19

Implementation of multimedia systems based on a real-time extension of Estelle

*S. Fischer**
University of Montréal, Département d'IRO
C.P. 6128, succ. centre-ville, Montréal (Québec) H3C 3J7, CANADA,
Email: stefis@iro.umontreal.ca

*This work was done by the author at the University of Mannheim, Germany, with funding by the DFG research grant Ef 14/1-2.

Abstract

Estelle is one of the standardized Formal Description Techniques for the specification of communication protocols and distributed systems. Unfortunately, Estelle is not capable of expressing real-time requirements resp. characteristics of services or protocols which are especially interesting in the context of distributed multimedia systems. Therefore, we developed an extension to Estelle called Real-Time Estelle that allows the description of real-time systems. In this paper, we first give a short introduction into Real-Time Estelle and compare it to earlier approaches to the inclusioin of real time into FDTs. In the main part, we discuss possibilities to implement such specifications in a manner guaranteeing QoS requirements by using a real-time operating system. An analysis shows that time bounds can be guaranteed not only for single Estelle module instances but also for module groups where single modules communicate asynchronously.

Keywords

Estelle, Real Time, FDT-based Implementation, Multimedia Systems

1 INTRODUCTION

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

While the use of Formal Description Techniques (FDTs) has proved useful for the

R. Gotzhein et al. (eds.), *Formal Description Techniques IX*

specification of traditional protocols and distributed systems, most of them lack a real-time concept. Thus, it is often impossible to express quantitative QoS requirements based on a formal syntax and semantics.

Estelle (ISO9074, 1989) is one of the standardized Formal Techniques for the specification of protocols and distributed systems. Estelle bears many advantages. Since the language is a superset of the programming language Pascal, Estelle specifications are easy to understand, and allow straight-forward derivation of simulations and efficient implementations.

In (Fischer, 1995) however, we showed that due to its weak time model Estelle is not very suitable for the specification of real-time constraints. Thus, we developed a real-time extension of Estelle called *Real-Time Estelle*. In this paper, we show how Real-Time Estelle can be used for the support of the implementation process, i.e., how implementations can be produced automatically from a formal specification that *adhere to the real-time restrictions* given in the specification.

The rest of this paper is organized as follows: Section 2 presents related work, i.e. some earlier approaches for introducing real time in FDTs. In Section 3, we give a short overview of Real-Time Estelle, though the focus of this paper is not on the language itself. This section should rather give an impression of how the language looks and serve as a base for the understanding of the implementation part. Section 4, as the main part, is concerned with the derivation of implementations for multimedia systems based on certain QoS requirements formulated in Real-Time Estelle. Section 5 analyzes properties of the presented implementation environment and shows that even for module groups, in which single modules communicate asynchronously, QoS guarantees can be provided. Section 6 concludes the paper.

Due to space limitations and the focus of this paper on implementation, the introduction into Real-Time Estelle will be brief. For a more thorough language description including formal syntax and semantics, a technical report is electronically available (Fischer, 1996).

2 RELATED WORK

In this section, we will give a short overview on earlier approaches to inclusion of real-time into FDTs.

Basic models. Nearly all of the existing models used to represent the behavior or characteristics of systems — constructive techniques such as automata, Petri nets, process algebra and descriptive techniques such as all kinds of logics — have been enriched by means to express real-time.

The work of (Alur and Dill, 1991), (Henzinger et al., 1991), (Rudin, 1986) and (Lynch and Vaandrager, 1991) 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., 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.

There has also been much and early work in the area of timed Petri Nets, e.g. *Time Petri*

Nets (Merlin and Farber, 1976), *Object Composition Petri Nets* (Little and Ghafoor, 1990) and *Time Stream Petri Nets* (Sénac et al., 1994). Especially Little and Ghafoor 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 (Nicollin and Sifakis, 1991), though development has since progressed.

Descriptive techniques are usually based on *temporal logic* (TL), for an overview see (Gotzhein, 1992)). 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) (Koymans, 1990) or Quality of Service Temporal Logic QTL (Bowman et al., 1994), or explicit timer variables as in Real-Time Temporal Logic (RTTL) (Ostroff, 1990) or TLA (Abadi and Lamport, 1991). Most of them are based on linear time and use natural numbers as their time domain.

Higher 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 (Quemada and Fernandez, 1987; Léonard and Leduc, 1994; Courtiat and de Oliveira, 1995). 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 (Dembiński and Budkowski, 1987; von Bochmann and Vaucher, 1988; Chamberlain, 1992), but none of the methods described above is addressed explicitly there. Dembiński and Budkowski go in the direction of TTSs by adding upper and lower bounds on the execution time of transitions. Chamberlain effectively does the same by adding a new clause — `doby(x)` — to transitions, implying a hard upper time bound for the transition execution once it has been enabled. The approach of von Bochmann and Vaucher is more complex, adding more performance-related constructs such as resources and probabilities to the language.

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., 1994) describe this for LOTOS in conjunction with QTL and (Leue, 1995) for SDL with MTL. We will employ this approach for the definition of Real-Time Estelle.

Suitability for QoS specification. We note that for most QoS parameters, a purely operational semantics, where temporal requirements can only be related to single transitions (as e.g. in Timed Transition Systems), is not sufficient. The analysis of typical QoS requirements (see (Fischer, 1996)) reveals, that most of them refer to more than one transition or even to more than one automaton. Logical formulas are much better suited to express such requirements. In addition, the intention of the specifier is usually easier to be understood.

3 REAL-TIME ESTELLE

Real-Time Estelle is an extension of standard Estelle. The basic idea is to enhance the transition model of Estelle with a time component and then restrict the occurrence of certain states in time. Therefore, a new section called `TIME CONSTRAINTS` may be added to a module body description. Here are collected all temporal requirements resp. restrictions concerning that module. The extension is defined with respect to the model of Estelle and its syntax and semantics.

Real-time Model. The most common model for real-time systems are *timed state sequences*. Each sequence identifies one possible behavior of the system. An element of a timed state sequence consists of a state description s_i and a time instant t_i. Each state s_i can be characterized by atomic propositions which are true in that state. The component t_i indicates the time instant at which the system starts to be in state s_i.

To provide a similar real-time model for Estelle, the Estelle state-transition model has to be checked for its compatibility with timed state sequences. Estelle specifications describe sets of *state sequences* (so-called *computations*). A new element is added to a state sequence by executing a module's transition or a system management phase. To "upgrade" this model to the real-time model outlined above, we have to (1) define by which atomic propositions an Estelle state should be characterized, and (2) add a time component to the sequences:

1. The Estelle standard already defines the components of a system state (Sections 5.3 and 9.4), e.g., value of local variables, major state of a module or contents of queues. To these state components, we add predicates `SENDING OF (p.m)` and `RECEIVING OF (p.m)` which are true when message `m` has been sent (output-statement) over resp. received (when-clause) at interaction point `p` during the last transition. Such predicates already proved very useful for QoS specifications (Leue, 1995). The new *instance operator* `[]` allows counting of messages sent or received at a certain IP.
2. Time is added to Estelle's state sequences by adding just one component to the state description. This *time component* is a positive integer variable, and the only restriction for its value is the following: given two subsequent states s_i and s_{i+1} of one sequence, with time components t_i and t_{i+1}, then $t_i \leq t_{i+1}$, i.e., if time changes then it increases.

With these definitions, the underlying model of Estelle is now a real-time model.

Syntax of Real-Time Estelle restrictions. The basic building blocks of temporal restrictions in Real-Time Estelle are *state descriptions*. The following rules define correct state descriptions:

1. If `p` is an atomic state 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.

The intended meaning of the keywords is intuitively clear. The expression `p OTHERWISE q` is equivalent to `p OR q`, but can be used to express a kind of preference for certain

behaviors. This may be useful for execution environments to find out the desired behavior and support it, but still leave a way out if the behavior cannot be supported. Semantically, however, both constructs are equivalent.

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, i.e., refer to the time component of the timed state sequences described above. 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 expression (rigid variables). Values of `now` can be "stored" in time variables and referenced in other parts of the expression. The type `time` is defined as `TYPE time=integer`, and the time domain is given by the `timescale` option. Time variables occurring in a Real-Time Estelle expression have to be quantified by an `EXISTS` or `FORALL` clause preceding the expression.

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 determine the occurrence of states in the future, which is a central concept for the specification of temporal relationships between states, 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 of time after `p`: `p AND x=now IMPLIES EVENTUALLY (q AND now <= x+3)` (omitting quantifiers).

In addition to the operators described above, some abbreviations may be used which make temporal restrictions more readable. They are defined with respect to the existing operators: the expression `p AND x=now` may be replaced by `p AT x`. For `p IMPLIES EVENTUALLY q`, one may write `p LEADSTO q`, and `p IMPLIES HENCEFORTH NOT q` may be substituted by `p FORBIDS q`.

Semantics. The semantics we use for Real-Time Estelle is called "hybrid", since it contains both operational and descriptive parts. The operational parts arise from standard Estelle, while the Real-Time Estelle part determines the descriptive part. The overall specification consists of timed state sequences constructed as follows: first, timed state sequences are constructed using the operational semantics described in the Estelle Standard and the real-time model extensions given above. Then, these sequences are used as models for the temporal logic formulas described by the Real-Time Estelle restrictions. Only those sequences which satisfy all formulas are part of the overall real-time system. A satisfaction relation $\models$ exists which defines the conditions timed state sequences must meet to fulfill Real-Time Estelle expressions (Fischer, 1996). The semantics is that of a first-order temporal logic with time variables and is similar to that of *Real-Time Temporal Logic (RTTL)* (Ostroff, 1990).

Specification Example. To get an impression of how QoS requirements may be specified in Real-Time Estelle, consider the following formula, expressing a maximum allowed time a message may rest within a module instance:

```
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));
```

The formula expresses that, whenever a T_DATAreq is received at the interaction point xtp_ip, then a message M_DATAreq is sent over the interaction point medium_ip within 5 units of time.

4 DERIVING IMPLEMENTATIONS

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

4.1 Prerequisites

In (Bredereke and Gotzhein, 1994) and (Fischer, 1995), it was shown that a major obstacle to efficient implementations of Estelle specifications is the complex parent-child synchronization. Real-time systems and efficient implementations are strongly related. If a timer expires in a module, which should lead to an immediate reaction by that module, then it may be impossible to run this module because the module's subsystem is currently executing another set of transitions. Though, in fact, the semantics allows implementations which are capable of fulfilling such timing requirements, it becomes very difficult to produce such implementations.

This knowledge also affects 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 fail to express the complex relationships between system parts.

Therefore, it is important to adapt the synchronization semantics for time critical applications. A solution to the problem already exists: Bredereke and Gotzhein suggested to use so-called *asynchronous process modules* (Bredereke and Gotzhein, 1994). By assigning the new keyword asynchronous to a module and its parent, this module can execute without having to synchronize with its parent or child modules. Such modules may thus be scheduled when necessary, independently of any other module. Some experiments presented in (Fischer, 1995) clearly indicate that this Estelle enhancement enables much faster implementations. For the following considerations, we use this technique.

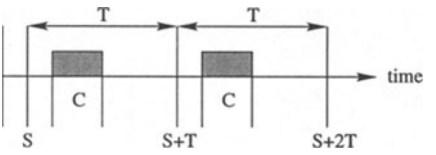

Figure 1 Model of a real-time task

4.2 Real-time operating systems and multimedia systems

Real-time operating systems are very useful in the implementation of multimedia systems. One of the main characteristics of continuous media data streams presented to a human user 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 at a rate of 25 frames per second, then every frame has a validity interval of $\frac{1}{25}s$. If a frame cannot be displayed during this interval, it is no longer useful, 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. It does this basically by reserving the resource *cpu time*.

The model for the cpu management of an rt-os is as follows: given are m tasks and n processors where each task is determined by its start time S, its computation time C and its period T (see Fig. 1). Find a task execution on the given set of processors such that every task gets C units of computation time in each period.

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 (Mok, 1993). In reality, this is no real restriction, since sporadic tasks may be mapped to periodic tasks with one period (Mercer et al., 1994). In addition, periodic tasks are a good abstraction for continuous 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.

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

In real-time systems, basically two algorithms are commonly used: the *Rate-Monotonic (RM)* and the *Earliest-Deadline-First* algorithms. RM uses static priorities; the task with the shortest deadline has the highest priority. In EDF, priorities are dynamic: the task with the earliest deadline always gets the highest priority. For both algorithms, a simple schedulability test exists. RM can schedule a given set of tasks if their total processor usage

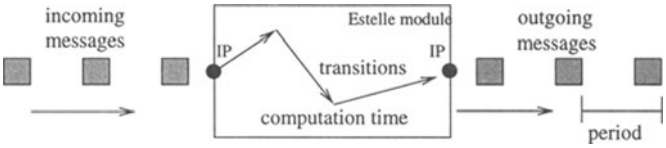

Figure 2 Period and computation time in Real-Time Estelle

is less than 69%. For EDF, this bound is 100%. Thus, the EDF algorithm seems superior to 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 is for example typical of MPEG videos. In such a case, the longest computation time has to be reserved, and for those periods where the computation time is shorter, the processor remains partly unused (Steinmetz, 1995). In practice, however, both algorithms have proved useful when rt-os have been used to implement multimedia systems (e.g. (Mercer et al., 1994; Kawachiya and Tokuda, 1996)).

4.3 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.

In an Estelle module, the handling of periodically arriving data units is best modeled by a repeatedly executed transition or a sequence of transitions. The sequence is started by a message read from an interaction point. One or more transitions will process this message, and finally, a new message will be sent to another (or to the same) interaction point. The period of this module is determined by the time between the sending (or the receipt) of two consecutive messages, and the computation time is that between reading an incoming message and sending out the new one. This model is visualized in Figure 2.

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

Less simple to determine is the computation time. It cannot be derived directly from the application or the specification, since it largely depends on the processor speed. A solution to this problem is to produce the software without timing constraints, run it on the implementation machine and perform measurements for the interesting operations. The measured values may then be used to write in Real-Time Estelle the timing constraint for the computation time. In (Wittig et al., 1994), 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 is thus a

throughput requirement. The other one defines the computation time of a real-time task and can be seen as a local delay requirement. The next step is to execute Real-Time Estelle modules as tasks of a concrete real-time operating system.

4.4 Design of a Real-Time Estelle Compiler and Runtime System

As a basis for the implementation of Real-Time Estelle specifications, we will use Real-Time Mach. This operating system provides *real-time threads* which represent real-time tasks. When initialized, a real-time thread gets as parameters a pointer to the function to be executed periodically, the period and the computation time. For the latter two, only one constant value may be specified by the user. This makes simple real-time threads a little inflexible for Real-Time Estelle restrictions, since here it is possible to give restrictions in form of intervals. Therefore, we do not use pure Real-Time Mach, but an enhancement thereof (Kawachiya and Tokuda, 1996) 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 choose one value in the interval for the actual scheduling. During runtime, if any problems occur, the value may be dynamically adjusted within the bounds of the interval and without notifying 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 will be mapped onto one thread. Then, a common period and computation time could be computed, but the thread would now have only knowledge of these global restrictions. It would not "know" that in fact, it has to fulfill two restrictions. Thus, a guarantee could not be given for either restriction.

In (Fischer and Effelsberg, 1995), we showed that grouping modules in threads instead of assigning one thread to each module usually delivers 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 to module execution: the set of available processors is divided into 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 (Fischer and Effelsberg, 1995).

Functionality of a Real-Time Estelle compiler. Compared to existing compiler tools (e.g. (Budkowski, 1992; Sijelmassi and Strausser, 1993)), 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

†Most of the results for real-time scheduling are only valid for the single-processor case. For multiprocessors, most of the algorithms are NP-complete. However,there are also some heuristics which allow real-time scheduling on multiprocessors (Stankovic et al., 1995; Hamidzadeh and Lilja, 1996).

QThreads. Thus, the compiler must first be informed which restriction determines which parameter. This information should be provided using qualified comments since it seems quite difficult to leave that to the tool. 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 plausibility. If both restrictions are in the correct form, the constants used in the interval containing `now` will be 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 containing temporal restrictions has to be initialized, 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. The C code of 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_non_real_time_thread(m,thread);
}
```

The decision as to whether a module resp. a QThread is schedulable must already 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 were part of the `init` function and the result were negative, then it would be impossible to respect this result in Estelle, since in Estelle, an `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 for RM:

```
boolean module_schedulable(unsigned int new_period, new_comptime)
{  .... /* variable declarations; m=number of running tasks */
   if ((total_processor_usage + new_comptime/new_period) <= sched_bound) {
      m++; total_processor_usage += new_comptime/new_period;
      comp_time[m] = new_comptime;  period[m] = new_period;
      return TRUE;
   } else  return FALSE;
}
```

Violation of real-time guarantees. Theoretically, using a real-time operating system can guarantee fulfillment of real-time restrictions. 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, it produces error

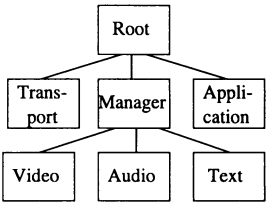

Figure 3 Structure of the sample Real-Time Estelle specification

messages for the Estelle runtime system which are passed to the user by means of the function `qos_violated`. The user then has to decide how to react to 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.

4.5 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 of text. A user may ask the connection manager to set up a new data stream. The manager tries to initialize the corresponding module. This can only be done, if the temporal restrictions can be fulfilled. Temporal restrictions exist for video and audio modules as well as for the transport module. No other modules have any temporal restrictions. The module structure is depicted in Figure 3.

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
      audio_module_counter = audio_module_counter + 1;
      init audio[audio_module_counter] with audio_module_body;
      output sap.INIT_AUDIOcnf(positive)
   end;

   provided OTHERWISE 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 for-

wards the audio samples contained in this data to a player. A violation of the period restriction is allowed since an OTHERWISE part exists:

```
TRANS
    from sending
    when tsap.T-DATAind(data-packet)
    provided not qos_violated to same
    begin
        extractSamples(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:
    HENCEFORTH (SENDING OF player.AUDIO_SAMPLES AT x LEADSTO
       SENDING OF player.AUDIO_SAMPLES AT (x+20 <= now <= x+30)
    OTHERWISE
       SENDING OF QOSVIOLind AT (x+30 < now) );

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

Consider 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, not even with the lowest parameter values. The video module already running 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 pessimistic bound), we also decrease the computation time parameter of the audio module by 1 ms. Now, all three modules are schedulable, without any user interaction being necessary.

5 ANALYSIS AND DISCUSSION

The considerations presented above are always concerned with guaranteeing restrictions of exactly one module implementing one protocol instance. In reality, however, the real-

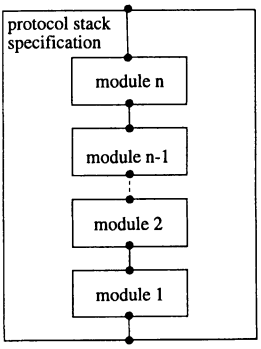

Figure 4 Structure of a multi-module protocol stack specification

time behavior of the whole underlying protocol stack, usually modelled by more than one module in Estelle, might be more interesting. Investigating such a module structure means, that we will have to deal with Estelle's asynchronous communication via *infinite message queues*. This leads to two questions:

- Is there an upper bound for the waiting time of messages in queues, i.e., can we give a maximum processing time of the protocol stack?
- Is there always a message available when a module of the stack is assigned the processor? If this is not the case, then the module is using a part of its computation time without doing protocol work, making estimations of period and data rate bounds difficult.

In the following, we will discuss these question using the module structure of Fig. 4 as an example, where the elements of a data stream are subsequently handled by module n, $(n-1)$, etc. We assume that each submodule has a specified period p_i and a computation time c_i.

Since we want to achieve a certain data rate for the whole stack and thus a total period of p_0 (one packet is handled in one period), each submodule also needs a period of p_0. It doesn't make sense to assign a higher rate to one of the modules, since there will not be more packets available for protocol processing due to the lower rate of the other modules (assuming no packet segmentation). This means that all threads executing one of the modules will be assigned the same priority, both in RM and EDF scheduling, which in turn means that none of these threads can interrupt one of the others. Practically, the result is a non-preemptive scheduling, and a possible execution sequence may look like that of Fig. 5.

Due to these two characteristics — all threads have the same period and they won't be interrupted when executing — it is assured that each module outputs a packet before it

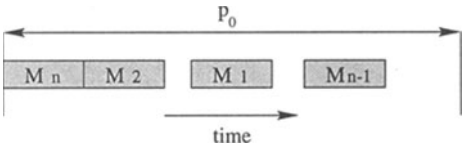

Figure 5 A sample real-time module execution

passes the processor to the next module. In turn, this means that each module will find a packet in its message queue when it gets the processor (not considering the stream's starting phase, i.e., when the first packets enter the stack). This fact is *independent of the module execution order*, which may be easily seen by examining the ways of packets through the module group of Fig. 4 using the execution sequence of Fig. 5. Consequently, each message queue contains a maximum of one message, and one queue contains exactly one message when its module starts execution. Thus, no processor time is wasted since no module is running idle, resulting in the fact that the total period and data rate can be maintained. If module n receives a packet from the user in every period, then module 1 outputs a packet in every period (after a certain startup delay).

We can also give an upper bound for the protocol stack's processing time for a given packet. This bound, however, depends on the module execution sequence. Assuming an optimal execution sequence n, $(n-1)$, ..., 2, 1, we obtain no additional waiting times, since each packet output by module i will be processed immediately afterwards by module $(i-1)$. Assuming the sequence of Fig. 5, the total computation time consists of the modules' computation times plus the waiting time in the queue of module $(n-1)$ while modules 2 and 1 are executing, plus the waiting time in the queue of module 2 while module n is executing. In the worst case — the sequence 1, 2, ..., $(n-1)$, n —, the stack's computation time is given by the formula $c_{stack} = c_1 + (n-1)\sum_{1}^{n} c_i$, which can be easily proved by induction.

As a result, we can guarantee for such a group of modules a certain period (and thus a data rate) and a certain computation time (and thus a maximum delay introduced by the protocol stack), despite the fact that these modules communicate via infinite message queues.

6 CONCLUSION AND OUTLOOK

Real-Time Estelle is an extension of Estelle allowing the specification of real-time restrictions for Estelle modules. In this paper, we described an idea of how to generate implementations of such real-time systems based on their formal specification and adhering to the given timing requirements. The basic idea is to use a real-time operating system and map real-time restricted modules onto real-time threads of the operating system. The

discussion revealed that with this method, real-time guarantees can be given not only for single modules, but also for whole module groups modelling, e.g., a protocol stack. Thus, it is possible to control Estelle's communication via infinite message queues.

Certainly, the current environment only supports very special real-time restrictions, i.e., it is possible to bound the time a message remains inside a module and the period between two subsequent messages output by a module. However, in reality, these are very common restrictions (modelling throughput and delay of a local protocol instance or a stack) and many applications may be covered by them.

One of the major future tasks is to enhance the implementation environment in order to cover more QoS requirements. It would be interesting to study non-local restrictions such as end-to-end delay with respect to their implementability. Another important task is to develop really powerful Estelle implementation tools which produce very efficient runtime code. Such tools will make it much easier to fulfill local time restrictions for numerous modules.

7 ACKNOWLDEGEMENTS

The author wishes to thank Wolfgang Effelsberg, Reinhard Gotzhein, Jan Bredereke and Stefan Leue for fruitful discussions on the specification of real-time requirements and for comments on an earlier version of this paper. Also, the comments of the anonymous reviewers have led to significant improvements.

REFERENCES

Abadi, M. and Lamport, L. (1991). An Old-Fashioned Recipe for Real Time. In (de Bakker et al., 1991), pages 1–27.

Alur, R. and Dill, D. (1991). The Theory of Timed Automata. In (de Bakker et al., 1991), pages 43–73.

Bowman, H., Blair, G., Blair, L., and Chetwynd, A. (1994). Time versus abstraction in formal descriptions. In (Tenney et al., 1994), pages 467–482.

Bredereke, J. and Gotzhein, R. (1994). Increasing the Concurrency in Estelle. In (Tenney et al., 1994), pages 127–141.

Budkowski, S. (1992). Estelle Development Toolset. *Computer Networks and ISDN Systems, Special Issue on FDT Concepts and Tools*, 25(1):63–81.

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

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

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

York.

Dembiński, P. and Budkowski, S. (1987). Simulating Estelle specifications with time parameters. In (Rudin and West, 1987), pages 265–279.

Dembiński, P. and Średniawa, M., editors (1995). *Protocol Specification, Testing and Verification XV*. Chapman & Hall, London.

Fischer, S. (1995). On the Suitability of Estelle for Multimedia Systems. In (Dembiński and Średniawa, 1995), pages 369–384.

Fischer, S. (1996). Real-Time Estelle. Technical Report TR-96-003, University of Mannheim. URL=ftp://pi4.informatik.uni-mannheim.de/pub/techreports/1996/tr-96-003.ps.gz.

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

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

Hamidzadeh, B. and Lilja, D. (1996). Dynamic Scheduling Strategies for Shared-memory Multiprocessors. In Buckles, B. and Chanson, S., editors, *IEEE Int. Conf. on Distributed Computing Systems, Hongkong*. IEEE Computer Society Press. To appear.

Henzinger, T. A., Manna, Z., and Pnueli, A. (1991). Timed Transition Systems. In (de Bakker et al., 1991), pages 226–251.

ISO9074 (1989). Information processing systems — Open Systems Interconnection — Estelle: A formal description technique based on an extended state transition model. International Standard ISO 9074.

Kawachiya, K. and Tokuda, H. (1996). Dynamic QOS Control Based on the QOS-Ticket Model. In *IEEE Int. Conference on Multimedia Computing and Systems (ICMCS'96), Hiroshima, Japan*. IEEE Computer Society Press.

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

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

Leue, S. (1995). Specifying Real-Time Requirements for SDL Specifications — A Temporal Logic-Based Approach. In (Dembiński and Średniawa, 1995), pages 19–34.

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

Lynch, N. and Vaandrager, F. (1991). Forward and Backward Simulation for Timing-Based Systems. In (de Bakker et al., 1991), pages 397–446.

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

Merlin, P. M. and Farber, D. J. (1976). Recoverability of Communication Protocols – Implication of a theoretical Study. *IEEE Transactions on Communications*, Com-

24:1046–1043.

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

Nicollin, X. and Sifakis, J. (1991). An Overview and Synthesis of Timed Process Algebras. In (de Bakker et al., 1991), pages 526–548.

Ostroff, J. S. (1990). *Temporal Logic of Real-time Systems.* Research Studies Press.

Quemada, J. and Fernandez, A. (1987). Introduction of Quantitative Relative Time into LOTOS. In (Rudin and West, 1987), pages 105–121.

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

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

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

Sijelmassi, R. and Strausser, B. (1993). The PET and DINGO tools for deriving distributed implementations from Estelle. *Computer Networks and ISDN Systems*, 25(7):841–851.

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

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

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

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

Wittig, H., Wolf, L. C., and Vogt, C. (1994). CPU Utilization of Multimedia Processes: HeiPOET – The Heidelberg Predictor of Execution Times Measurement Tool. In Steinmetz, R., editor, *Multimedia: Advanced Teleservices and High-Speed Communication Architectures (LNCS 868)*, pages 92–103. Springer–Verlag Berlin Heidelberg New York.

8 BIOGRAPHY

Stefan Fischer received his diploma in Computer Science Applied to Business Administration and his doctoral degree in Computer Science from the University of Mannheim, Germany, in 1992 and 1996, respectively. From 1992 to 1996, he was a research assistant at the Institute of Applied Computer Science (Department for Computer Networks) of the University of Mannheim, headed by Wolfgang Effelsberg. Since September 1996, he is working as a postdoctoral fellow at the University of Montral, Canada, in the group of Gregor von Bochmann. His current research interests include formal description techniques for real-time and multimedia systems as well as implementation techniques for high-speed protocols and applications.