL) [34] or TLA [1]. Most of them are based on linear time and use natural numbers as their time domain.

High Level Languages. To make the use of these approaches more convenient, several of them have been realized in higher level languages. Especially for timed process algebras, several extensions of LOTOS have been developed, e.g. in [35, 24, 11]. Basically, in all these approaches, the occurrence of actions is restricted to certain time intervals. There have also been some approaches to extend the language Estelle by some real-time aspects [13, 45, 9], but none of the methods described above is addressed explicitly there. A popular recent method is to provide a hybrid language. A system's functional behavior is specified by automata or algebra, while its timing constraints are expressed in temporal logic. The real-time system is then characterized by all the timed state sequences produced by the automata/algebra which in addition fulfill the temporal logic formulas. Bowman et al. [6] describe this for LOTOS in conjunction with QTL and Leue [25] for SDL with MTL. We will employ this approach for Estelle in Section 4.

Suitability for Quality of Service Specification. Our main goal is to describe quality of service requirements or characteristics. Typically, those requirements cannot be expressed by timing expressions belonging to one transition of one automaton or transition system. Looking e.g. at the requirement that a connection setup should take no longer than 5 units time, usually, at least two transitions of an automaton and often even several automata are involved. The suitable means to express this requirement is by relating communication events, e.g. "After receiving a CONreq, you have to send a CONcnf within 5 units time ". This, however, may be much better done using descriptive techniques. In MTL, one would write: $RECEIVE(CONind) \supset \Diamond_{\leq 5} SEND(CONcnf)$. On the other hand, this makes it much more difficult to derive efficient implementations, as we do not have an explicit algorithm to construct the system. In distributed system specification, descriptive techniques are often used for service specifications while protocols are described by constructive techniques like automata [18].

Besides QoS requirement modeling, performance descriptions may be an interesting application. Here, data are often available on the execution speed of single transitions, which makes it suitable to restrict their execution times and thus use e.g. the TTS semantics. Therefore, we do not restrict Real-Time Estelle to one or the other technique. In Section 3, we develop a syntax which is flexible enough for a descriptive and an operational semantics, both described in Section 4.

2.3 Preparing the Estelle Model for Real-Time

Before the syntax and semantics of Estelle may be extended, we first have to evaluate whether the current model of Estelle fits into the real-time model described in Section 2.1. Our task is thus to compare the four constituents of the general real-time model to the Estelle model. Where there is no equivalence, the Estelle model has to be extended.

The set of states $\mathcal{S}$. According to the Estelle standard, an Estelle specification describes a set of state sequences. A state is called a *global situation (sit)* and consists basically of the states of the single modules and the connection and hierarchical relationships between them. The global state of the described system is changed by the execution of a transition of one of the modules or by the selection of a new set of executable transitions in a system module. A sequence $sit = sit_0 sit_1 sit_2 ...$ of global situations is called a *computation* and is a state sequence. All possible computations compose the system described by the Estelle specification.

The observables $\mathcal{P}$ and the labeling function μ. The Estelle standard does not define what is "observable" in a specified system. It is, however, clear that states which exist during the execution of a transition are not visible meaning that transitions are instantaneous. States may thus only be observed between the execution of two transitions.
Adhering to the information hiding principle of Estelle, which says that no internal data — variables, queue contents, major state etc. — of a module is accessible to the outside, we define that internal data are also not observable from the outside. However, they may be used inside a module body to define timing constraints on states. Hence, service requirements which are imposed externally on a system may only be expressed by reference to its interface, i.e. interaction points and external variables.
We define the following to be observable inside a module:

- local and exported variables,
- enabling conditions of transitions,
  for each of which we define a predicate. For example, the predicate `WHEN(p.m)` is true, if Message m is the first message of the queue of interaction point p. The state of a module may be accessed by the predicate `MAJOR_STATE(x)`, which is true if the module is currently in state x.
- In accordance with [25], we define predicates RECEPTION-OF(p.m) and SENDING-OF(p.m) which are true, if in the last transition, message m has been received resp. sent via interaction point p. To allow counting of messages, the instance operator `[]` is introduced. SENDING-OF(p.m)[z] means the sending of the zth instance of m. The use of this operator implies the existence of an event counter for each possible message at each interaction point. These predicates allow to describe relationships between communication events, a very important feature of QoS specification.

With this definition, we also have already gained our labeling function.

The set $\mathcal{T}$ of timed state sequences. It remains to map Estelle state sequences into real-time. As we have seen, transitions from one state to the next are instantaneous. Thus, we may restrict ourselves to singular intervals, i.e. points in time, as well as to natural numbers as time domain. We will also use interleaving semantics which is standard in Estelle anyway.

We construct a timed observation sequence from the Estelle state sequence by adding a function $f : \mathcal{S} \mapsto \mathbb{N}$ that maps each global situation sit_i onto its time of occurrence. The time function increases monotonically, in conformance to the Estelle standard. In fact, we can simply use the external time process defined there.

A timed state sequence as we need it thus has the form $\tau = (sit_0, f(sit_0)), (sit_1, f(sit_1)), ...$ With these definitions, every Estelle specification describes a set of timed state sequences and thus a real-time system. In the next sections, we develop a language extension which allows to describe timing constraints on timed state sequences.

3 Syntax Extensions

We now have a real-time transition model that maps every Estelle specification onto a real-time system. However, we still have no means to express characteristics of such real-time systems, i.e. certain timing requirements on state sequences. These means will be developed syntactically in this section and semantically in the following one.

The design of the language extension was driven by the following criteria:

1. The extension should fit harmoniously into the existing Estelle language. Estelle is easy to understand, and we do not want to make its specifications unreadable.

2. It should be usable in as many stages of the software life cycle as possible. After specifying a system, it should be possible to verify or test the system, but it is also important that it be possible to derive efficient implementations automatically.

3. Generally, it should be possible to assign several — descriptive and operational — semantics to the extension.

In the following, we describe the syntactic extensions. First, we show how to write real-time state descriptions. These states are then put into a timed relationship. Afterwards, the extension is embedded into the existing language Estelle. We also describe ways to specify exceptions, i.e., what happens when a timing constraint cannot be fulfilled. Finally, some abbreviations are presented which are very useful in many situations.

State descriptions. State descriptions consist of state propositions defined in Section 2.3. The following rules define correct state descriptions:

1. If `p` is an observable proposition in a module, then it is a state description.
2. If `p` and `q` are state descriptions, then `p AND q`, `p OR q`, `p IMPLIES q`, `p OTHERWISE q` and `NOT p` are state descriptions.

Temporal relationship between states. Refering to time in state descriptions becomes possible by using the time function `now` and *time variables.* The function `now` provides values of type `time`, which denote the current system time. In a Real-Time Estelle restriction, the value of `now` may be different for different states. The value of time variables is the same for the whole formula (rigid variables). This is enforced by quantifiers discussed in the next paragraph. Values of `now` can be "stored" in time variables and later be referenced. The type `time` is implicitly defined as `TYPE time=integer`, and the time domain is given by the `timescale` option.

Time variables can be used in *time expressions.* They may be compared to each other or to time constants (using the operators $=, <, >, \geq, \leq$). `Now` is considered to be a special time variable and may also be used in time expressions. Finally, it is allowed to add constants to time variables.

To express, e.g., that during the last transition the message `DATAreq` has been received and this is observable at time instant 7, we may write in Real-Time Estelle:

```
SENDING_OF DATAreq AND now=7
```

To determine the occurrence of states in the future, it is necessary to use *temporal operators.* In Real-Time Estelle, the operators `HENCEFORTH` and `EVENTUALLY` are available. `HENCEFORTH p` means, that from now on, `p` is always true. Similarly, `EVENTUALLY p` means that there is a future state where `p` is true. The following bounded-response property expresses that `q` is observable within 3 units time after `p`:

```
p AND x=now IMPLIES EVENTUALLY (q AND now <= x+3)
```

Quantification of variables and embedding into standard Estelle. To embed real-time requirements into standard Estelle specifications, we considered three possibilities: assign them to the module header description, i.e. to the interface, put them into the transition part or define a new section in the module body. The first choice was rejected, as it would then be impossible to access local variables. The second choice was rejected because it is not general enough. Our timing requirements are not necessarily related to transitions. Thus, it was decided to add a new section to a module body, named `TIME CONSTRAINTS`. All timing requirements for one module are collected here.

Every single time constraint may be enhanced by a `NAME` clause and quantification clauses `FORALL` and `EXISTS`. The expression `FORALL x:time; p AND x=now` is the typical first-order logic expression for `(p AND 0=now) AND (p AND 1=now) AND (p AND 2=now)` ....

Similarly, `EXISTS x : time; p AT x` is equivalent to `(p AND 0=now) OR (p AND 1=now) OR (p AND 2=now) ....`

Every time variable occuring in a temporal restriction has to be quantified.

Exceptions. It is often unrealistic to assume that hard real-time guarantees can really be given. Therefore, it is useful to allow the specification of exceptions, i.e. to specify what happens, when a timing requirement is not fulfilled. Generally, such a reaction could be expressed as follows in Real-Time Estelle:

```
NOT ((p AND x=now) IMPLIES EVENTUALLY (q and now < x+c) IMPLIES
EVENTUALLY (r and now < x+d ))
```

or alternatively using the operator `OR`:

```
(p AND x=now) IMPLIES (EVENTUALLY (q and now <x+c) OR
EVENTUALLY (r and now < x+d))
```

However, this makes it difficult to understand the intention of the specifier. Even more important, it will confuse a compiler and run-time system trying to guarantee existing real-time constraints. The compiler would have no chance to find out which constraint to fulfill. Thus, for the expression of exceptions, we add the language construct `OTHERWISE` to Real-Time Estelle. The expression above then reads

```
(p AT x IMPLIES EVENTUALLY (q and now <x+c))
OTHERWISE EVENTUALLY  (r and now < x+d)
```

Both problems mentioned above disappear: the meaning is easy to understand, and for a compiler, the constraint to be optimized can be uniquely identified. However, as we will see in the next section, there is no *semantical* difference between all three descriptions which means that semantically, none of the alternatives `q` or `r` is favored. But a compiler may detect on a purely syntactical analysis which constraint should be fulfilled and which describes the way out, if the main constraint cannot be fulfilled.

Abbreviations. The following abbreviations simplify the specification of many practically relevant temporal restrictions: the expression `p AND` π, where π is a time expression, can be replaced by `p AT` π. All occurences of `=now` may be omitted. Thus, `p AT x` is equivalent to `p AND x=now`. The combination `IMPLIES EVENTUALLY (`ϕ `and x=now)` can be replaced by `LEADSTO` ϕ ersetzt werden[1]. Similarly, `IMPLIES HENCEFORTH (`ϕ `and x=now)` may be replaced by `FORBIDS NOT` ϕ.

[1]The operator `LEADSTO` in this meaning was first introduced by Lamport [23].

Example. A complete time restriction is given in the following example: a module has some transitions which transform its major state from `idle` to `connected`. This transition should be executed within 5 units time after its activation. The transition part reads as follows:

```
TRANS
   FROM idle TO pending begin
      output sap.REQUEST;
   end;

   FROM pending TO connected
      WHEN sap.CONFIRMATION begin end;
```

The `TIME CONSTRAINTS` part contains the following restriction:

```
TIME CONSTRAINTS
   NAME  tc1:
   FORALL  x : time;
       HENCEFORTH (
        MAJOR_STATE(idle) AT x LEADSTO
           MAJOR_STATE(connected) AT (now<=x+5));
```

Examples of usage of all of these clauses are given in Section 5. The formal syntax rules in BNF may be found in Appendix A.

4 Semantics

In the previous section, we defined how to write syntactically correct time descriptions. Now, we will assign a formal semantics to these language features. In Section 2, several formal semantics were evaluated with respect to their suitability for the specification of QoS restrictions. It turned out that descriptive techniques are much better suited than operational semantics. However, the semantics of Estelle is operational. Thus, we will first discuss a hybrid semantics which integrates Estelle's operational semantics with a descriptive one for the time restriction. Since Real-Time Estelle is not only useful for QoS descriptions, we present, in the second part, a purely operational semantics based on Timed Transition Systems.

4.1 A hybrid approach

The following semantics is called "hybrid", since it contains both operational and descriptive parts.The operational parts arise from standard Estelle, while the Real-Time

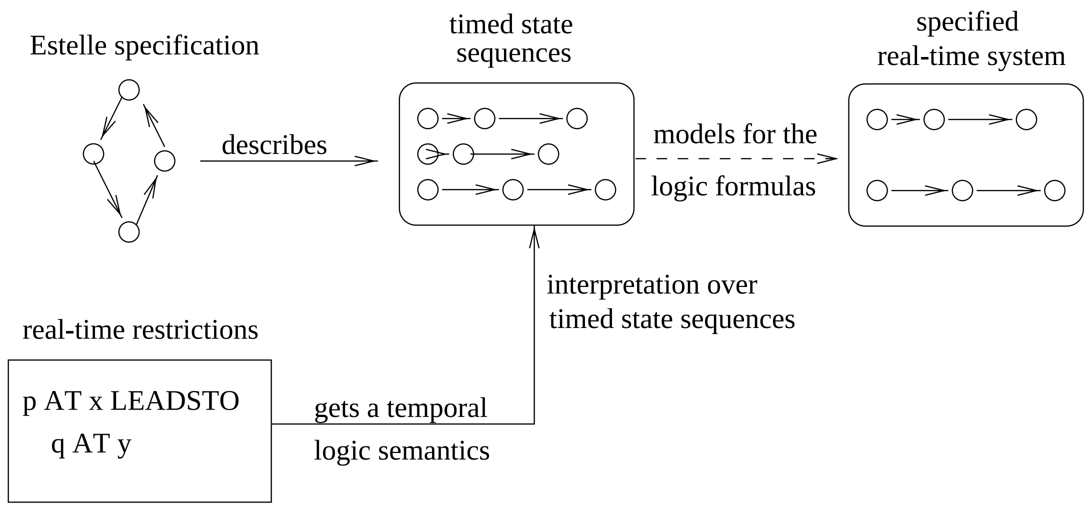

Figure 3: Hybrid semantics for Real-Time Estelle.

Estelle part determines the descriptive part. The overall specification consists of timed observation sequences which are constructed as follows: first, timed state sequences are constructed using the operational semantics described in the Estelle Standard and the real-time extensions developed in Section 2. Then, these sequences are used as models for the temporal formulas described by the Real-Time Estelle restrictions. Only those sequences which satisfy all formulas are part of the overall real-time system. This approach is shown graphically in Figure 3.

The task of this section is therefore to provide the satisfaction relation for timed state sequences with respect to Real-Time Estelle restrictions.

The following terms are used in the satisfaction relation's definition: V is the set of all time variables and $x \in V$. Let $p \in \mathcal{P}$ be an observable proposition, $\pi \in \Pi(V \cup \{now\})$ a time expression containing free variables from V and ϕ, ϕ_1, ϕ_2 time restrictions in Real-Time Estelle. In addition, let $\rho = (\breve{\sigma}, \breve{I})$ be a timed observation sequence and $B : V \to \mathbb{N}_0^+$ a value assignment for all time variables.

Using these terms, the hybrid semantics of Real-Time Estelle may be defined as follows with the satisfaction relation: A timed observation sequence ρ satisfies the restriction ϕ if and only if $(\rho, 0) \models_B \phi$ for all environments B. The relation $\models_B$ is inductively defined by:

(i) $(\rho, t) \models_B p$ iff $p \in \sigma_i$, where $t \in I_i$

(ii) $(\rho, t) \models_B \pi$ iff $B[now := t] \models \pi$

(iii) $(\rho, t) \models_B \texttt{NOT}\ \phi$ iff $(\rho, t) \not\models_B \phi$

(iv) $(\rho, t) \models_B \phi_1$ `AND` ϕ_2 iff $(\rho, t) \models_B \phi_1$ and $(\rho, t) \models_B \phi_1$

(v) $(\rho, t) \models_B$ `HENCEFORTH` ϕ iff $(\rho, t') \models_B \phi$ for all $t' \geq t$

(vi) $(\rho, t) \models_B$ `FORALL x:time;` ϕ iff $(\rho, t) \models_{B[x:=t']} \phi$ for all $t' \in \mathbb{N}_0^+$

(vii) $(\rho, t) \models_B$ `EXISTS x:time;` ϕ iff $(\rho, t) \models_{B[x:=t']} \phi$ for some $t' \in \mathbb{N}_0^+$

Informally, those definitions express the following: definition (i) determines the interpretation of observable propositions. A timed observation sequence satisfies p at time instant t if and only if p is observed at t. Definition (ii) gives the meaning of time expressions. $(\rho, 0)$ satisfies a time expression if and only if the time variables' value assignment satisfies the time expression, assuming that `now` is set to `t`. An example: The time expression $now < 3$ is satisfied if $t = 2$, but not if $t = 5$. Topics (iii) and (iv) define the meaning of logical operators, and (v) that of temporal operators. Finally, (vi) and (vii) define the semantics of the `FORALL` and `EXISTS` clauses.

The remaining operators may be derived from those defined above as follows:

(i) ϕ_1 `OR` $\phi_2 \equiv$ `NOT (NOT` ϕ_1 `AND NOT` ϕ_2 `)`

(ii) ϕ_1 `IMPLIES` $\phi_2 \equiv$ `NOT` ϕ_1 `OR` ϕ_2

(iii) ϕ_1 `OTHERWISE` $\phi_2 \equiv \phi_1$ `OR` $\phi 2$

(iv) `p AT x` $\equiv$ `p AND x=now`

(v) `EVENTUALLY` $\phi \equiv$ `NOT HENCEFORTH NOT` ϕ

(vi) ϕ_1 `LEADSTO` $\phi_2 \equiv \phi_1$ `IMPLIES EVENTUALLY(` ϕ_2 `AND y=now)`, where `y` is used in ϕ_2 (if a new variable is used).

(vii) ϕ_1 `FORBIDS` $\phi_2 \equiv \phi_1$ `IMPLIES HENCEFORTH NOT (` ϕ_2 `AND y=now)`

This is the semantics of a first-order logic with time variables. It is similar to the semantics of RTTL [34]. As a result, Real-Time Estelle is very expressive, but undecidable. For many applications, a decidable logic such as MTL would have been sufficient, but in Section 5.2, we show some examples which are not specifiable in languages like MTL. This semantics was selected since we are more interested in expressiveness than in decidability.

If a timed observation sequence has been produced by this hybrid semantics, it is a correct representation of the overall system.

4.2 Timed Transition Systems

In some previous work [13, 9], approaches to a kind of TTS semantics have been made for Estelle. There, time restrictions refer to exactly one Estelle transition. Such a semantics can be useful as several examples show.

Standard Estelle specifications define transition systems. With the new Estelle real-time model defined in Section 2, a basis for the transition to *timed* transition systems is available. Using Real-Time Estelle restrictions, upper and lower time bounds for the execution of a transition may be defined. The TTS semantics is defined by translating Real-Time Estelle language constructs into those of TTS.

To allow such a translation, real-time restrictions must not refer to more than one transition. In such a case, we would not be able to assign a restriction to exactly one transition, which is necessary for TTS. The time restriction has to express that, if a certain transition is activated, there is a time interval during which the transition has to be fired. Each restriction has to identify the state in which a certain transition is enabled (`p`), and the state which is reached by firing the transition. The state description `p` has to be a conjunction of all enabling conditions of the transition, and `q` will typically determine the major state to be reached:

```
FORALL x:time;
(enabling_condition_of_t AT x FORBIDS
  MAJOR_STATE(to-state) AT (now < x+l)) AND
(enabling_condition_of_t AT x LEADSTO
  MAJOR_STATE(to-state) AT (x+l <= now <= x+u))
```

If these conditions are fulfilled, then every Real-Time Estelle restriction describes a restriction for a transition of a TTS. The mapping rules are as follows:

- If the time expression of `q` has the form `(x+l OP1 y OP2 x+u)`, then the referred Estelle transition is labeled with the interval $[l, u]$. The operator (`<,<=`) determines the character of the interval (open, closed, half open).
- If the time expression is simplified to `(y OP x+c)`, then the transition is labeled according to the following rules:
  - `<` resp. `<=` leads to $[0, c[$ resp. $[0, c]$.
  - `>` resp. `>=` leads to $]c, \infty[$ resp. $[c, \infty[$.
  - `=` leads to $[c, c]$.

The process of constructing a TTS from Real-Time Estelle specifications may be seen in Figure 4. A complete example is given in Section 5.3.

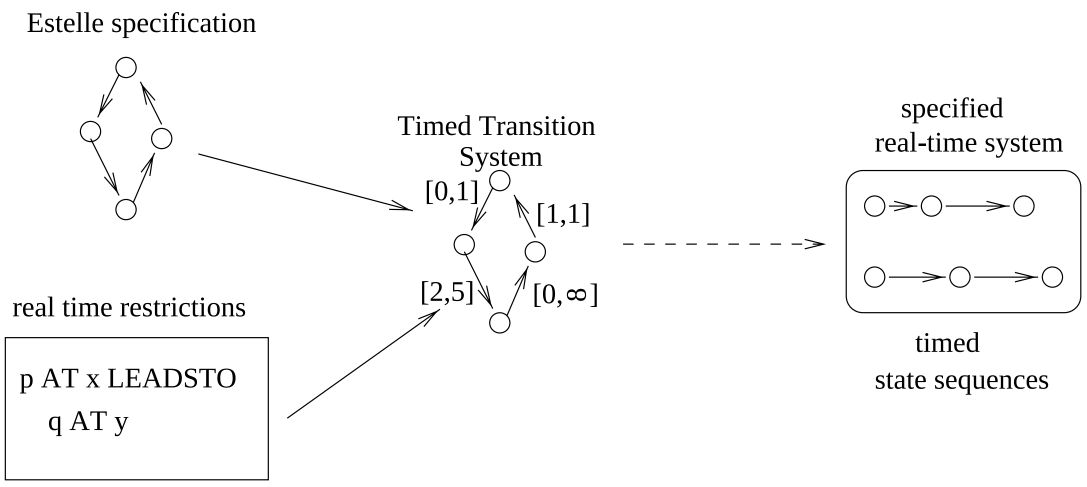

Figure 4: Semantics of Timed Transition Systems for Real-Time Estelle.

5 Examples

In this section, some sample applications for Real-Time Estelle will be presented. The main part is concerned with QoS restrictions and mechanisms. But we also examine the suitability of the new language for the specification of Timed Transition Systems.

Usually, only the time restrictions are given. Where necessary, however, the relevant parts of the Estelle automaton are described, too.

5.1 QoS

In the following, the specification of a transport service displayed in Figure 5 is used to show how Real-Time Estelle can be used to specify QoS requirements and restrictions. The transport service is provided by the XTP protocol [42]. The Estelle specification of XTP used in this context is provided by the INT (France) [8].

5.1.1 Data Rates and Throughput

Data rates and throughput may be specified in many different ways. They may be expressed by the number of messages per unit time or alternatively by the number of user data octets per unit time. Also, it is important to determine the point of reference. Throughput may, e.g., be related to the interface between service user and provider, which means that the network throughput is not regarded. A definition, however, that refers to the number of data transmitted from a sender to a receiver, would take the

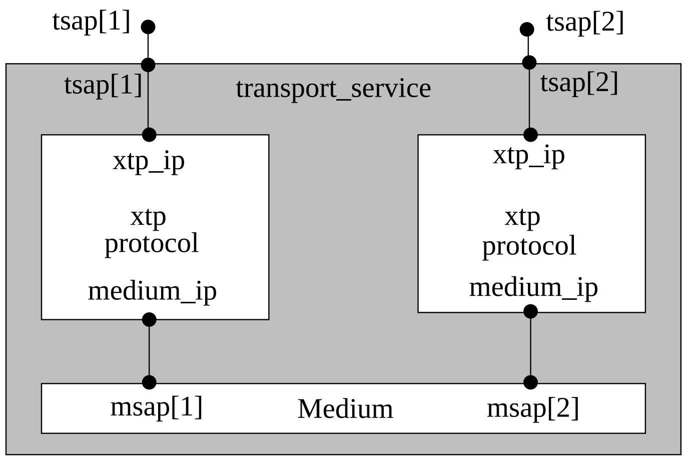

Figure 5: Sample specification of QoS restrictions

network into account. In the following examples, some practically relevant data rates are specified.

Minimum message rate at the transport interface. A minimum data rate based on messages offered by the transport service may be specified by giving an upper bound for the packet inter-arrival time.

```
NAME minimal-message-rate:
FORALL x : time;
  HENCEFORTH (
    SENDING OF tsap[2].T_DATAind AT x LEADSTO
      SENDING OF tsap[2].T_DATAind AT (x < now <= x+5));
```

It is important to note that the occurrence of the second `T_DATAind` has been restricted to an instant `y > x`. Without this restriction, this formula would be always satisfied as the point in time $y = x$ would also be included. Thus, the same message would be referenced twice. Using the instance operator, this problem disappears:

```
NAME minimum-throughput
FORALL x : time;
FORALL z : integer;
  HENCEFORTH (
    SENDING OF tsap[2].T_DATAind[z] AT x LEADSTO
    SENDING OF tsap[2].T_DATAind[z+1] AT (now <= x+5));
```

Maximum message rate at the transport interface. Similarly, the user may specify that his traffic pattern will respect a certain maximum message rate. To specify this, the sending of a data unit has to be forbidden for a certain time after the last sending of a data unit.

```
NAME maximum-data-rate:
FORALL x : time;
  HENCEFORTH (
    SENDING OF tsap[1].T_DATAreq AT x FORBIDS
    SENDING OF tsap[1].T_DATAreq AT (x < now <= x+5));
```

The same method may be used to specify that a telecommunications system may only provide a certain maximum throughput. For the sample medium module, this restriction is given below:

```
NAME maximum-possible-data-rate:
FORALL x : time;
  HENCEFORTH (
    SENDING OF msap[2].M_DATAind AT x FORBIDS
    SENDING OF msap[2].M_DATAind AT (x < now <= now+5));
```

Average rates. To provide average restrictions, it is necessary to restrict the time not between two consecutive, but between the n-th and (n+x)-th data packets. The following restriction expresses the wish of a transport service user, that every sequence of 10 consecutive data packets have to be received within 35 to 45 units time.

```
NAME average-message-rate:
FORALL x : time;
FORALL z : integer;
  HENCEFORTH (
    RECEIVING OF tsap[2].T_DATAind[z] AT x LEADSTO
    RECEIVING OF tsap[2].T_DATAind[z+9] AT (x+35 <= now <= x+45));
```

The variable `z` is determined by the current value of the instance counter for message `T_DATAind` at interaction point `tsap[2]`. Thus, when at a point in time x_0 the message `T_DATAind` has just been received at IP `tsap[2]`, then the actual value of this counter will be frozen in `z`, and a point in time y_0 must exist which is between 35 and 45 units time after x_0 and where a new instance of `T_DATAind` has just been received and where the actual value of the instance counter is `z+9`.

Throughput formulated in Mbit/s. Real-Time Estelle also allows specifications of data rates which are not based on messages per unit time, but on the more popular unit Mbit/s. Consider a transport service user who is expecting a data stream with a rate of at least D Mbit/s with a packet size of $size$ Kbit:

```
NAME mbit-throughput:
FORALL x: time;
FORALL z: integer;
  HENCEFORTH (
    RECEIVING OF tsap[2].T_DATAind[z] AT x LEADSTO
    RECEIVING OF tsap[2].T_DATAind[z+1000*D/size-1]
        AT (now <= x+1000));
```

In the Estelle specification, the selected `TIMESCALE` should be `milliseconds`.

5.1.2 Delays

Delays in communication subsystems are induced by the duration of processes. On local sites, they are due to computation times for protocol processing, I/O operations etc. From the global point of view, delays are produced by network transfer. Local and global delays combine to end-to-end delay experienced by the user. In the following, some important delays are specified:

Minimum delay of data transfer. To model a minimum delay introduced by the medium, the output of a received data packet at the receiver's IP has to be forbidden for a certain amount of time:

```
NAME minimum-delay:
FORALL x : time;
FORALL z : integer;
  HENCEFORTH (
    RECEPTION OF msap[1].M_DATAreq[z] AT x FORBIDS
    SENDING OF msap[2].M_DATAind[z] AT (now < x+5));
```

It is worth noting that the modeled computation time covers the time from receiving the message from the sender's IP until the new message is output at the receiver's IP. Due to the asynchronous nature of IP communication in Estelle, this is *not* the same as the time between *sending* a message to the medium and *receiving* it at the other site. Thus, it has to be carefully analyzed which time interval really has to be modeled. This is a general problem of all asynchronous communication mechanisms.

Maximum end-to-end delay. A transport user wishes that the sending of a data packet leads to its reception by another transport user not later than 5 units time after sending. This a global delay.

```
NAME end-to-end-delay:
FORALL x : time;
  HENCEFORTH (
    SENDING OF tsap[1].T_DATAreq AT x LEADSTO
    RECEIVING OF tsap[2].T_DATAind AT (now <= x+5));
```

Specifying the restriction this way, a new problem emerges which is due to the missing relationship between a sent and a received packet. The above specification only states that 5 units time after sending a packet another packet has to be received. It does not state that the received packet is the one sent. Again, this problem can be solved using the instance operator by relating the z-th sent to the z-th received packet:

```
...
  SENDING OF tsap[1].T_DATAreq[z] AT x LEADSTO
  RECEIVING OF tsap[2].T_DATAind[z] AT (now <= x+5)
...
```

However, even with this formulation, not all problems are solved. Consider an unreliable service where not every packet sent is received at the other site. Using more complex Real-Time Estelle expressions (e.g. containing the `OTHERWISE` construct) helps specifying such situations. We do not further elaborate on this and omit such situations in the examples to keep them simple.

Maximum computation time in the transport protocol. To restrict the time which a data packet may remain inside a protocol instance (local delay), the following expression is useful:

```
NAME service-processing-time:
FORALL x : time;
  HENCEFORTH (
    RECEIVING OF xtp_ip.T_DATAreq AT x LEADSTO
    SENDING OF medium_ip.M_DATAreq AT (now < x+5));
```

This kind of information may be very useful for operating system schedulers with respect to priorities, necessary computation time etc. The satisfiability of such restrictions largely depends on the availability of local resources such as the CPU.

Connection setup delay. The reaction of a service provider to a connection setup request should occur within 5 units time. However, this reaction may also be negative.

```
NAME connection-establishment-delay:
FORALL x : time;
  HENCEFORTH (
    SENDING OF tsap[1].T_CONNreq AT x LEADSTO
      (RECEPTION OF tsap[1].T_CONNcnf OR
       RECEPTION OF tsap[1].T_DISind) AT (now <= x+5));
```

5.1.3 Jitter

Global jitter. The term jitter describes a variance in delay. Consider the case where data packets are transferred from a sender to a receiver. The packets are all arriving with a certain delay. Specifying a maximal jitter restricts the differences between these delays to a certain amount. Jitter may be specified by the transport user in the following way:

```
NAME jitter-global:
FORALL x : time;
  HENCEFORTH (
    (SENDING OF tsap[1].T_DATAreq AT x FORBIDS
        RECEPTION OF tsap[2].T_DATAind AT (now < x+5)) AND
    (SENDING OF tsap[1].T_DATAreq AT x LEADSTO
        RECEPTION OF tsap[2].T_DATAind AT (now <= x+7)));
```

In general, a relationship between the sent and received packet should be established using the instance operator.

In the above restriction, the reception of the data packet is forbidden for the interval $[0;5]$ after sending. Within the next two units time, the packet has to be received. If the functional specification part ensures that double packet delivery is impossible, the jitter restriction my be simplified using the instance operator:

```
...
  SENDING OF tsap[1].T_DATAreq[z] AT x LEADSTO
  RECEPTION OF tsap[2].T_DATAind[z] AT (x+5 <= now <= x+7)
...
```

In the case of possible double delivery, the reception of the original packet during the interval $[0;5]$ and of the duplicate during $]5;7]$ would be allowed.

Global jitter restrictions are difficult to implement. At least two nodes of a distributed system are concerned by this kind of restriction. Thus, the jitter problem can no longer be solved locally. Many protocol architectures, however, assume that the jitter problem has to be solved locally anyway. The next paragraph handles local jitter.

Local jitter. Speaking of local jitter, only the packet arrival times at the receiver site are taken into account. The jitter is 0 if the inter-arrival time between packets is always the same; the greater the fluctuation, the greater is the jitter. The restriction to a certain maximum local jitter may be written in Real-Time Estelle as:

```
NAME jitter-local:
FORALL x : time;
  HENCEFORTH (
    (RECEPTION OF tsap[2].T_DATAind AT x FORBIDS
       RECEPTION OF tsap[2].T_DATAind AT (x < now < x+2)) AND
    (RECEPTION OF tsap[2].T_DATAind AT x LEADSTO
       RECEPTION OF tsap[2].T_DATAind AT (now <= x+3)));
```

Isochronous Sending. Especially for the transfer of continuous media such as audio and video, it is desirable to send consecutive data units with the same time gap. The following Real-Time Estelle restriction describes this requirement for the XTP module:

```
NAME isochronous-send:
FORALL x : time;
  HENCEFORTH (
    (SENDING OF medium_ip.M_DATAreq AT x FORBIDS
       SENDING OF medium_ip.M_DATAreq At (x < now < x+5)) AND
    (SENDING OF medium_ip.M_DATAreq AT x LEADSTO
       SENDING OF medium_ip.M_DATAreq AT (now = x+5)));
```

5.1.4 QoS mechanisms.

QoS negotiation and renegotiation. QoS (re-) negotiation usually results in a variation of an existing QoS restriction. This fact is best modeled by describing current QoS values in variables instead of constants. The value of this variable is then changed when QoS has been renegotiated.

Let us take as an example a variable data rate. Let *min_rate* be such a variable of the XTP module which contains the current minimum packet rate per second. The timescale used in this specification is milliseconds. Thus, the maximum time between to consecutive data packets computes to $\frac{min\_rate}{1000}$, i.e., with 2000 packets per second, a packet will be send at least every 2 ms. In Real-Time Estelle, we write:

```
NAME changing_rate:
FORALL x : time;
  HENCEFORTH (
    SENDING OF medium_ip.M_DATAreq AT x LEADSTO
    SENDING OF medium_ip.M_DATAreq AT (x < now <= x + min_rate/1000));
```

QoS violation. All of the above QoS restrictions imply that a system really behaves as it is specified. If, for example, the transport service contains a restriction that every connection has to be set up within 5 units time, then this restriction has to be satisfied by every implementation. Otherwise, it does not conform to the specification. For many cases, this is unrealistic. A solution to this problem consists in offering the system a way out which allows a reaction on such a QoS violation. This way out will typically be of the form "if restriction x is not satisfied, then do y." In Real-Time Estelle, the key word `OTHERWISE` may be used to specify such reactions. Look at the transport connection setup as an example. Receiving a setup request, the transport system should send a confirmation or a reject within 5 units time. If this is impossible, then after 6 units time, it sends a violation message to the service user.

```
NAME qos-violation:
FORALL x : time;
  HENCEFORTH (
    RECEPTION OF tsap[1].T_CONNreq AT x LEADSTO (
      (SENDING OF tsap[1].T_CONNcnf OR SENDING OF tsap[1].T_DISind)
          AT (now <= x+5))
    OTHERWISE
      SENDING OF tsap[1].T_QOSVIOLind AT (now = x+6));
```

The advantage of using the `OTHERWISE` construct instead of the semantically equivalent `OR` is that it is possible to make a difference between these alternatives based on the syntax. For automatic implementations and the attempt to guarantee QoS requirements, this possibility is of major importance.

The possibility of a reaction to a QoS violation also has to be specified within the functional specification part. Otherwise, the automaton would not be able to produce timed state sequences which contain such reactions, leading to an inconsistent overall specification. Thus, a module containing such reactions should be equipped with transitions similar to the following:

```
TRANS
    FROM idle TO pending
       WHEN tsap[1].T_CONNreq begin
          output medium_ip.M_CONNreq;
       end;

    FROM pending
       WHEN medium_ip[1].M_CONNcnf TO connected begin
          output tsap[1].T_CONNcnf;
       end;

       delay(5) TO idle begin
```

```
        output tsap[1].T_QOSVIOLind;
    end;
```

In principle, using the `delay` clause in this example would not be necessary since non-functional behavior is already specified in the Real-Time Estelle restriction. However, using this clause makes the intention of the specifier more obvious. It should be noted that an enabled delayed transition is disabled as soon as another transition is executed [10]. Especially when the executed transition does not change the enabling conditions of the delayed transition, this is not obvious. The clarification allows the use this method even in the case where the major state is not changed, e.g., when transferring huge amounts of data. The clarified semantics of the clause [10] guarantees that the delay timer is restarted every time.

5.2 Examples for the need of time variables

All the examples given above are similar in that timed reference points of state observations are frozen in a variable, and another state is related to the first. For such applications, a simpler syntax with implicit reference points as it is provided e.g. by MTL would have been sufficent. In MTL, the bounded-response property "Always when p is observable, q has to be observable within three units time." is expressed by the formula $\Box(p \to \Diamond_{\leq 3} q)$, while it reads in Real-Time Estelle:

```
FORALL x: time;
HENCEFORTH (p AT x LEADSTO q AT (now <= x+3));
```

The MTL specification is obviously shorter. However, such languages have the disadvantage of not being able to express so-called *non-local* properties (at least not always) and absolute times.

Non-local restrictions. We call properties non-local which relate more than two time instants resp. state observations. A typical example (given in [4]) is: "Always when p occurs, q will later be observable, and after q, r will be observable within 5 units time after p." The MTL formula $\Box(p \to \Diamond(q \wedge \Diamond_{\leq 5} r))$ does not do the job, since the reference point of r is not p, but q. If we use discrete time domains (with dense time, this is impossible), the correct way of expressing this restriction is

$$\begin{aligned}\Box(p \to (\quad & \Diamond_{=0}(q \wedge \Diamond_{\leq 5} r) \vee \Diamond_{=1}(q \wedge \Diamond_{\leq 4} r) \vee \\ & \Diamond_{=2}(q \wedge \Diamond_{\leq 3} r) \vee \Diamond_{=3}(q \wedge \Diamond_{\leq 2} r) \vee \\ & \Diamond_{=4}(q \wedge \Diamond_{\leq 1} r) \vee \Diamond_{=5}(q \wedge \Diamond_{\leq 0} r)),\end{aligned}$$

but Real-Time Estelle is capable to express this much more elegantly:

```
FORALL x: time;
HENCEFORTH (
   p AT x LEADSTO (q AND EVENTUALLY (r AT now <= x+5)));
```

Conveyed to multimedia systems, the following restriction is non-local: "Always when a user starts a movie, than he receives an acknowledgement, and afterwards, the film starts within 5 units time after his request." In Real-Time Estelle, this reads:

```
FORALL x : time;
HENCEFORTH (
  SENDING OF PLAYreq AT x LEADSTO
       (RECEIVING OF PLAYcnf AND
        EVENTUALLY firstframe AT (now <= x+5)));
```

Absolute times. Using languages with implicit time references, absolute time restrictions such as "State p will be observable on June 13th, 1996." are impossible. Real-Time Estelle, however, is capable to express such restrictions. Consider the following *Near-Video-on-Demand* system: movie transmissions will not start immediately after a user's request, but only at 6, 8 and 10pm. All movie requests which arrive at least 5 minutes before the movie transmission time will be serviced. For the 8pm movie, the following restrictions apply[2]:

```
FORALL x : time;
  (MOVIEreq AT x and (17:55 <= x < 19:55)) LEADSTO startMovie AT 20:00;

FORALL x : time;
  (MOVIEreq AT x AND (x > 19:55)) FORBIDS startMovie AT x < now < 22:00;
```

The term `AT x AND` may be left out, since `x` is implicitly determined by `now`.

5.3 A Timed Transition System

In Section 4.2, a TTS semantics for Estelle was introduced. Using an example, this section shows how to model TTS in Real-Time Estelle. The modeled example is that of Figure 2 and is specified using one Estelle module. We just give the transition and time constraints parts.

```
TRANS
   FROM s TO t
      PROVIDED c begin end;
```

[2]To improve readability, times are not given as integers.

```
   FROM t TO u
      PROVIDED a=5 begin end;

   FROM u TO v
      WHEN uip.DATAreq begin end;

   FROM v TO s begin end;


TIME CONSTRAINTS
   NAME s_to_t:
   FORALL x : time;
     HENCEFORTH (
       ((MAJOR_STATE(s) and PROVIDED(c) AT x) FORBIDS
           MAJOR_STATE(t) AT (now <x+1)) AND
       ((MAJOR_STATE(s) and PROVIDED(c) AT x) LEADSTO
           MAJOR_STATE(t) AT (now <= x+2)));

   NAME t_to_u:
   FORALL x : time;
     HENCEFORTH (
       (MAJOR_STATE(t) and PROVIDED(a=5) AT x) LEADSTO
          MAJOR_STATE(u) AT (now <= x+1));

   NAME u_to_v:
   FORALL x : time;
      HENCEFORTH (
       ((MAJOR_STATE(u) and WHEN(uip.DATAreq) AT x) FORBIDS
          MAJOR_STATE(v) AT (now < x+2)) AND
       ((MAJOR_STATE(u) and WHEN(uip.DATAreq) AT x) LEADSTO
          MAJOR_STATE(v) AT (now <= 10)));

   NAME v_to_s:
   FORALL x : time;
     HENCEFORTH (
       MAJOR_STATE(v) AT x FORBIDS
          MAJOR_STATE(s) AT (now < x+1));
```

Obviously, the conditions of the time restrictions and the transitions are the same which makes it possible to use the TTS semantics. In addition, a translation from this notation to those described in [9] and [13] can be easily performed[3]. Compared to those approaches,

[3]Small restrictions have to respected. In our approach, it is for example impossible restrict the time for system management phases.

Real-Time Estelle produces quite a huge overhead, due to the fact that enabling conditions of transitions have to be given twice. However, it has been shown that such a semantics for Real-Time Estelle is possible and useful. On the other hand, QoS restrictions may not or only in a very restricted way expressed in the other approaches.

6 Implementation

Implementing specifications of real-time systems is impossible without a *real-time environment*. In this section, we show how real-time requirements specified in Real-Time Estelle may be guaranteed using a *real-time operating system (rt-os)*. We only concentrate on local requirements, since global requirements such as an end-to-end delay need more than just an rt-os. Thus, we assume the existance of e.g. a network with real-time capabilities such as isochronous data transfer.

Using an rt-os, it is possible to reserve one of the most important resources necessary for a guaranteed local performance: cpu time. If a process has asked for a certain amount of cpu time, and this amount has been assigned, then the rt-os does everything to guarantee this reservation. Especially, new processes will not be assigned the cpu resource if already running processes would be disturbed.

6.1 Prerequisites

In [15] we showed that a major obstacle for efficient implementations of Estelle specifications is the complex parent-child synchronization. Real-time systems and efficient implementations are strongly related. If, e.g., a timer expires in a module which should lead to an immediate reaction of that module, then it may be impossible to run this module because the module's subsystem is currently executing another set of transitions. In fact, the semantics allows implementations which are capable of fulfilling such timing requirements, but it becomes very difficult to generate such implementations.

This knowledge also has an impact on the specifier's technique. Knowing that the implementation will only be efficient if there is very little synchronization, he/she will try to write specifications consisting of many system modules and having a very flat hierarchy. However, such specifications will often not express the complex relationships between system parts.

Therefore, it seems to be important to adapt the synchronization semantics for time critical applications. The new semantics should be developed with respect to the following requirements:

1. There should be a way to have parallel modules which do not influence each other. The result would be a greater flexibility with respect to parallelism.

2. The position of a module within a hierarchy should have no impact on its performance. The position should only be a means for structuring of specifications.

In fact, an Estelle enhancement already exists which exactly provides such features. Bredereke and Gotzhein suggested to use so-called *asynchronous process modules* which are independant of their parent modules [7]. The new keyword `asynchronous` is provided by which `process` and `systemprocess` modules may be additionally attributed. Making a module `asynchronous` means that it is no longer synchronized with its child modules. If a module itself and its parent module are both asynchronous, then the module has not to be synchronized with any other module. Its behavior is that of system modules, with the difference, that such modules may still be dynamically created or released.

Some experiments presented in [15] clearly indicate that with this Estelle enhancements, much better implementations may be produced. For the following considerations, we use this technique.

6.2 Real-time operating systems and multimedia systems

Real-time operating systems are very useful for the implementation of multimedia systems. One of the main characteristics of continuous media data streams is that the single parts of the stream — audio samples or video frames — are only valid during a certain time interval. If a video should be displayed with a rate of 30 frames per second, then every frame has a validity interval of $\frac{1}{30}s$. If a frame cannot be displayed during this interval, it is not useful anymore, since then, the next frame will already be displayed. It is the task of an rt-os to guarantee that each frame is handled during its validity phase[4]. It does this basically by reserving the resource cpu time.

The model for the cpu management of an rt-os is as follows: given m tasks and n processors where each task is determined by its computation time C, its deadline d and its period T (see Fig. 6). Find a task execution on the given set of processors such that every task gets C units time of computation time in each period and meets its deadline. Such an execution is called *real-time* scheduling.

Normally, it suffices that the end of the period and the the deadline are the same. In this case, a task may simply be described by period and computation time. It is important to note that in this model, only periodic tasks are discussed. The reason is that it is quite difficult to schedule aperiodic or sporadic tasks [32]. In reality, this is no real restriction, since sporadic tasks may be mapped to periodic tasks with one period [30]. In addition, periodic tasks are a good abstraction for continous data streams. Such streams usually consist of periodically arriving data units. In one period of the task, one data unit is handled.

Another important issue is whether tasks may be preempted or not. It is often more difficult to schedule non-preemptive tasks, since this may lead to the well-known problem of priority inversion.

[4]As stated above, the ability of an rt-so to fulfill this task is heavily influenced by other system parts such as the network. If the network is not able to provide data with the correct rate, then an rt-os cannot fix this.

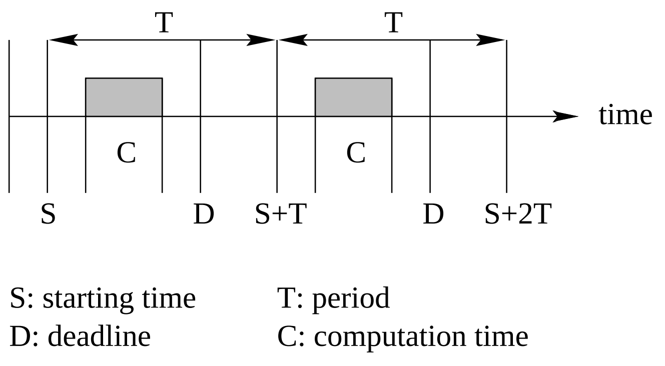

Figure 6: Model of a real-time task

Most of the results summarized here are only valid for the single-processor case. For multiprocessors, most of the algorithms are NP-comlete. However,there are also some heuristics which allow real-time scheduling on multiprocessors [39].

The assignment of cpu time to a task is managed by a *scheduling algorithm*. A set of task is schedulable, if the scheduling algorithm can execute all these tasks while guaranteeing their computation times and periods. New tasks will only be accepted by the algorithm, if the new set of task is still schedulable.

In real-time systems, basically two algorithms are used which have proved their usefulness. Both work on preemptive tasks:

- The *static* Rate-Monotonic algorithm (RM) assigns to each task in a task set a static priority depending on their period. The smaller the period, the higher the priority. When a task with higher priority than that of the currently executed becomes active, then the current task is preempted, and the cpu is assigned to the higher order task. After the latter has finished, the preempted task continues executing.

  There is a simple test, if a set of tasks is schedulable. If a new task arrives, than in the worst case, the following condition must hold: $\sum_{i=1}^{m+1} \frac{C_i}{T_i} \leq 0,69$, i.e. the processor usage of all tasks must be less than 69%. If this is not the case, the new task will be rejected. In the average case, the bound is about 80%.

- The *dynamic* Earliest-Deadline-First algorithm (EDF) assigns the highest priority to the task whose deadline is reached next. This leads to a dynamic priority adjustment. EDF is able to schedule both periodic and sporadic tasks. The schedulability test of EDF is: $\sum_{i=1}^{m+1} \frac{C_i}{T_i} \leq 1$.

The functionality of both algorithms is shown in Figure 7. The digitally-numbered stream has a short period and represents e.g. an audio stream, while the alphabetically-numbered stream has a longer period and represents a video stream.

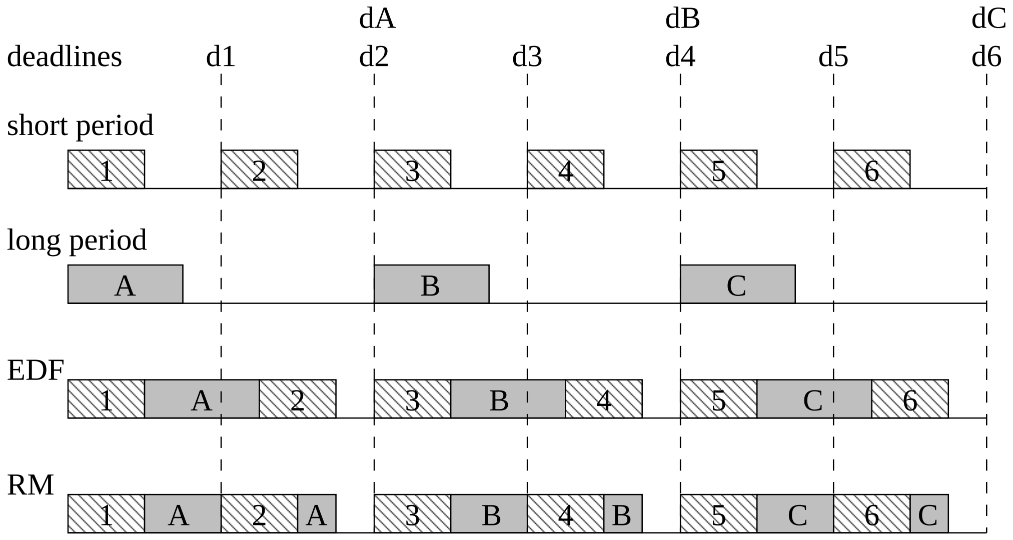

Figure 7: Rate-Monotonic- and Earliest-Deadline-First scheduling

Both algorithms are optimal in their class, i.e., there is no static resp. dynamic algorithm which can schedule a task set which is not also schedulable by RM resp. EDF. However, some improvemements for the basic algorithms have been achieved, e.g. [27, 5, 29, 46].

The EDF algorithm seems to be better than RM since it is obviously able to schedule more task sets. However, dynamic priority assignment can be quite time-consuming, and this overhead is not taken into account in the schedulability test. A problem which arises for both algorithms consists in a varying computation time as it for example typical for MPEG videos. In such a case, the longest computation time has to be reserved, and for those periods where the computation time is smaller, the processor remains partly unused [41]. In practice, however, both algorithms have proved useful for multimedia systems.

In the last few years, several rt-os have been developed. One of them is *Real-Time Mach* [44]. Many projects showed that this operating system is well-suited to support multimedia applications (e.g. [17, 30, 21]). RT Mach provides real-time threads which are able to implement real-time tasks.

In the remainder of this section, we show how a variant of RT Mach can be used to implement Real-Time Estelle specifications.

6.3 Mapping Real-Time Estelle restrictions onto real-time tasks

Time restrictions in Real-Time Estelle are specified within modules. The goal of using an rt-os is to assign to a module enough cpu time to satisfy its time restrictions. Thus, for an rt-os, a Real-Time Estelle module is a real-time task. For simplicity's sake, we only investigate periodic tasks which are, however, a good abstraction for multimedia tasks. Problems of handling sporadic tasks are discussed in Section 6.6.

The handling of periodically arriving data units is modeled best, in an Estelle module, by a repeatedly executed transition (or a sequence of transitions). The sequence is started by a message read from an interaction point. Some or more transitions will process this message, and finally, a new message will be sent to another (or the same) interaction point. The period of this module is determined by the time between the sending (or the reception) of two consecutive messages, and the computation time is the time between reading an incoming message and sending out the new message. This model is visualized in Figure 8.

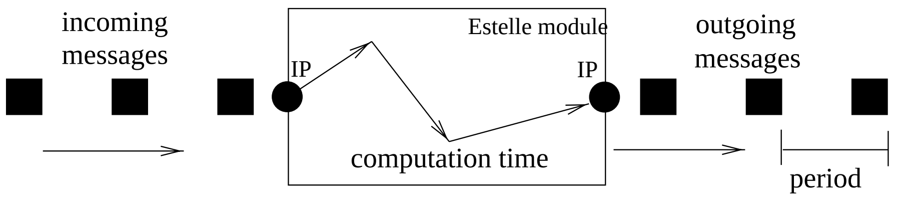

Figure 8: Period and computation time in Real-Time Estelle

The necessary period depends on the application. A video module should be able to process 25 frames per second, i.e., the period is $\frac{1}{25}s$.

It is less simple to determine the computation time. It cannot be derived directly from the application or the specification, since it largely depends on the processor speed. A solution for this problem is to produce the software without timing constraints, run it on the implementation machine and perform measurements for the interseting operations. The measured values may then be used to write the timing constraint for the computation time in Real-Time Estelle. In [47], such a measurement tool was developed, and some experiments showed that computation times may be measured very exactly.

We now have two timing contraints. One of them describes the period and the other one the computation time of a real-time task. The next step is to execute Real-Time Estelle modules in a concrete real-time operating system. We discuss this using a variant of Real-Time Mach.

6.4 Implementation issues of a Real-Time Estelle Compiler

As a basis for the implementation of Real-Time Estelle specifications, we will use Real-Time Mach. This operating systems provides, as shortly described above, real-time threads which are capable to execute real-time tasks. When initialized, a real-time thread gets as parameters a pointer to the function to be executed periodically, and the period and computation time. For the latter two, only one constant value may be specified by the user which makes simple real-time threads a little unflexible for Real-Time Estelle restrictions. As we know, in this language, it is possible to give restrictions in form of intervals. Therefore, we do not use pure Real-Time Mach, but an enhancement [21] which offers so-called *QThreads*. Qthreads are characterized by the fact that for period and computation time, both a lower and upper bound may be given. The operating system is free to chose one value in the interval for the actual scheduling. During runtime, if there occur any problems, the value may be dynamically adjusted within the bounds of the interval and without notification of the user.

QThreads are initialized similarly to real-time threads. A QThread can only be initialized if the resulting set of threads is still schedulable.

An important decision is related to the mapping of modules to threads. Modules with temporal restrictions need their thread *exclusively* to be able to fulfill the restrictions. Suppose two such modules to be mapped onto one thread. Then, a common period and computation time could be computed, but the thread now has only knowledge of these global restrictions. It does not “konw” that in fact, it has to fulfill two restrictions. Thus, a guarantee cannot be given for both restrictions.

In [16], we showed that grouping of modules in threads instead of assigning one thread to each module usually leads to much better resource usage levels and better performance. Due to the considerations above, this *configuration* approach cannot be applied to modules equipped with real-time restrictions. Thus, for a multiprocessor system, we suggest the following approach of module execution: the set of available processors is divided in two subsets. On the first subset, all real-time restricted modules are executed using QThreads which are scheduled using the EDF or RM algorithms. On the other subset, all other modules are executed according to the configuration approach described in [16].

Functionality of a Real-Time Estelle compiler. Compared to existing tools, a Real-Time Estelle compiler additionally has to consider the temporal restrictions during the code generation process. From these restrictions, it has to derive the period and computation times for the QThreads. Thus, the compiler has first to be informed which restriction determines which parameter. This information should be provided using qualified comments since it seems quite difficult to let the tool find out. Both restrictions should have the form `FORALL x:time; HENCEFORTH (p AT X LEADSTO q AT (x+c <= now <= x+d);`. For the period restriction, `p` and `q` should be identical, possibly except for the instance operator. Considering these restrictions, the compiler may check the temporal restrictions for plausability. If both restrictions have the correct form, the constants used in the interval containing `now` are used as lower and upper bounds for the QThread's period and computation time, respectively.

Functionality of the Real-Time Estelle runtime system. The central function of the runtime system's real-time part is the `init` function, implementing Estelle's `init`. If a module has to be initialized which contains temporal restrictions, then a new QThread will be generated. Otherwise, the module will be assigned to an existing non-real-time thread and will be configured for execution. In C, this function reads:

```
void init(Module m)
{
   .... /* variables etc.  */
   if (m.restricted) {
     qthread_attribute_init(
          m.exec, m.arguments,
          m.lower_period, m.upper_period,
          m.lower_comp_time, m.upper_comp_time,
          ....);
     qthread_create(...);
   } else {
     add_module_to_thread(m,thread);
   }
   ...
}
```

The decision, whether a module resp. a QThread is schedulable already has to be made in an Estelle function which may be used inside a `provided` clause or an `if` statement inside the transition containing the `init` statement. If this test was part of the `init` function and the result was negative, than it would be impossible to respect this result in Estelle, since a call to `init` is always successful. The module has to be produced, even though a QThread must not be generated. The following primitive function `module_schedulable` contains the schedulability test:

```
boolean module_schedulable(unsigned int new_period, new_comptime)
{
   .... /* variable declarations */

   for (i=0;i<m;i++)  /* m=number of running tasks */
      total_rate = total_rate + comp_time[i]/period[i];

   if ((total_rate + new_comptime/new_period) <= 1) {
      m++;
      comp_time[m] = new_comptime;
      period[m]    = new_period;
      return TRUE;
   } else
      return FALSE;
}
```

Violation of real-time guarantees. Theoretically, by using a real-time operating system, it can be guaranteed that real-time restrictions are fulfilled. However, due to exceptions such as network or processor failure, violations of these restrictions may possibly occur. When the QThread library detects such a violation, it first tries to adjust period and computation times of the running threads within the interval boundaries. If this is impossible, the Estelle runtime system receives error messages from the library. These messages are passed to the user by means of the function `qos_violated`. Now, the user has to decide how to react on a violation. He could for example release the module. To allow for violations, they should already be foreseen in the specification, using the `OTHERWISE` construct.

6.5 An Example

The following example models a simple multimedia application in Real-Time Estelle. The application consists of a user module, a connection manager and a transport system. The connection manager has three further modules for the handling of video and audio streams and for text. A user may ask the connection manager to set up a new data stream. The manager tries to initialize the corresponding module. This will only be possible, if the temporal restrictions can be fulfilled. Temporal restrictions exist for video and audio module as well as for the transport module. All other modules do not have any temporal restrictions. The module structure is depicted in Figure 9.

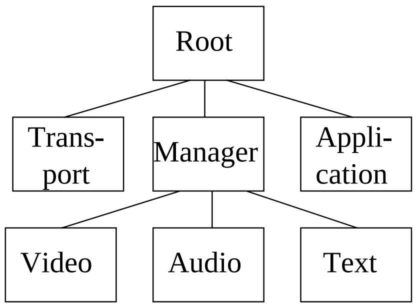

Figure 9: Strukture of Real-Time Estelle implementation example

The central transitions of the manager module handle incoming user requests. For an audio stream, this looks as follows:

```
TRANS
   when sap.INIT-AUDIOreq
   provided module_schedulable(period[audio_m_body],
                               comptime[audio_m_body])
   begin
      init audio[i] with audio_module_body;
      output sap.INIT_AUDIOcnf(positive)
   end;

   provided OTEHRWISE begin
      output sap.INIT-AUDIOcnf(negative)
   end;
```

The audio module itself contains two temporal restrictions which model period and computation time. The central transition gets data from the transport system and forwards the audio samples contained in this data to a player. A violation of the period restriction is possible since an `OTHERWISE` part exists:

```
TRANS
    from sending
    when tsap.T-DATAind(t-data-packet)
    provided not qos_violated to same
    begin
        extractSamples(t-data-packet,samples);
        output player.AUDIO_SAMPLES(samples);
    end;

    provided OTHERWISE to waiting
    begin
        output control.QOSVIOLind;
    end;

TIME CONSTRAINTS
    FORALL x : time;
    NAME period:
       SENDING OF player.AUDIO_SAMPLES AT x LEADSTO
           SENDING OF player.AUDIO_SAMPLES AT (x+20 <= now <= x+30)
    OTHERWISE
       SENDING OF QOSVIOLind;

    FORALL x : time;
    NAME computation_time:
       RECEIVING OF tsap.T-DATAind AT x LEADSTO
           SENDING OF player.AUDIO_SAMPLES AT (x+4 <= now <= x+6);
```

Let's have a look at a typical situation: the manager module has already accepted a video module with a period between 40 and 50 ms (current value: 40) and computation time between 10 and 15 ms (15). In addition, there is a running audio module with a period between 20 and 30 ms (20) and a computation time between 4 and 6 ms (6). As scheduling algorithm we use RM. The resulting processor usage is $\frac{15}{40} + \frac{6}{20} = 67.5\%$. Now, we want a new video module with the same characteristics as the first one to be accepted. We suppose it has the lowest priority. The schedulability test tells us that the new module may not be scheduled, even not with the lowest parameter values. The already running video module has the second lowest priority. Using the lowest values within the specified bounds for this module, we obtain a processor usage of $\frac{6}{20} + 2 \times \frac{10}{50} = 0.7$ which is still greater than 69%. To take no risk (69% is a very pessimistic bound), we also decrease the computation time of the audio module by 1 ms. Now, all three modules are schedulable.

6.6 Implementation Problems

The implementation process suggested above certainly covers many possible and important temporal restrictions, but not all possible. Aperiodic or sporadic modules may be scheduled by EDF [41], but not by RM.

Another problem consists in Estelle modules which contain both kinds of restrictions. A mapping has to be found that fulfills both kinds. An example for such a pair of restriction is a constrained connection setup time and a certain period.

7 Conclusion and Outlook

In this paper, we described an enhancement to the formal description technique Estelle to allow the specification of real-time propositions of protocols or systems. Both formal syntax and semantics were provided. We showed that this enhancement is especially well-suited for the specification of quality-of-service characteristics and requirements. We also gave first hints on how an implementation environment for Real-Time Estelle could look like.

One of major future tasks is to realize this implementation environment. It would also be interesting to study non-local restrictions such as end-to-end delay with respect to their implementability. Another important task is to provide really powerful Estelle implementation tools which produce very efficient runtime code. Such tools will make it much easier to fulfill local time restrictions for many modules.

A Syntax Definition of Real-Time Estelle

This appendix contains the syntax definition of Real-Time Estelle in Backus-Naur Form. It complements the the syntax of Estelle given in Appendix A.1 of the standard. Some of the definitions given there are used here. These are e.g. `expression`, `relational-operator` or `body-definition`. The latter is the only syntax rule which has been slightly adapted by adding the `TIME CONSTRAINTS` part.

The following meta symbols have been used in accordance with the Estelle standard:

Meta symbol	meaning
$=$	shall be defined to be
$\vert$	alternatively
.	end of definition
$[x]$	0 or 1 instance of x
$\{x\}$	0 or more instances of x
$+\{x\}$	1 or more instances of x
$(x\vert y)$	grouping: either x or y
$x\Psi y$ $a_1\Psi...\Psi a_n$	$xy\vert yx$ all possible strings consisting of all the elements concatenated in an arbitrary order
"xyz"	the terminal symbol xyz

Real-Time Estelle is defined as follows:

body-definition = declaration-part
initialization-part
transition-declaartion-part
time-constraint-declaration-part.

event = IDENTIFIER.

exists-clause = "EXISTS" +{variable-declaration ";"}.

forall-clause = "FORALL" +{variable-declaration ";"}.

logic-operator = "AND"| "OR" | "IMPLIES" | "OTHERWISE" |
"EVENTUALLY" | "HENCEFORTH" | "LEADSTO" |
"FORBIDS".

name-clause = "NAME" IDENTIFIER ":".

simple-state-description = "MAJOR_STATE(" IDENTIFIER ")"
| "WHEN(" IDENTIFIER ")"
| "PROVIDED(" IDENTIFIER ")"
| "RECEIVING OF" event ["[" IDENTIFIER "]"]
| "SENDING OF" event ["[" IDENTIFIER "]"]
| expression.

```
state-description =       simple-state-description
                       |  state-description logic-operator state-description
                       |  timed-state-description
                       |  "NOT" state-description
                       |  "(" state-description ")".

tc-clause-block =         { forall-clause }
                     Ψ    { exists-clause }
                     Ψ    { name-clause }.

tc-list =   +{ time-constraint }.

time-constant   =   IDENTIFIER.

time-constraint =   [ tc-clause-block ] time-constraint-block.

time-constraint-block =   time-constraint-expression.

time-constraint-declaration-part =   [ "TIME CONSTRAINTS" tc-list ].

time-constraint-expression   =   state-description.

time-expression =        time-variable
                     |   time-constant
                     |   time-variable "+" time-constant
                     |   time-expression relational-operator time-expression
                     |   "(" time-expression ")".

time-variable   =   IDENTIFIER.

timed-state-description   =   state-description "AT"
                              time-expression.
```

References

[1] M. Abadi and L. Lamport. An Old-Fashioned Recipe for Real Time. In de Bakker et al. [12], pages 1–27.

[2] R. Alur and D. Dill. The Theory of Timed Automata. In de Bakker et al. [12], pages 43–73.

[3] R. Alur and T. Henzinger. Logics and models of real time: A survey. In de Bakker et al. [12], pages 74–106.

[4] R. Alur and T. A. Henzinger. Real-time Logics: Complexity and Expressiveness. In *Proceedings of the Fifth Annual IEEE Symposium on Logic in Computer Science*, pages 390–401, 1990.

[5] I. Barth. Extending the Rate-Monotonic Scheduling Algorithm to Get Shorter Delays. In Steinmetz [40], pages 104–114.

[6] H. Bowman, G. Blair, L. Blair, and A. Chetwynd. Time versus abstraction in formal descriptions. In Tenney et al. [43], pages 467–482.

[7] J. Bredereke and R. Gotzhein. Increasing the Concurrency in Estelle. In Tenney et al. [43], pages 127–141.

[8] S. Budkowski, B. Alkhechi, M. L. Benalycherif, P. Dembiński, M. Gardie, E. Lallet, J. P. Mouchel La Fosse, and Y. Souissi. Formal specification, validation and performance evaluation of the Xpress Transfer Protocol. In A. Danthine, G. Leduc, and P. Wolper, editors, *Protocol Specification, Testing and Verfication XIII.* Elsevier Science Publishers B.V. (North–Holland), Amsterdam, 1993.

[9] S. C. Chamberlain. *Estelle Enhancements for Formally Specifying Distributed Systems.* PhD thesis, University of Delaware, USA, 1992.

[10] J.-P. Courtiat. A Petri Net Based Semantics for Estelle. In M. Diaz, J.-P. Ansart, J.-P. Courtiat, P. Azema, and V. Chari, editors, *The Formal Description Technique Estelle*, pages 135–174. Elsevier Science Publishers B.V. (North–Holland), Amsterdam, 1989.

[11] J.-P. Courtiat and R. C. de Oliveira. RT-LOTOS and its application to multimedia protocol specification and validation. In B. Sarikaya and S. Saito, editors, *IEEE International Conference on Multimedia Networking (MmNet95), Participants' Proceeedings*, pages 31–45. IEEE Computer Society Press, Sept. 1995.

[12] J. W. de Bakker, C. Huizing, W. P. de Roever, and G. Rozenberg, editors. *Real-Time: Theory in Practice (LNCS 600).* Springer–Verlag Berlin Heidelberg New York, 1991.

[13] P. Dembiński and S. Budkowski. Simulating Estelle specifications with time parameters. In Rudin and West [37], pages 265–279.

[14] P. Dembiński and M. Średniawa, editors. *Protocol Specification, Testing and Verfication XV.* Chapman & Hall, London, 1995.

[15] S. Fischer. On the Suitability of Estelle for Multimedia Systems. In Dembiński and Średniawa [14], pages 369–384.

[16] S. Fischer and W. Effelsberg. Efficient Configuration of Protocol Software for Multiprocessors. In R. Puigjaner, editor, *High Performance Networking VI*, pages 195–210. Chapman & Hall, London, Sept. 1995.

[17] R. Gopalakrishna and G. M. Parulkar. Efficient Quality of Service Support in Multimedia Computer Opertaing Systems. Technical Report WUCS-94-26, Washington University St. Louis, USA, 1994.

[18] R. Gotzhein. Specifying Communication Services with Temporal Logic. In L. Logrippo, R. L. Probert, and H. Ural, editors, *Protocol Specification, Testing and Verification X*, pages 295–309. Elsevier Science Publishers B.V. (North–Holland), Amsterdam, 1990.

[19] R. Gotzhein. Temporal logic and applications – a tutorial. *Computer Networks and ISDN Systems*, 24:203–218, 1992.

[20] T. A. Henzinger, Z. Manna, and A. Pnueli. Timed Transition Systems. In de Bakker et al. [12], pages 226–251.

[21] K. Kawachiya and H. Tokuda. QOS-Ticket: A New Resource-Management Mechanism for Dynamic QOS Control of Multilmedia. In *Multimedia Japan'96*, 1996.

[22] R. Koymans. Specifying Real-Time Properties with Metric Temporal Logic. *Real-Time Systems Journal*, 2(4):255–299, Nov. 1990.

[23] L. Lamport. Proving the Correctness of Multiprocess Programs. *IEEE Transactions on Software Engineering*, SE-3(2):125–143, 1977.

[24] L. Léonard and G. Leduc. An enhanced version of timed LOTOS and its application to a case study. In R. Tenney, P. Amer, and M. Uyar, editors, *Formal Description Techniques VI*, pages 483–498. Elsevier Science Publishers B.V. (North–Holland), Amsterdam, 1994.

[25] S. Leue. Specifying Real-Time Requirements for SDL Specifications — A Temporal Logic-Based Approach. In Dembiński and Średniawa [14], pages 19–34.

[26] T. D. C. Little and A. Ghafoor. Synchronization and storage models for multimedia objects. *IEEE Journal on Selected Areas in Communication*, 8(3):52–61, Apr. 1990.

[27] J. W. S. Liu, K. J. Lin, and S. Naturajan. Scheduling Real-Time Periodic Jobs Using Imprecise Results. In *IEEE Real-Time Systems Symposium*, pages 252–260. IEEE Computer Society Press, 1987.

[28] N. Lynch and F. Vaandrager. Forward and Backward Simulation for Timing-Based Systems. In de Bakker et al. [12], pages 397–446.

[29] A. Mauthe and G. Coulson. Scheduling and Admission Testing for Jitter Constrained Periodic Threads. In R. Gusella and T. D. C. Little, editors, *Network and Operating System Support for Digital Audio and Video (NOSSDAV'95)*, pages 219–226, Apr. 1995.

[30] C. W. Mercer, S. Savage, and H. Tokuda. Processor Capacity Reserves: Operating System Support for Multimedia Applications. In L. Belady, S. M. Stevens, and R. Steinmetz, editors, *IEEE International Conference on Multimedia Computing and Systems*, pages 90–99. IEEE Computer Society Press, 1994.

[31] P. M. Merlin and D. J. Farber. Recoverability of Communication Protocols – Implication of a theoretical Study. *IEEE Transactions on Communications*, Com-24:1046–1043, Sept. 1976.

[32] A. K. Mok. *Fundamental Design Problems of Distributed Systems for the Hard Real-Time Environment.* PhD thesis, MIT, 1993.

[33] X. Nicollin and J. Sifakis. An Overview and Synthesis of Timed Process Algebras. In de Bakker et al. [12], pages 526–548.

[34] J. S. Ostroff. *Temporal Logic of Real-time Systems.* Research Studies Press, 1990.

[35] J. Quemada and A. Fernandez. Introduction of Quantitative Relative Time into LOTOS. In Rudin and West [37], pages 105–121.

[36] H. Rudin. The dimension of Time in Protocol Specification. In *Lecture Notes in Computer Science 248*, pages 360–372. Springer–Verlag Berlin Heidelberg New York, 1986.

[37] H. Rudin and C. H. West, editors. *Protocol Specification, Testing and Verfication VII.* Elsevier Science Publishers B.V. (North–Holland), Amsterdam, 1987.

[38] P. Sénac, M. Diaz, and P. de Saqui-Sannes. Toward a formal specification of multimedia synchronization scenarios. *Annuaires Télécommunication*, 49(5–6):297–314, 1994.

[39] J. A. Stankovic, M. Spuri, M. D. Natale, and G. C. Buttazzo. Implications of Classical Scheduling Results for Real-Time Systems. *IEEE Computer*, 28(6):16–25, June 1995.

[40] R. Steinmetz, editor. *Multimedia: Advanced Teleservices and High-Speed Communication Architectures (LNCS 868).* Springer–Verlag Berlin Heidelberg New York, 1994.

[41] R. Steinmetz. Analyzing the Multimedia Operating System. *IEEE MultiMedia*, 2(1):68–84, Spring 1995.

[42] W. T. Strayer, B. J. Dempsey, and A. C. Weaver. *XTP The Xpress Transfer Protocol.* Addison–Wesley, Reading, Massachusetts, 1992.

[43] R. L. Tenney, P. D. Amer, and M. Ü. Uyar, editors. *Formal Description Techniques, VI.* Elsevier Science Publishers B.V. (North–Holland), Amsterdam, 1994.

[44] H. Tokuda, T. Nakajima, and P. Rao. Real-Time Mach: Towards Predictable Real-Time Systems. In *USENIX 1990 Mach Workshop*, Oct. 1990.

[45] G. von Bochmann and J. Vaucher. Adding Performance Aspects to Specification Languages. In S. Aggarwal and K. Sabnani, editors, *Protocol Specification, Testing and Verification VIII*, pages 19–29. Elsevier Science Publishers B.V. (North–Holland), Amsterdam, 1988.

[46] J. Werner and L. C. Wolf. Scheduling Mechanisms Reducing Contention Situations in Multimedia Systems. In B. Butscher and E. Moeller, editors, *European Workshop on Interactive Distributed Multimedia Systems and Services*, Mar. 1996.

[47] H. Wittig, L. C. Wolf, and C. Vogt. CPU Utilization of Multimedia Processes: HeiPOET – The Heidelberg Predictor of Execution Times Measurement Tool. In Steinmetz [40], pages 92–103.