Formal Methods for Broadband and Multimedia Systems

Stefan Fischer [a]

[a] *University of Montréal, DIRO, C.P. 6128, succ. Centre-Ville, Montréal, (PQ) H3C 3J7, CANADA, Email: fischer@iro.umontreal.ca*

Stefan Leue [b]

[b] *Electrical and Computer Engineering, University of Waterloo, Waterloo, Ontario N2L 3G1, CANADA, Email: sleue@swen.uwaterloo.ca*

The proper capture of desired system properties is a pivotal step in providing high quality systems. The formal specification of these properties is necessary to provide unambiguous documentation as well as automated transformation of system requirements during all stages of the life cycle. The standardized Formal Description Techniques (FDTs) Estelle and SDL have proved useful for the specification of traditional protocols and distributed systems. With the availability of high-speed networks new applications with additional requirements and characteristics are becoming reality. These requirements are often referred to as *Quality of Service* (QoS) requirements. We show that the above mentioned FDTs do not possess the expressiveness to capture important classes of QoS requirements, namely quantitative deterministic real-time-related properties. It is the purpose of this paper to exemplify steps that need to be taken in order to overcome this deficit.
We first discuss choices that need to be made when designing a suitable real-time execution model for SDL and Estelle and proceed to present two remedies to the inexpressiveness problem: First, we introduce the concept of complementary real-time specification by reconciling the semantic models of Metric Temporal Logic and SDL and showing how both languages can be used in a complementary fashion. Second, we suggest a language extension and the corresponding semantic interpretation for Estelle. While we present examples from the domain of multimedia and broadband systems, the applicability of our specification methods extends to hard real-time systems. Finally, we discuss extensions of our techniques to capture QoS stochastic properties, and we allude to formal requirements verification and automatic implementation based on our techniques.

Condensed version to appear in Computer Networks and ISDN Systems. 21 August 1997

1 Introduction

The specification of requirements on the observable behavior of distributed, communicating real-time system is an important step in the engineering of these systems. Requirements specifications help avoiding inconsistencies in the requirements, they are the basis for deriving correct system designs, they are essential in establishing the system's correctness by serving as a basis for testing and formal verification, and they are important in the documentation of system requirements. [20] suggests that proper requirements engineering, of which requirements specification is an important step, is pivotal in avoiding pitfalls of what is called the 'software crisis'. Facets of the software crisis include systems that have low reliability or even unusability at time of delivery to the customer, and software projects that exceed budget limits and project deadlines. [20] argues that the cost of repairing a bug due to a falsely stated requirement at the maintenance stage of the system's life cycle is about 200 times higher than the cost of repair at the requirements specification stage.

The complexity of software projects in the area of embedded real-time and telecommunications systems calls for automated tool support. This can only be provided if the underlying engineering methods have a formal foundation. Formal Description Techniques (FDTs) like Estelle [38] and SDL [40] have been successfully applied to the specification of 'traditional' communication protocols, services and network applications. With the deployment of high-speed networks such as ATM [27] new distributed applications with a new set of characteristics have emerged. This leads to new requirements on the communication subsystems: while in traditional systems the notion of system correctness is defined in terms of the functional *correct ordering* of externally observable events, the new type of systems is deemed correct in case it abides to the functional correctness as well as a set of quantitative characteristics, frequently called *quantitative Quality of Service (QoS)* characteristics. We conjecture that the high-speed and multimedia systems that are currently being developed are largely software driven, which is why our methods are relevant to the engineering of these systems.

Distributed multimedia systems are an important example for the emerging class of applications. For these systems requirements on a single media stream (e.g. throughput, delay and delay jitter) and on multiple streams (e.g. stream synchronization) have to be taken into account. These requirements are largely related to timing aspects in the system, which is why we limit the discussion of QoS characteristics to real-time related properties. The relevance of QoS requirements is not restricted to services and protocols in individual layers of the communication subsystem. QoS requirements also apply to end-to-end connections and to application-user interfaces, and our specification method shall be general enough to accommodate all of these application areas.

In this paper we will focus on the standardized FDTs Estelle and SDL that enjoy a high degree of acceptance within the telecommunications engineering community. Both are based on the paradigm of extended, communicating finite state machines. Both have textual syntaxes and semi-formally defined standardized semantics. In addition, SDL has a graphical syntax and it enjoys support by a wide range of industrial-strength CASE tools. It is therefore of great importance that the various QoS requirements can be adequately expressed in the chosen formalism, and hence that these formalisms have an adequate expressiveness to account for real-time phenomena.

The standardized versions of both techniques provide asynchronous timer-based real-time concepts. As we will show these concepts are not suitable to express the real-time related QoS characteristics that we alluded to. We shall therefore develop extended notations and their formal semantics that allow for expressing the required characteristics. For Estelle we define a real-time language extension (called *Real-Time Estelle*), while we show for SDL how SDL specifications can be complemented by real-time temporal logic formulas (we call this approach *SDL/MTL*).

Paper organization: In Section 2 we survey related work, especially approaches towards extending formal specification techniques by real-time expressiveness. In Section 3 we introduce the QoS-related terminology, in particular as far as the relevant ISO standard is concerned, and identify 'typical' real-time related QoS parameters for various components of broadband and multimedia systems. In Section 4 we argue for the inexpressiveness of standard Estelle and SDL with respect to real-time properties. We next present our approaches to adding real-time expressiveness to both languages. We first introduce a general state-transition based model for timed systems in Section 5. SDL/MTL will then be presented in Section 6, followed by Real-Time Estelle in Section 7. To show the applicability of both approaches to QoS requirement specification we give examples in Section 8. Finally, Section 9 concludes the paper with a comparison of the approaches, perspectives for verification, automatic implementation and the incorporation of stochastic expressiveness.

Precursors of this work have appeared in [26], which introduces Real-Time Estelle, and in [52,53] in which the SDL based approach was first presented.

2 Related Work

Basic models. Many of the existing models used to specify functional system behavior – operational techniques such as automata, Petri nets, process algebra, and descriptive techniques like logics – have been enriched by means to express non-functional real-time properties. The work of [3], [35], [64] and [57] is based on variants of timed automata. Time restrictions are introduced by labeling transitions or states of extended finite state machines with time limits, clocks and time variables. An upper bound u and a lower bound l is assigned to each transition of the timed automaton. Once enabled, the transition may be executed not sooner than l and not later than u time units after the enabling. Similar conditions for the enabling of transitions can be formulated referring to the values of time variables. These are model-theoretic techniques in that they distinguish all execution sequences of a system into those that satisfy the timing constraints (the "good" ones), and those that do not satisfy them (the "bad" ones). Only those systems that can only reveal "good" execution sequences implement the specification. Similar timed extensions have been defined for Petri Nets, see *Time Petri Nets* [59], *Object Composition Petri Nets* [54] and *Time Stream Petri Nets* [67]. There have also been numerous real-time extensions to process algebras, see for example [61] and [63,51,19]. A process algebra-based QoS specification language based on the FDT LOTOS has been proposed in [9–11]. We note that as of the time of writing state-transition based specification methods dominate in the area of practical telecommunications systems engineering, which is why we adhere to these techniques and do not pursue Petri net and process algebra based approaches.

Temporal logics are the descriptive counterpart to specifying state transition systems by automata [58,47,28]. Temporal logics specify qualitative temporal relationships between states. A program satisfies a temporal logic specifications if all of its execution state sequences satisfy these temporal relations. As can be expected, extensions have been introduced to augment temporal logics with constructs that specify quantitative real-time relations between states. Examples are Metric Temporal Logic (MTL) [44] and Quality of Service Temporal Logic QTL [9–11] that use real-time interval annotations to the temporal operators, and techniques introducing explicit timer variables as in Real-Time Temporal Logic (RTTL) [62] or Temporal Logic of Actions (TLA) [1].

Suitability for QoS specification. The specification of many QoS characteristics requires reference to events and states that belong to different transitions in an automaton model, and sometimes even to different automata if the model is composed of concurrently executing state machines like in SDL or Estelle (see discussion in [26]). Due to their syntactic independence from particular automata constructs logic formulas are much better suited to express QoS requirements than real-time annotations on automata transitions. Both Real-Time Estelle and SDL/MTL therefore encompass real-time temporal logic formulas.

3 QoS Terminology and Typical Requirements

To lay a foundation for the remainder of the paper we first discuss standard QoS terminology as defined by ISO. Next, we introduce aspects of QoS architecture and discuss a number of quantitative QoS requirements that typically appear in broadband and multimedia systems. The examples are presented in natural language here, but we will revisit some of them in Section 8 to illustrate the use of Real-Time Estelle and SDL/MTL in formally specifying them. Finally we motivate the importance of real-time aspects in the specification of QoS properties.

3.1 QoS Terminology and Standards

As it can be expected for a relatively young and immature area like 'QoS-engineering', there is an abundance of terminology with ambiguous interpretations in the literature. In a joint effort, the International Standards Organization (ISO) and the International Telecommunications Union - Telecommunications Standardization Sector (ITU-T) have devised a standards document for Quality of Service in an Open Systems Interconnection context [39]. We will briefly review some of the terminology used in this standard and relate the meaning to our usage of these terms.

Quality of Service. The standard does not provide a clear definition of what Quality of Service denotes. However, it derives a definition from the related Open Systems Interconnection (OSI) standards document:

> *"Quality of Service: A set of qualities related to the provision of an (N)-service, as perceived by an (N)-service user."*

We will assume that the underlying ontology of our discussion will be that of *characteristic aspects* of communications services, where the presence, absence or gradual presence of these aspects defines qualities of a communications service to be provided.

QoS Characteristic. In line with the above definition the standard uses the term *QoS characteristic* as a fundamental term from which many other terms and concepts are derived:

> *"... some aspect of QoS that can be quantified ... It is defined independently of the means by which it is represented or controlled."*

The important aspect here is that the QoS characteristic is independent of particular mechanism for its implementation, a concept helpful in maintaining abstraction in specifications. This term defines the *physical* aspect of Quality of Service, in the terminology of the standard *"the true underlying state of affairs"*.

QoS Requirement. The standard phrases:

> *"QoS information that expresses part or all of a requirement to manage one or more QoS characteristics, ...; when conveyed between entities, a QoS requirement is expressed in terms of QoS parameters."*

In standard software and systems engineering terminology, requirements commonly express design-time desiderata - properties that are required to hold of some artifact, independently of its implementation [20]. We note problems relating the standard terminology to the more common usages of the term *"requirement"*, in particular because a "requirement" in the standard seems to relate to a run-time function of the implementation rather than defining an implementation independent abstract design-time requirement. We prefer to use the more common connotation and say that the "QoS requirement" defines a constraint on one or more of the system's QoS characteristics that may at run-time not be violated without invalidating the system's purpose. In other words, QoS requirements define the *software engineering* aspect of QoS - they are the basis for the documentation of QoS related system constraints, and they are used to perform QoS-related testing and system validation.

QoS Parameter. The standard describes this as

> *"... a vector or scalar value relating to QoS that is conveyed between entities."*

In other words, this covers the *syntactic* aspect of QoS related mechanisms at run-time of the system. As an example, QoS parameters can be found in a protocol data unit when two protocol entities negotiate a QoS value for a particular connection.

QoS Guarantees and Mechanisms. While we understand QoS requirements to represent design-time constraints, we say that at run-time a system is to provide a service *guaranteeing*[1] a certain level of QoS. Open Distributed Systems frequently change their appearance and characteristics: bandwidth

[1] Note that the standard does not know the concept of a *QoS guarantee*, while it is a term that is frequently used in the literature (see for example [46]).

requirements change largely over time, users sign on and sign off, and network components are being added or removed from the system. Therefore, in order to implement QoS guarantees, the service will need to employ certain QoS mechanisms:

> *"Meeting a QoS requirement may require the use of mechanisms for QoS establishment, QoS monitoring, QoS alert, QoS maintenance, QoS control or QoS enquiry ..."*

In a somewhat broader sense we will say that QoS mechanisms are algorithms or functions implementing a QoS guarantee at run-time [2].

QoS Management Function. The boundary between QoS management functions and QoS mechanisms is not really crisp. Both relate to the *algorithmic* aspect of QoS. The standard says:

> *"... is the general term for a function designed to meet a QoS requirement."*

We will use this term to denote particular *protocol* or *service provision* mechanisms that help in implementing QoS requirements at run-time. Examples are QoS negotiation, monitoring and adaptation.

3.2 QoS Architecture

A few years ago QoS was only defined at the boundary between applications and the communication network transport service. It was then realized that the identification of quantitative measures of service quality are also necessary at other interfaces, for instance between user and application or between the communication system and the operating system, and between end users. To deal with end-to-end QoS, several so-called *QoS architectures* have been developed [6]. One of the most important tasks of a QoS architecture is to provide specifications for QoS requirements on different layers of a given system. Since different layers provide different services the QoS requirements need to be defined in terms of the terminology used at the respective layer interface. For example, it makes no sense to offer a parameter like *maximum allowed ATM cell transfer delay* at the user interface since to a user at the application level ATM cell transfer delay is a meaningless QoS characteristic. We now give an overview of typical QoS requirements encountered at different system levels.

[2] As an example the "playout buffer" mechanism in ATM implements a delay jitter bound [50].

User/Application QoS. At the user level, QoS requirements on multimedia data are formulated in a way human users can easily understand. A typical example in a video-on-demand environment would be to offer a movie in different qualities: *high resolution color* or *low-resolution black&white*. The audio belonging to this movie could either be *mono* or *stereo*, it could be *telephone* or *CD* quality. A user does not care if the movie needs a throughput of 2 or 4 MBit/s since this is meaningless to him. Furthermore, the user will not specify that video and audio parts of a movie are lip-synchronized — he expects that implicitly. Therefore, the mapping of user-defined QoS requirements to the respective transport parameters has to be done by the application. It also has to split a movie in its different streams (video and audio) and derive the respective QoS requirements for each of them as well as necessary inter-stream synchronizations. For this purpose, intermediate application-oriented QoS requirements may be used, e.g. the *frame rate* of a video stream (25 frames per second) or the *sampling rate* of an audio stream.

Transport System QoS. On the transport level, well-known parameters are *throughput*, *transfer delay* and *jitter* on an end-to-end basis, i.e. process to process communication. Requirements on these parameters have to be derived from the user requirements. To be able to transfer the huge amount of data a movie usually consists of, a certain throughput is required. By specifying a certain delay, the time between sending a given packet at one side and receiving it at another may be limited. This is especially important for video or voice conferences with direct human-to-human interaction. Jitter, finally, is a measure for the variation in delay. If the jitter becomes too high, which means that data packets arrive on a very irregular basis, the quality of audio transmission may be highly affected.

Medium Access QoS. The QoS characteristics for medium access may differ widely, depending on the kind of medium or network to be accessed. For ATM, there are several QoS parameters defined on the cell level, e.g. *cell loss rate*, *cell transfer delay* or *cell insertion ratio*.

3.3 The Role of Real-time in QoS Specification

The QoS characteristics that we discussed above can be classified as *quantitative QoS characteristics*, as far as they refer to layers below the end user application. Note that they all refer to quantifiable properties as for example in "a rate of 15 frames per second" or a "delay of 200 milliseconds". Note also that all the examples we introduced refer to *real-time*. A desired throughput, as an example, is typically expressed by a a certain number of data packets or bits to be sent *per second*. Inter-stream synchronization requires two data units of two independent streams to be received within a time interval of several *milliseconds*.

In other words, the correctness of these systems does not solely depend on the functionally correct sequence of observable events but also on a correct timing of the event sequences. We conclude that in order to use an FDT for the specification of quantitative QoS requirements this FDT has to possess suitable real-time expressiveness, including a formal semantics accounting for real-time. In the following Section we analyze whether standard Estelle and SDL meet these requirements.

4 Real-time concepts in Estelle and SDL

Both standard Estelle and SDL already have a built-in real-time mechanism. Estelle uses a `delay` clause to express timing constraints whereas SDL uses a *timer* mechanism. We analyze the suitability of these constructs to express hard real-time constraints.

4.1 Real-time in Estelle

Informally, the semantics of a clause `delay(E1,E2)` associated with a transition t can be described as follows (see also [38]):

(i) Once newly enabled [3], t cannot be executed until it remains enabled for at least $E1$ time units.
(ii) If t remains enabled but is not executed for E time units, $E1 \leq E \leq E2$, then even if t is the only enabled transition within a module instance at the moment, t still may or may not be executed.
(iii) If t has been enabled for E time units, $E \geq E2$, then if t is the only enabled transition, t will execute. Otherwise, any other enabled transition may also be executed, possibly disabling t.

It is possible to omit the second parameter: the form `delay(E1)` is equivalent to `delay(E1,E1)`.

[3] An Estelle transition is enabled if all its enabling conditions are true. These conditions comprise the `when` and `provided` clauses. Note that satisfaction of the `delay` clause is not required.

The result of the above is that a transition t with `delay` clause `delay(E1)` will, if at all, execute at a point in time T_x. Let T_0 denote the time instant at which t becomes enabled, then $T_x = T_0 + E1 + \epsilon$. The value of ϵ is unbounded since it largely depends on the execution times of other Estelle modules in the same system module subtree. The Estelle standard explicitly states that no assumptions can be made about the execution speed of transitions. We conclude that T_x is also unbounded and that no real-time guarantee can be given for the point in time at which t executes. This makes it impossible to specify hard real-time constraints using the `delay` clause. As we argue above, however, hard real-time requirements are an essential ingredient in specifying deterministic QoS requirements, and we conclude that standard Estelle is therefore not suitably expressive to specify this sort of QoS requirements.

Suitability of transition-based real-time constraints. The `delay` clause does only refer to single transitions. Many quantitative QoS requirements, however, refer to more than one state transition of a process, or even to more than one process. Consider a process doing MPEG decoding. This process can be further structured into a dispatcher module and some decoding modules running in parallel. Every time a group of frames arrives at the process, the dispatcher assigns it to one of the decoders where the frames are decoded and passed back to the dispatcher. The latter then passes them on to the presentation device. A typical QoS requirement on such a process would be that no more than a certain delay may be added to the overall delay by the execution of this process. Inside the process, this maximum additional delay has to be further distributed among the dispatcher and decoder modules, but we are not interested in specifying which module gets which share of the allowed delay. This illustrates the need for a more global real-time construct that is independent of the reference to individual transitions.

4.2 Real-time in SDL

In SDL, real time is introduced by an *asynchronous timer mechanism.* Figure 1 specifies an SDL "design pattern" found in many specifications: the requirement is that if for a request `Q` (that is sent either to another process or the environment) a response `A` is not received within `t` time units, a signal `alarm` will be sent. An SDL specification can access the value of a global clock by reference to a variable called `NOW` which always refers to the current moment in time. The SDL command `set(NOW+t,T)` sets the value of a timer called `T` to a time value t time units greater than the current moment of time. A process which sets a timer is called a *timed process.* The set timer is controlled by an independent *timer process.* Each time a timed process in the specification sets a timer, an instance of the timer process is generated (compare

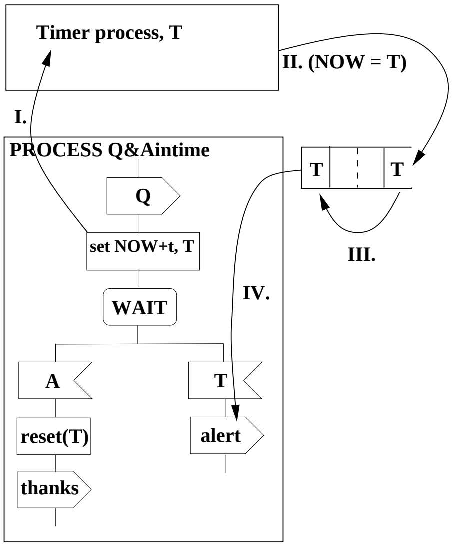

Fig. 1. Partial SDL specification with timer

I. in Figure 1). The timer process continuously compares the value to which the timer is set with the current global time. When the value that has been set is reached or exceeded (*II.*), the timer process communicates the expiry to the timed process by placing a *timer signal* at the end of the input queue of the timed process. Like any other signal, the timed process may consume the timer signal from its input queue whenever it has reached the head of the queue (*III.*), and react accordingly (*IV.*). Timers may be reset by the timed process in which case the timer process deactivates the respective timer. The reset also removes the timer signal from the timed processes' input queue in case the timer expired before the reset but hasn't yet been consumed by the time the reset occurs.

Similar to Estelle, the delay δ between the point in time when the timer expires and the moment at which the SDL specification reacts to the expiry is unbounded. The value of δ can be estimated as $0 \leq \delta \leq T_1 + T_2 + T_3$. T_1 is the time between the generation of the timer signal and its placement in the timed processes' input queue. T_2 is the time it takes for the time signal to reach the head of the input queue. Finally, T_3 is the time it takes for the timed process to consume the signal once it has reached the head of its input queue. All of these delays are unbounded and consequently δ is unbounded. As is the case with Estelle, no hard real-time properties and therefore no deterministic QoS requirements can be specified using the SDL timer mechanism.

4.3 Conclusion

Our analysis of the built-in real-time mechanisms in Estelle and SDL reveals a major difference between these techniques and the model theoretic techniques we alluded to earlier in the paper. This difference lies in the fact that systems satisfy the Estelle and SDL specifications even if they exceed the time limits specified by the `delay` clause of the timer mechanism by an unspecified and even potentially unbounded amount of time. In other words, they do not allow to tell the "good" execution sequences from the "bad" ones, as far as hard real-time constraints are concerned. The most they can express is that there is a *minimum* amount of time that passes between the setting of the timer and the recognition of its expiry by the timed process. In subsequent Sections we will show how Estelle and SDL can be extended to model theoretic techniques. In the following Section we define a general timed execution model that will later serve for interpreting Real-Time Estelle and SDL/MTL specifications.

5 A Real-Time Execution Model

We discuss the design of a timed execution model that will allow us to interpret the executions of systems specified in Real-Time Estelle and SDL/MTL. There are three major questions that need to be answered when defining a timed execution model: (1) is it adequate to assume the existence of *global* time, (2) how does one augment untimed state sequence execution models to account for time, and (3) what impact does the inherent concurrency of the specification model have on the choice of the time representation?

Global time model. We first address the question of finding a suitable time model that adequately reflects the distributed nature of the systems we are interested in. Essentially, for any 'reasonably' behaving system the *global time* assumption is valid [4], which is why we conjecture that we can always use a global time model in our specifications.

Timed observation sequences. We model system executions by infinite discrete state sequences $s = s_0, s_1, \ldots$ [5]. We restrict ourselves to the externally observable state component of any system and assume the existence of global system states. For an arbitrary state sequence s described by an Estelle or SDL specification the semantics of Estelle or SDL determines which transitions from an s_i to an s_{i+1} are legal. Alternatively, we may obtain s by observation of

[4] Lamport's axiomatization of temporal relationships between events in [48] makes minimal assumptions about the system. In particular, it assumes no atomicity of events and it is space-time relativistic. [8] and [5] show independently that for any system satisfying Lamport's axioms, and we call any such system 'reasonable', there is a global time model.

[5] Our presentation here follows [4].

the sequences of global states of an executing system at run-time. Following the global time assumption we associate a global time interval $I_i \in R$ with any system state s_i and assume that state changes only occur at the left and right interval boundaries l_i and r_i, respectively. For a sequence of intervals $I_0, I_1 \ldots$ we assume that any two neighboring intervals are adjacent and that for any $t \in R$ there is an I_i so that $t \in I_i$. As we assume that state changes only occur at the interval boundaries and that events triggering state changes are instantaneous, we may use either the left or the right interval boundary to describe the sequence of global time intervals associated with the sequence of states. For instance, we represent the interval sequence

$$[1,3), [3,3.1), [3.1,5), \ldots$$

by the sequence

$$1, 3, 3.1, 5, \ldots$$

of left interval boundaries. In any finite interval I_i there can be only finitely many observable state changes or events (the *finite variability* assumption). Furthermore, we assume that all events in the system coincide with the clicks of a global clock. As a consequence, there can only be countably many state changes in any execution sequence and it suffices to use the nonnegative integers as time domain [6]. For example, we may represent the above sequence of rational left interval bounds by the integer sequence

$$10, 30, 31, 50, \ldots$$

We define a timed observation sequence $o = o_1, \ldots$ as a tuple (s, l) where $s = s_0, s_1, \ldots$ is a discrete state sequence and $l = l_0, l_1, \ldots$ is the sequence of corresponding left interval bounds. An example of a timed observation sequence is

$$(s_0, 10), (s_1, 30), (s_2, 31), (s_3, 50), \ldots$$

Interleaving semantics and real-time. Both SDL and Estelle describe concurrent systems, and it is most useful and common to give them an interleaving trace semantics. Assume that s_1, s_2 and s_3 are global system states of an SDL or an Estelle specification, and that s_2 and s_3 are concurrent states. Following an interleaving semantics, both $s_1, s_2, s_3, \ldots$ and $s_1, s_3, s_2, \ldots$ are admissible sequences in the untimed model. In order to express that both s_2 and

[6] However, discrete time domains can hinder refinement steps. Hence, if refinement is crucial dense time domains like the positive reals are necessary (c.f. [1] and [57]).

s_3 may occur concurrently in the timed model (which means that they have the same time stamp) we have to allow that both timed observation sequence

$$(s_1, l_1) \to (s_2, l_2) \to (s_3, l_3) \to \ldots$$

and

$$(s_1, l_1) \to (s_3, l_2) \to (s_2, l_3) \to \ldots$$

are admissible and that $l_2 = l_3$. Therefore, we generally assume the sequence l_i of a left interval bound time stamps to be a *weakly*-monotonic integer sequence.

6 Complementary SDL and Metric Temporal Logic Specifications

In this Section we define a rudimentary computational model for SDL specifications, a so-called *Global State Transition System* (GSTS), which will serve as a common formal model for the interpretation of SDL specifications and temporal logic formulas. We define the global state in the GSTS model to be determined by the local state of the processes plus the state of the communications in between processes. The main components of the GSTS model are:

- *Process control and data manipulation.* This component represents the local behavior of an SDL process which executes transitions between *symbolic* states.
- *Communication.* SDL processes communicate asynchronously via potentially unbounded queues. Each SDL process has exactly one input queue handling all incoming communication from any other process [7]. The local state of an SDL process hence consists of the combination of current values for the data variables, the point of local process control, and the state of the input queue.
- *Global System States and State Transitions.* The global system state (GSS) is the product of all local states of all processes of an SDL specification. SDL processes run concurrently. In accordance with the standard documents [40,41] we choose an interleaving approach to represent this concurrency. We assume a nondeterministic choice when more than one process has an enabled transition in a given GSS.

[7] For reasons of conciseness we do not address inter-process communication mechanisms like *viewing* or *remote procedure call*, but a treatment of these communication mechanisms within our framework is straightforward. Furthermore, we only consider so-called *non-delay* channels in the SDL specifications.

Note that the resulting GSTS model for SDL specifications is not finite. For a given SDL specification, the unwinding of the corresponding GSTS model will describe all admissible sequences of states of an SDL specification, called its computations. In describing sequences of states, the model also describes sequences of state transitions, which are in turn triggered by events (e.g. input and output) in the system. The computations will later serve as models for what we call *complementary* temporal logic specifications, only those specifications which satisfy both the properties expressed by the SDL specification *and* the properties expressed by the temporal logic specifications are considered to satisfy the complementary specification. It should be emphasized that the goal here is not to define *yet another* formal semantics for SDL in addition to the ones already defined (e.g. [40] or [14]). The motivation for defining our own SDL semantics is two-fold: First, we found none of the published semantics suitable for our purposes, which are to interpret SDL specifications and Temporal Logic formulae on common model-theoretic grounds. Second, our semantics is exemplary in nature and intended to cover only a small subset of the SDL language.

Overview. In Section 6.1 we define the notion of a *Process State Transition System* (pSTS). A pSTS has components similar to an extended finite state machine, plus a process-unique input queue. We also define a transition relation and the notion of an admissible state sequence for pSTS here. The interpretation of pSTS as SDL processes is presented in Section 6.2. In Section 6.3 we demonstrate how to augment `INPUT` and `OUTPUT` statements to state-propositions. In Section 6.4 we define global state transition systems (GSTS) which correspond to SDL specifications. They consist of concurrently operating pSTS. We show in this Section how to formally handle `OUTPUT` statements and we define global system state sequences which yields the computational model over which we will later interpret temporal logic formulas.

Related work. Our definitions here are close to the *Basic Transition Systems* of [58]. Our pSTS models can be seen as a logic-based formulation of *Extended Finite State Machines* (EFSM) [55]. The modeling of SDL processes as EFSMs has been suggested in [7] and [66]. However, as we will see later, the mapping of SDL process transitions as informally described in these approaches is too coarse in order to adequately represent the structure of an SDL transition. [56] contains a formalization of SDL based on FSM, hence without treating data variables over infinite domains. Formalizations of EFSMs can be found in [36] (where the state space is finite by limitation of the range of data variables and variables representing the state of communication channels to finite domains), and in [17] and [45] (from where we take part of our formalization). [12] describes and formalizes the use of queues to model the collective behavior of concurrent FSM which communicate asynchronously via queues (there called *protocols*) and we use part of their formalization for our work. Similar to our approach [13] presents a temporal logic-based semantics for Estelle using Dijkstra's predicate transformers. Our interest here is not primarily in verification, therefore we use the more intuitive Hoare-style triples consisting of a pre-condition, a code fragment, and a post-condition. [60] agrees with our analysis of the shortcomings of the SDL real-time mechanism, and proposes reconciling SDL with the *Duration Calculus*.

6.1 Process State Transition Systems

The pSTS that we introduce in this Section define an SDL process as a set of symbolic states, a set of program variables (consisting of control and data variables), and a set of communication events (input and output of signals). The 'logic' of an SDL process is encoded in its state transition relation.

Formal Definition Process State Transition System (pSTS). A Process State Transition System P is defined as a tuple (S, D, V, O, I, Q, T, C) where

S is a finite set of *symbolic states*,
D is an n-dimensional linear space where each D_n is an *interpretation domain*,
V is a finite set of *program variables*, $V = \{\pi, v_1, \ldots, v_n\}$ where π is a *control* variable ranging over elements of S and $v_1, \ldots, v_n$ are *data* variables so that $v = (v_1, \ldots, v_n) \in D$,
O is a finite set of *output signal types*,
I is a finite set of *input signal types*,
Q is a linear sequence $q_1, \ldots, q_m$ (in the standard mathematical sense) of elements from $I \times D$ which we call *input queue*,
T is a *transition relation*, with $T : S \times 2^D \times Q \rightarrow S \times 2^D \times Q$, and
C is an *initial condition* on $S \times 2^D \times Q$.

A *state* s, is a function $s : V \times Q \rightarrow 2^S \times 2^D$ assigning a value to every variable in V and to Q. s can evidently have a potentially infinite range. By $s[x]$ we denote the value of variable x in state s.

Transition Relation, Admissible Sequences, and Reachable States. We associate a set $\mathcal{T}_T = \{\tau_1, \ldots, \tau_m\}$ of *transitions* with the transition relation T of an pSTS. With each transition τ_j we associate a pair of state propositions P_j and Q_j and we call P_j a *precondition* and Q_j a *postcondition* of transition τ_j. We assume the existence of a satisfaction relation $\models_P$ which relates assertions about the system state to system states for a given pSTS P[8]. In particular, we write $s \models p$ iff state s satisfies state-proposition p[9]. Now, in order to relate states s and s' we say that $(s, s') \in T$ iff

$$(\exists \tau_j \in \mathcal{T}_T)(s \models P_j \wedge s' \models Q_j).$$

Let $\sigma = s_0, \ldots, s_k$ denote a finite sequence of states. We call this sequence *admissible* iff

$$(\forall 0 \leq j < k)((s_j, s_{j+1}) \in T).$$

This definition extends to infinite sequences in the obvious way. A state s_k is a *reachable* state iff the sequence $\sigma = s_0, \ldots, s_k$ is admissible and $s_0 \models C$, i.e. s_0 is the initial state. In state formulas, when referring to states s and s' with $(s, s') \in T$ we sometimes denote $s[v]$ by v and $s'[v]$ by v'. In order to express that a transitions τ_k is *enabled* in a state s we write $s \models en(\tau_k)$ iff $s \models P_k$. For a pair of states (s, s') we say the transition τ_l has been *taken* iff $s \models en(\tau_l)$ and $s' \models Q_l$. We denote this by $ta(s, s', \tau_l)$. Let the variables X and Y range over the queues of a pSTS, i.e. over sequences of signal types, and A over signal types. The concatenation of a sequence and a singleton element is expressed by juxtaposition. For a signal queue X and a signal type A the term XA describes a sequence where A is the *last* element. Conversely, AY describes a sequence where A is the *first* element.

6.2 Interpreting SDL-Processes as pSTS

We now explain the mapping of an SDL process to the components of a pSTS. In SDL terminology, a *transition* describes the change of processes control from one symbolic state to a symbolic successor state. In the example in Table 1 the two symbolic states are `S1` and `S2`, hence for the corresponding pSTS $S = \{S1, S2\}$. The body of a transition consists of different sorts of statements, like assignments, decisions, communication statements, etc. In order to describe the state of the system before and after the execution of a transition we assign pre- and postconditions to every transition. In a few cases, when the transition body has a trivial structure, the determination of pre- and post-conditions is straightforward. However, as we shall see later, we also need to treat more complex transition structures differently.

[8] We omit the reference to P when this is clear from the context.

[9] We will not define all details of the relation $\models$ formally and refer the reader to [58].

```
STATE S1;
INPUT(A);
TASK x := y + 1;
NEXTSTATE S2;
```

Table 1
SDL Transition I

τ_j	P_j	Q_j
τ_1	$\pi = S1 \wedge Q = AX$	$\pi' = S2 \wedge Q' = X \wedge x' = y + 1$
τ_2	$\pi = S1 \wedge Q = CX \wedge C \neq A$	$\pi' = S1 \wedge Q' = X$

Table 2
pSTS predicates for Transition I

Formal Treatment of `INPUT` Statements, Control Flow, and Variable Assignments. For the time being we only consider single SDL processes and we do not yet interpret effects of communication. `INPUT` statements have a semantics purely local to one process, i.e. to remove the signal at the head of the input queue and assign its value to a local variable. Table 2 shows the mapping of an SDL transition to transitions τ_j of a corresponding pSTS. More precisely, when executing a transition associated with an `INPUT(X)` statement, the process first checks whether the signal at the head of its input queue is of type `X` [10] . If this is true the process consumes the signal by removing it from the head of the queue and assigning its value to a local variable with the name `X`. However, if the signal at the head of the queue does *not* have the expected type, then the message is removed from the head of the queue, discarded, and the same `INPUT` statement is re-enabled. We therefore need to split the treatment of `INPUT` statements into two logical cases, the first being the one where the expected signal type is not at the head of the queue, and the second where the expected signal is at the head. Hence, we treat transitions with `INPUT` statements as two transitions which are mutually exclusive (see transitions τ_1 and τ_2 in Table 2). The logical exclusion is encoded by the test $Q = AX$ which is *true* in case the head of the input queue contains the message of expected type `A`, and the test $Q = CX \wedge C \neq A$ which evaluates to *true* iff this is not the case. Attention has also to be paid to the *control flow* in a transition. If we consider a transition which brings a process from symbolic state `S1` into symbolic state `S2`, then this can be interpreted as though control lies in code location `S1` before execution of the transition, and in location `S2` afterwards. We defined a distinguished variable π to range over code locations, called symbolic states, and we use this variable to formulate pre- and postconditions characterizing the control flow inside an SDL process (see the use of variable π in Table 2). *Variable assignments* are treated in a very standard way, as for

[10] For reasons of conciseness we do not treat the handling of `SAVE` statements here, for their modeling in the context of an FSM interpretation we refer the reader to [56].

```
STATE S1;
INPUT(A);
DECISION D(A);
 (true):
   NEXTSTATE S2;
 (false):
   NEXTSTATE S3;
ENDDECISION;
```

Table 3
SDL Transition II

τ_j	P_j	Q_j
τ_1	$\pi = S1 \wedge Q = AX \wedge D(A)$	$\pi' = S2 \wedge Q' = X$
τ_2	$\pi = S1 \wedge Q = AX \wedge \neg D(A)$	$\pi' = S3 \wedge Q' = X$
τ_3	$\pi = S1 \wedge Q = CX \wedge C \neq A$	$\pi' = S1 \wedge Q' = X$

Table 4
pSTS predicates for Transition II

example, described in [58]. Let x and y denote variables in a state s, let x' and y' denote these variables in the successor state s', and let the system transit from s to s' through the execution of a statement `y:= x + 1`. We describe this transition by the postcondition $y' = x + 1$ which is required to hold in state s' (see Table 2 for the postcondition describing the update of variable x in the example of Table 1).

Formal Treatment of `DECISION` Statements. We decompose a `DECISION P(x)` statement into two mutually exclusive transition alternatives. The first is that the decision predicate holds, i.e. $P(x)$ is *true*, the second is that $P(x)$ is *not* true. As an example see the treatment of the decision in Table 3 in Table 4.

Handling Iterative Transitions. So far we assumed that the symbolic states in the set S are identical to the symbolic states used in the SDL specification. However, SDL transitions may have iterative structure, achieved by a goto and labeling mechanism (the goto statement is called `JOIN` in SDL, see Table 6). Therefore we need to abandon the idea that a transition in an SDL process leads from one symbolic state to a symbolic successor state, as for example suggested in [7]. We need to allow cyclic control flow structures and suggest introducing *auxiliary* symbolic states which correspond to the target locations in the control flow to which a process jumps back or forth when executing `JOIN statements`. In the example in Table 6 we introduced an additional symbolic state `S1-1`, corresponding to the point of control which is reached when jumping to label `l1` (we introduced a comment `/* S1-1 */` in the SDL code at the location corresponding to auxiliary state `S1-1`). The transitions τ_4 and τ_5 represent cases in which control lies in the auxiliary symbolic state `S1-1`.

```
STATE S1;
INPUT(A);
/* S1-1 */
l1:
DECISION D(A);
 (true):
   NEXTSTATE S2;
 (false):
   OUTPUT(B);
   TASK A:=A-1;
   JOIN l1;
ENDDECISION;
```

Table 5
SDL Transition III.

Atomicity of transitions. The SDL standard semantics [41] is not explicit about the question which constructs in SDL are to be executed atomically. It can only be inferred from [41] that atomicity is at the primitive statement level (according to the standard the interpretation of complex statements is based on a decomposition into primitive statements). The above resolution of the join statement in Table 6 assumes a "maximum progress" semantics in which the atomic parts are assumed to be as large as possible. This reduces the state space when using validation tools like *Spin* [36].

pSTS and Extended Finite State Machines. The derivation of an EFSM from a pSTS is straightforward. For the example in Tables 5 and 6 the resulting EFSM would have 3 states ($S1$, $S1-1$ and $S2$), and 5 transitions.

τ_j	P_j	Q_j
τ_1	$\pi = S1 \wedge Q = CX \wedge C \neq A$	$\pi' = S1 \wedge Q' = X$
τ_2	$\pi = S1 \wedge Q = AX \wedge D(A)$	$\pi' = S2 \wedge Q' = X$
τ_3	$\pi = S1 \wedge Q = AX \wedge \neg D(A)$	$\pi' = S1-1 \wedge Q' = X \wedge A' = A-1$
τ_4	$\pi = S1-1 \wedge D(A)$	$\pi' = S2$
τ_5	$\pi = S1-1 \wedge \neg D(A)$	$\pi' = S1-1 \wedge A' = A-1$

Table 6
pSTS for Transition III

6.3 State Propositions INPUT and OUTPUT

The state predicates we defined so far allow us to specify formulas referring to the current point of control (e.g. $\pi = S1$) or on the state of data variables (e.g., $Q = AX \wedge A = \texttt{DR}$ where `DR` stands for a message type). However, sometimes one would much rather specify properties of communication events to happen, i.e. input or output of signals that are about to take place or that have just been executed. We therefore introduce state predicates which indicate which transition has been taken as a last step in a computation, and whether this transition entailed any communication events. Technically, we introduce two relations, *inlabel* and *outlabel*, which label the transitions of the pSTS with the `INPUT` or `OUTPUT` statements that are executed during the course of a transition. We omit the straightforward technical construction of this labeling here. In the example in Tables 5 and 6, we see that for example $\mathit{inlabel}(\tau_3) = \{\texttt{INPUT(A)}\}$ and $\mathit{outlabel}(\tau_3) = \{\texttt{OUTPUT(B)}\}$. Let $s = s_1, s_2, \ldots$ be an admissible state sequence for a given pSTS, and let $\mathcal{T}_T$ denote the set of transitions for this pSTS. We say that $s_i \models \mathit{INPUT}(A)$ iff $(\exists \tau \in \mathcal{T}_T)(\mathit{ta}(s_{i-1}, s_i, \tau) \wedge (\texttt{INPUT(A)} \in \mathit{inlabel}(\tau)))$, and $s_i \models \mathit{OUTPUT}(A)$ iff $(\exists \tau \in \mathcal{T}_T)(\mathit{ta}(s_{i-1}, s_i, \tau) \wedge (\texttt{OUTPUT(A)} \in \mathit{outlabel}(\tau)))$. This construction augments these labels to state propositions.

6.4 Global State Transition Systems

SDL Specifications Formally. SDL specifications consist of collections of concurrent SDL processes. We say that the Global State Transition System (GSTS) G_P corresponding to an SDL specification P is a tuple $G_P = (P^0, \ldots, P^n)$ where each P^i for $i = 1, \ldots, n$ is a pSTS. P^0 (which represents the environment behavior) is not a full pSTS, it only consists of an input and an output alphabet, and an input queue. P^0 has no state and we rely on the facilitating assumption that P^0 will provide any of the other processes with input signals whenever these wish to consume any such signal, and that P^0 consumes instantly any signal which it receives from any process of the SDL system.

SDL processes communicate asynchronously via one unique infinite input queue per process. We interpret the sending of a signal A from a process P^1 to a process P^2, indicated by an `OUTPUT(A)` statement, such that a signal of type A is appended to P^2's input queue Q^2. We slightly simplify the SDL mechanism of mapping of an output signal to a receiving process by assuming that a signal `A` is sent from a process P^i to a process P^j iff $A \in I^j$ [11].

[11] In SDL this involves a mapping of signal names via signal lists to signal routes or channels that point to the receiving process. Note that we only model *non*-delaying

```
PROCESS P1;      PROCESS P2;
  STATE S1;        STATE S3;
  INPUT(A);        INPUT(B);
  OUTPUT(B)        NEXTSTATE S3;
  NEXTSTATE S2;
```

Table 7
SDL specification

τ_j^1	P_j^1	Q_j^1
τ_1^1	$\pi^1 = S1 \wedge Q^1 = AX \wedge Q^2 = Y$	$\pi'^1 = S2 \wedge Q'^1 = X \wedge Q'^2 = YB$
τ_2^1	$\pi^1 = S1 \wedge Q^1 = CX \wedge C \neq A$	$\pi'^1 = S1 \wedge Q'^1 = X$

Table 8
Predicates describing SDL specification

Furthermore, we require $(\forall i = 1, \ldots, n)(\forall a \in O^i)(\exists j \neq i)(a \in I^j)$ and $(\forall i = 1, \ldots, n)(O^i \cap I^i = \emptyset)$. As we saw in Section 6.2, the execution of an `INPUT(A)` statement (which in the SDL terminology is often just referred to as *signal-consumption*) represents an event purely local to some SDL process.

Transition Predicates for OUTPUT statements. The execution of an `OUTPUT` statement involves a non-local action. It means that the execution of the statement is a local event of the sending process, whereas the reception (which in SDL is different from the consumption of the message and just means that the message will be appended to the tail of the receiving process' input queue) is a local event of another (the receiving) process. Therefore, one can not formalize these transitions by state propositions that solely refer to state variable of only one process. Table 8 presents a simple example of a two-process SDL specification $P = (P^0, P^1, P^2)$. Transition τ_1^1 describes both the state change in P^1 and the appending of the signal B to the input queue of P^2. Although strictly speaking this transition also changes the state of process P^2 for our formal treatment we consider transition τ_1^1 to be a transition belonging to process P^1.

Global System States, Transitions, Global State Sequences, and the Satisfaction Relation. Let $G_P = (P^0, \ldots, P^n)$ denote the GSTS for an SDL specification P. We say that the vector $s = (s^1, \ldots, s^n)$ is a *global system state* (GSS) of the SDL specification P iff s^i is a state of pSTS P^i for all $i = 1, \ldots, n$. In the course of each change of the GSS exactly one pSTS changes its local system state. This implements the interleaving semantics that we use to model concurrency in SDL specifications. In a given GSS s, a demon selects nondeterministically one out of all enabled transitions in all pSTS to

channels here, the modeling of delaying channels is a straightforward extension.

be executed next. Executing a transition defines the successor GSS s'. Let $\sigma = s_0, \ldots, s_k$ denote a finite sequence of GSS. We call this sequence *admissible* iff $(\forall 0 \leq j < k)(\exists \tau_l^i)((s_j^i, s_{j+1}^i) \in T^i))$. This definition extends to infinite sequences in the obvious way. Also, the interpretations of the state propositions *en*, *ta*, *INPUT* and *OUTPUT* extend in the obvious way from pSTS states to GSS. Based on the above definitions we may now define a satisfaction relation $\models_{\mathrm{SDL}}$ for SDL specifications. Let P be an SDL specification and let Σ_P^ω the set of all infinite sequences of GSS of P. For a $\sigma \in \Sigma_P^\omega$ we write $\sigma \models_{\mathrm{SDL}} P$ iff σ is an admissible sequence with respect to P.

6.5 Using Temporal Logic for SDL Specifications

The characterization of properties by the use of temporal logic is accomplished by interpreting the temporal logic specification such that the models satisfying all formulas determine the set of admissible state sequences of the system. Now, as we have seen in Section 5, SDL specifications also specify admissible sequences of states. Temporal logic formulas can be thought of as filters on the admissible sequences specified by the SDL specification and therefore can be used to specify those real-time and liveness properties inexpressible in SDL. A crucial point is the selection of a suitable temporal logic language. We will use a temporal logic similar to the logic described in [58], called Propositional Temporal Logic (PTL) and a real-time extension based on PTL, called Metric Temporal Logic (MTL) [34] [43]. However, other temporal logics like TLA [49] or CTL [24] may be linked to SDL specifications in very much the same way as we present it here for MTL.

A State Proposition Language. We assume that the state propositions we use in complementary temporal logic formulas all refer to observable components of the system state, and we use, in particular, the following state propositions for an SDL specification P:

(i) *Actual State:* let $S = S_1^i, \ldots, S_n^i$ denote the symbolic states for a given process P^i of P, then $at\_S_k^i$ denotes the state proposition that the i-th component of the global system state is in symbolic state S_k^i, i.e. $\pi^i = S_k^i$.
(ii) *Input and output:* we use the state propositions *INPUT* and *OUTPUT* as defined above to denote that we are in a state where an input or an output of a signal has just occurred in the last GSS transition.
(iii) *Data:* we allow the reference to visible data variables and allow standard comparison operators on the variables. We allow state formulas to be constructed by using boolean operators between state propositions and we call composed state formulas *state predicates.*

Example. The state formula $n \leq 3 \wedge INPUT(A)$ holds in all GSS in which the value of variable n is less than or equal to 3 and an input of a signal of type A has just been executed. The state formula $at\_S1 \supset n \geq 3$ holds in all those GSS in which if the control is in symbolic state $S1$ then the value of variable n is greater than or equal to 3.

Temporal Logic. The Propositional Temporal Logic (PTL) we use here is a linear time temporal logic taken from [58]. We'll use the future operators $\Diamond$ ("eventually"), $\Box$ ("henceforth") and $\mathcal{U}$ ("until"). In addition we define a *strong eventuality* operator $\diamonddot$ so that $\diamonddot p$ holds in some *future* state s that is not the current state, formally $s_i \models \diamonddot p$ iff $(\exists j > i)(s_j \models p)$. The formal semantics of PTL defines a satisfaction relation $\models_{\mathrm{PTL}}$. An execution sequence $\sigma = s_0, \ldots$ of states s_i satisfies a formula ϕ iff ϕ holds in s_0, and we write $\sigma \models_{\mathrm{PTL}} \phi$. We say that a system satisfies a formula ϕ iff all its execution sequences satisfy ϕ.

Metric Temporal Logic. We use an extension of PTL for the specification of real-time requirements, called *Metric Temporal Logic* (MTL). For a complete formal definition of the syntax and semantics of MTL we refer the reader to [4] and [34, Section 3.4]. The models over which we interpret MTL formulas are timed observation sequences $o = o_1, \ldots$ as defined in Section 5. Informally, MTL contains formulas of the form $\Diamond_I \phi$ which assert that *one* of the following states within the time-interval described by expression I is a state which satisfies ϕ. Formulas of the form $\Box_I \phi$ assert that *all* states in the time-interval described by I satisfy ϕ. The expression I describes an either open or closed interval over the time domain and we sometimes use semi-algebraic expressions to refer to these intervals. We write $o \models_{\mathrm{MTL}} p$ iff the sequence o satisfies the MTL formula p.

6.6 Complementary Specifications

Assume we have an SDL specification P and a set of formulas M in MTL. Now, P and M are *complementary* specifications if we require from the specified system that for all its timed observation sequences $o = (s_0, t_0), \ldots$ the following condition holds:

$$s \models_{\mathrm{SDL}} P \wedge o \models_{\mathrm{MTL}} M.$$

Scoping. It is beyond the scope of this paper to discuss the scoping of names used in SDL/MTL propositions in detail. In the later examples we will only make use of signal names as basic propositions. According to the SDL definition, signal names have to be unique in the name scope of an SDL system. Hence there is no problem with name ambiguities. However, when process internal variables are to be used it is easy to disambiguate these by prefixing their names in formulas with the name of the context (e.g., the process or block name) which defines their scope.

7 Real-Time Estelle

7.1 Introduction

The standardized FDT Estelle [38], like SDL, is an automata-based language. An Estelle specification describes a system as a set of hierarchically ordered finite state machines called *modules*. Modules communicate with each other via asynchronous FIFO queues. The communication ports of a module are called *interaction points* (IPs). A module's behavior is described by states and transitions between them. A transition is composed of communication actions, variable assignments, procedure and function calls. Modules at level n in the hierarchy may be dynamically created or destroyed by their *parent module* (module at level $n-1$) during system runtime. According to synchronization rules specified in the Estelle standard modules can be allowed to execute concurrently, or they can exclude each other from simultaneous execution. The execution model of Estelle uses an interleaving approach to model the concurrency in an Estelle specification. An Estelle transition is the smallest observable execution unit. It is atomic and executed as a whole or not at all. Intermediate results during its execution are not visible.

The execution model of Estelle is very similar to the state transition execution model introduced in Section 5. This makes it an easy task to enhance this model in order to describe timed state sequences, see Subsection 7.2. Notions similar to the *Global State Transition System* and the *Process State Transition System* defined in Section 6 already exist in Estelle and need not be defined.

Unlike SDL/MTL, where functional and temporal specifications are expressed by completely different syntactic constructs, we developed Real-Time Estelle as a syntactic extension of standard Estelle, The syntax of standard Estelle is a superset of the programming language Pascal. In the design of the real-time extension, the main considerations were that (i) the real-time restrictions should be included in the functional specification and (ii) the additional language constructs should be in line with the spirit of the existing language, keeping Estelle as simple as it is from a syntactic point of view. This includes the idea that every specification should be writable in pure ASCII to make it immediately processable by a machine. The syntax of Real-Time Estelle is described in Subsection 7.3.

Due to the high degree of flexibility that complementary semantics offer, we adopt this approach for Real-Time Estelle. The details can be found in Subsection 7.4. Finally, we present a first introductory example showing how Estelle and Real-Time Estelle parts are combined to express real-time requirements based on a given functional specification.

Related work. It has previously been suggested to extend Estelle by further real-time constructs. In [22] the authors add upper and lower bounds (there called "execution time parameters") on the execution time to transitions. The idea is to constrain the real-time behaviour of a system by using known execution times of certain system components as execution time parameters. In [15] the same effect is obtained by adding a new clause — `doby(x)` — to transitions, implying a hard upper time bound for the transition execution relative to the time of enabling. The approach described in [71] is more performance-oriented, introducing constructs such as resources and probabilities into the language. All these approaches relate real-time or performance requirements to single transitions. While this technique is suitable for timed simulations, the same critique as for the basic timed automata models described in Section 2 applies to it with respect to QoS requirement specification. A new timed variant of Estelle has recently been proposed in [70] where timing relationships can be specified between modules and transitions. However, the approach is rather incomplete, lacks a formal definition, and mistakes in the specification examples in [70] make it difficult to evaluate its usefulness.

7.2 Real-time Model

During the execution of an Estelle specification, each module is characterized by a current state which is composed of the current symbolic state, the current values of local variables, and the current contents of message queues. A *global situation* (overall state of the specification, equivalent to the global system state in SDL) is composed of the states of all modules plus additional information about module hierarchies, communication relationships and transitions still to be executed. Transitions of a specification are executed in two-phase cycles. During the *system management phase*, a number of transitions (at maximum one per module) is selected for execution according to certain rules. The transition execution itself takes place during the second phase. As soon as all selected transitions are executed, the system enters a new management phase. The global situation of a specification is changed by executing either a module's transition or a system management phase. In reference to Section

6, a module can be compared to a Process State Transition System, while the overall specification transition system is similar to the Global State Transition System.

Estelle offers several ways to specify indeterministic behavior, leading to numerous ways to select transitions for execution in each global situation. Each sequence of global situations produced by the global transition system is called a *computation* in Estelle, and the system is fully specified by all computations produced by the transition system. Compared to the real-time model described in Section 5, (*timed* state sequences), Estelle's computations do not have a time component. To upgrade this model to timed state sequences we need to add the time aspect to computations. We do this by adding one component to the global state description. This *time component* is a positive integer variable, and the only restriction on its value is the following: given two subsequent states s_i and s_{i+1} of one computation, with time components l_i and l_{i+1}, then $l_i \leq l_{i+1}$, i.e., if time changes then it increases. This notion of time is exactly compatible with that introduced in the standard. Conforming to the real-time model introduced in Section 5, a pair (s_i, l_i) denotes the point in time l_i from where on the system is in state s_i.

In addition to the time aspect, we also extend the possibilities to characterize states. The Estelle standard 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 either been sent (output-statement) over or received (when-clause) at interaction point `p` during the last transition, respectively. These predicates are comparable to those described in Section 6. As we will see in Section 8, they are very useful to describe temporal relationships between communication events, a major means to specify QoS requirements.

Furthermore, we introduce the *instance operator* `[]` to allow counting of messages sent or received at a certain IP since the instantiation of the respective module. The predicate `SENDING OF (p.m[z])` is true when the zth instance of message `m` has just been sent over interaction point `p`. As we will see, this operator is useful for specifying properties of consecutive communication events of the same type, and requiring numbers of occurrences of such events in a given time interval.

7.3 Syntax of Real-Time Estelle

All real-time constraints referring to a module are collected in a new section of the module's behavior description, i.e., inside the module's `body`. The section is marked by the keywords TIME CONSTRAINTS. Assigning the constraints to the body description is advantageous compared to other solutions. Assigning them to the module interface description would make it impossible to refer to states of the module. Assigning them to single transitions as in [22] proves to be too inflexible with respect to QoS requirement specification, as we argued above. It should be noted that this approach does not limit the expressiveness of Real-Time Estelle with respect to more global requirements concerning more than one module. Such requirements can be expressed in a higher-level module comprising the modules in question. A typical example would be a service module which includes two protocol modules. A global delay requirement can then be expressed by referring to the service module's communication interface rather than to those of the protocol modules.

The following is an example of a typical module body in Real-Time Estelle:

```
body m-behavior for m-interface;
  var v1 : V1Type;

  state s1, s2, sn;

  time constraints
    constraint1;
    constraint2;

  initialize to s1 begin end;

  trans
    from s1 to s2 when ....
end;
```

We now discuss how real-time constraints are constructed. The basic building blocks of temporal restrictions in Real-Time Estelle are *state descriptions*. State descriptions are basically boolean algebra expressions composed of atomic state propositions and boolean operators:

(i) If p is an atomic state proposition in a module, then it is a state description.
(ii) 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.

Atomic state propositions can be composed of any of the predicates defined in the Estelle standard concerning a module's state, plus the predicates we added in Subsection 7.2. A typical atomic proposition is WHEN(p.m), which is true iff the message m is at the head of the message queue of IP p. The operators AND, OR and IMPLIES have their usual boolean logic semantics. The expression p OTHERWISE q is semantically equivalent to (NOT p AND q) OR (p AND NOT q) and therefore to the boolean exclusive-or operator. The choice of the keyword, however, indicates that it can be used to express a difference between the desired behavior and the one the system has to show if the former cannot be provided. In this case, p describes a condition on the desired behavior, whereas q is a condition that holds if the desired behavior cannot be achieved [12] .

[12] This construct proves useful to support a run-time environment in deciding which behavior to support and what has to happen if this support fails, based on the Real-

Using the global time function `now` and *time variables* allows to refer to real-time in state descriptions. 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`[13], and the unit of time steps is given by the standard Estelle optional `timescale` clause.

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. It is allowed to add constants to time variables. Time variables occurring in a Real-Time Estelle expression have to be quantified over by `EXISTS` or `FORALL` clauses preceding the expression.

To specify temporal properties of states we 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`: `FORALL x:TIME; HENCEFORTH (p AND x=now IMPLIES EVENTUALLY (q AND now <= x+3));`. In addition to the operators described above, some abbreviations may be used. 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`. As the examples in Section 8 will show, these abbreviations make real-time constraints much more readable and easier to understand. With these definitions, the syntax of Real-Time Estelle is that of a first-order temporal logic with time variables and is similar to that of *Real-Time Temporal Logic (RTTL)* [62]. The complete syntax in BNF can be found in [26].

7.4 Semantics

The semantics we use for Real-Time Estelle is called complementary, since it is composed of two partial semantics: an operational part for the untimed portion of a Real-Time Estelle specification and a model-theoretic part for the

Time Estelle specification.

[13] As we argue above, only non-negative values should be assigned to variables of type `time`. However, Estelle does so far not possess a built-in type "non-negative integer".

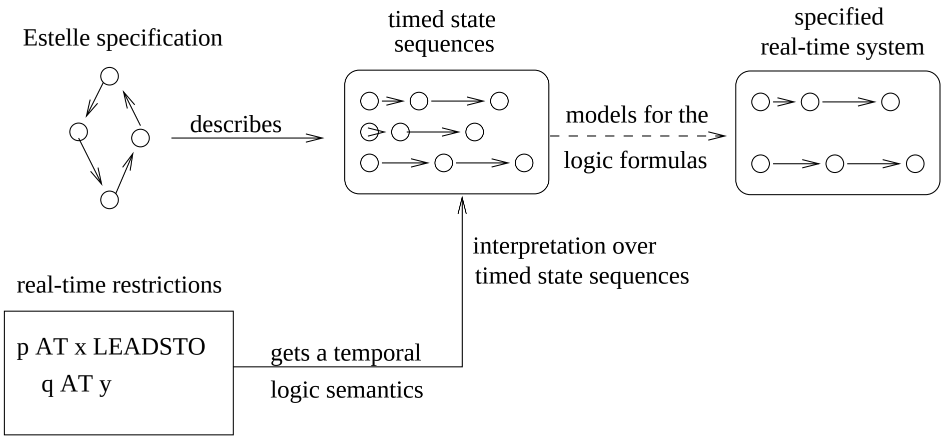

Fig. 2. Hybrid semantics for Real-Time Estelle.

real-time component of the language. Therefore, the specification of a system's behavior consists of timed state sequences constructed as follows: first, timed state sequences are obtained by using the operational semantics described in the Estelle Standard and the real-time model extensions given in Section 7.2. 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. This scheme is visualized in Figure 2.

A satisfaction relation $\models$ for Real-Time Estelle has been defined in [26]. It defines the conditions a timed state sequence must meet to fulfill Real-Time Estelle expressions. This semantics is, as well as the syntax, similar to that of RTTL [62].

7.5 An Example

The following (partial) Real-Time Estelle specification describes a module which receives a message, performs two transitions and outputs another message. The real-time constraint expresses the requirement that between reception of the first and sending of the second message no more than 5 units of time may elapse.

```
body m-behavior for m-interface;
  state s1, s2;

  time constraints
    FORALL x:time; HENCEFORTH (
    RECEIVING OF p1.m1 AT x LEADSTO
    SENDING OF p2.m2 AT (x < now < x+5));
  ...
  trans
    from s1 to s2 when p1.m1
      begin (* do some work *) end;

    from s2 to s1 begin
      (* do some more work *)
      output p2.m2;
    end;
end;
```

8 Specifications

Sender-Receiver System. We are now interested in illustrating the specification of QoS requirements using the languages that we have introduced. We will keep our running example low in functional complexity. While this facilitates the presentation, it is also in line with the typically low complexity of high-speed communication protocols and services. The running example is rather simplistic: We will consider a *sender-receiver system* (SRS) that consists of a `S` (sender) process, an `R` (receiver) process and an underlying `Medium` service. Users in the system's environment are sending data from the sender end to the receiver end of the communication link. We assume that the user at the sender end sends data by a `UDreq` service primitive, and receives `UDcon` or `UDrej` primitives that indicate successful and unsuccessful data transmission, respectively. A process `S` implements the sender side of the service by invoking an underlying `Medium` service using `MDreq`, `MDcon` and `MDrej` primitives in the obvious fashion. The `Medium` process (the behavior of which we do not explicitly specify) is assumed to be unreliable. However, we assume that it possesses the miraculous capability to detect, whether or not a data unit that has been sent could be successfully delivered at the receiver end of the connection. In case of successful delivery the data unit will be presented by an `MDind` primitive to the process `R` which in turn hands the same data over to the user at the receiving end using an `UDind` primitive. In case of unsuccessful delivery, process `S` is informed of the unsuccessful data transmission by a `MDrej` primitive. A `UDrej` indicates this circumstance to the user at the sender end of the connection. It will be the responsibility of higher layer protocols to provide for error-correcting mechanisms.

The SRS example captures some typical features of high-speed protocols, namely simple protocol functionality, absence of flow control, simplicity of the failure indication mechanism and absence of a retransmission mechanism. Compare with similar protocol mechanisms in ATM [27,50]. A graphical description of the Estelle specification's architecture for SRS is given in Figure 3 [14]. The functional specification of the `sender` and `receiver` Estelle module bodies can be found in Figure 4 [15]. The SDL version of the SRS example is given in two parts: Figure 5 presents the SDL system level diagram, and Figure 6 contains the behavior of the sender and receiver processes.

[14] Note that this diagram is not part of the Estelle specification for SRS.

[15] The specifications include `TIME CONSTRAINTS` sections. These will be filled by the real-time constraints developed later in this Section.

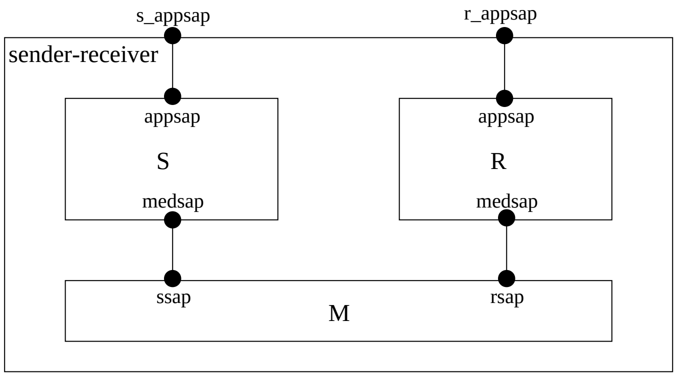

Fig. 3. Estelle Architecture of SRS

```
BODY sender_body FOR sender;
  STATE s1, s2;
  INITIALIZE TO s1;

  TRANS
    FROM s1 TO s2
    WHEN appsap.UDreq BEGIN
      OUTPUT medsap.MDreq;
    END;

  TRANS
    FROM s2
    WHEN medsap.MDrej TO s1 BEGIN
      OUTPUT appsap.UDrej;
    END;

    WHEN medsap.MDcon TO s1 BEGIN
      OUTPUT appsap.UDcon;
    END;

    TIME CONSTRAINTS
      (* to be filled *)
    END:

END;


BODY receiver_body FOR receiver;
  STATE s1;
  INITIALIZE TO s1 BEGIN END;

  TRANS
    FROM s1 TO same
    WHEN msap.MDind BEGIN
      OUTPUT appsap.UDind;
    END;

    TIME CONSTRAINTS
      (* to be filled *)
    END:
end;
```

Fig. 4. Estelle specification of sender and receiver modules for SRS

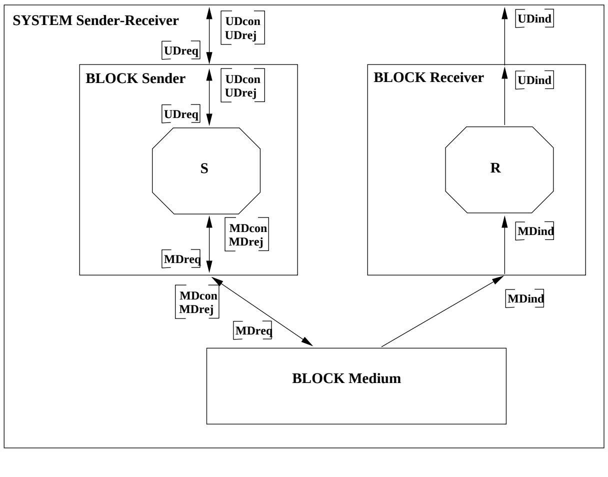

Fig. 5. SDL System Diagram of SRS

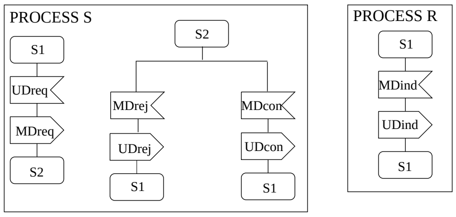

Fig. 6. SDL Process Diagrams for Sender-Receiver System

8.1 Application QoS Requirements

As argued in Section 3, application level QoS requirements are often formulated in the terminology of the end-user. Also, they often refer to the human perception rather than technical characteristics of the communication system. Therefore we will give two examples of how application level requirements can be translated in formally specified requirements on the communication system.

8.1.1 Inter-stream synchronization

In most existing multimedia systems, audio and video streams of the same application level connection are handled separately as two different streams. That makes handling of such multimedia applications more flexible, for instance by permitting different storage locations for video and audio data. An example for such an application is the distributed multimedia news-on-demand database described in [31]. However, synchronization becomes a problem, since video frames have to be played out at approximately the same time as the corresponding audio packets. According to [68] the video stream must not fall behind the corresponding audio stream by more than 120 ms since otherwise the end user perceives lip synchronization to be lost. Likewise, the video stream must not precede the audio stream by more than 15 ms to avoid loss of perceived lip synchronization.

In the SRS, we model two streams by sequences of `UDind` packets. Each packet has two parameters: the first indicates if the packet belongs to the audio (`a`) or the video stream (`v`), and the second contains the sequence number [16]. For the sake of simplicity we assume that each video packet is associated with exactly one audio packet with the same sequence number.

SDL/MTL. Inter-stream synchronization is a QoS requirement. The underlying communication subsystem has to provide for mechanisms guaranteeing

[16] We will, for instance, write $INPUT(UDind_v(n))$. Note that the parameter n can be interpreted as a variable. For the formal treatment of variables see Section 6.

this requirement. We introduce the specification of two different sorts of inter-stream synchronization, namely the *exact* and *time-bounded* variants.

Exact synchronization means that we require the audio as well as the video data unit to be received at the *same instant in time*. This leads to an interesting problem: SDL processes have just one input queue, so they can not receive messages simultaneously but only in a nondeterministically chosen order. We can offer two ways around this problem.

Concurrent video and audio systems: We assume that there is one instance of the SRS system for the video-data, and one for the audio-data and that both together form an SDL system. We'll write subscripts v for messages belonging to the video subsystem, and a for the audio subsystem. The processing of the audio and the video stream is handled outside the system by the user which is located in the environment. It would be tempting to specify the exact inter-stream requirement as

$$\Box(\mathit{INPUT}(UDind_v(n)) \equiv \mathit{INPUT}(UDind_a(n))).$$

This expresses that in the same global state in which $\mathit{INPUT}(UDind_v(n))$ holds, $\mathit{INPUT}(UDind_a(n))$ has to hold as well. However, based on our interleaving model there can be no such state, because in any global system state only one INPUT predicate can hold and the above formula is therefore non-satisfiable.

There is a finite supply of sequence numbers. Hence, when pairing video and audio packets that have been disambiguated by the sequence number we need to make sure that identical instances of a sequence number usage are being paired. Therefore, we introduce a time constant t_s chosen such that more than t_s time units pass between the i-th and the $i+1$-st usage of any sequence number n for all natural $i > 0$ [17].

We modify the above unsatisfiable formulation of the synchronization requirement using the $\Diamond_{=0}$ operator. We require that if we observe a video frame with sequence number n that is followed within t_s time units by an audio frame with the same sequence number at all, then we'll see this audio frame within 0 delay in a future state [18]:

$$\Box((\mathit{INPUT}(UDind_v(n)) \wedge \Diamond_{\leq t_s}\mathit{INPUT}(UDind_a(n)))$$

$$\supset \Diamond_{=0}\mathit{INPUT}(UDind_a(n))).$$

[17] For an estimation of t_S compare the number of frames transmitted per second in a multimedia application, which is typically less than 2^5, with the possible size of the range of a sequence number of 2^m where m is likely to be greater or equal to 8. TCP has recently been changed to have a 32 bit sequence number.

[18] Note that we are not interested in specifying any reliability or liveness properties here. I.e., it is not required that we will see a corresponding audio frame at all.

A similar relationship holds if we see the audio frame first:

$$\Box((\mathit{INPUT}(UDind_a(n)) \wedge \Diamond_{\leq t_s} \mathit{INPUT}(UDind_v(n)))$$

$$\supset \Diamond_{=0} \mathit{INPUT}(UDind_v(n))).$$

Luckily, we defined the time stamp component of our timed state sequence model such that it is weakly monotonic, allowing concurrent states with identical time stamps to appear in any order of an execution sequence. Consequently, this requirement is satisfiable.

Interleaving of video and audio stream: Now, we assume that the SRS system transmits the audio and the video stream along the same medium connection and consider the type to be a parameter of the messages received. The exact inter-stream synchronization requirement then reads

$$\Box((\mathit{INPUT}(UDind(v,n)) \wedge \Diamond_{\leq t_s} \mathit{INPUT}(UDind(a,n)))$$

$$\supset \Diamond_{=0} \mathit{INPUT}(UDind(a,n)))$$

$$\Box((\mathit{INPUT}(UDind(a,n)) \wedge \Diamond_{\leq t_s} \mathit{INPUT}(UDind(v,n)))$$

$$\supset \Diamond_{=0} \mathit{INPUT}(UDind(v,n))).$$

Note that the definition of MTL we have used does not allow for quantification over variables. We assume that there is a finite set of sequence numbers, and if we want to avoid quantification we have to repeat each of the formulas n times replacing the variable n with a constant. Alternatively, restricted quantification as in [9,10] can be used.

In the **time-bounded** model of inter-stream synchronization we do not require the two data streams to be synchronized with zero time delay. Instead we allow for an upper limit of 120 ms and 15 ms on the loss of synchronization of the audio and video streams, respectively. Assume $t_s > 120ms$. In this case, for the concurrent audio and video model the time-bounded inter-stream synchronization reads:

$$\Box((\mathit{INPUT}(UDind_v(n)) \wedge \Diamond_{\leq t_s} \mathit{INPUT}(UDind_a(n)))$$

$$\supset \Diamond_{\leq 15ms} \mathit{INPUT}(UDind_a(n)))$$

$$\Box((\mathit{INPUT}(UDind_a(n)) \wedge \Diamond_{\leq t_s} \mathit{INPUT}(UDind_v(n)))$$

$$\supset \Diamond_{\leq 120ms} \mathit{INPUT}(UDind_v(n))).$$

This extends to the interleaved video and audio streams in an obvious way.

Real-Time Estelle. Similar considerations apply to the Real-Time Estelle specification. For the concurrent systems approach, we assume an SRS with two sender and two receiver modules, handling audio and video streams separately. The SRS now has four external interaction points instead of two, namely `s_audiosap`, `s_videosap`, `r_audiosap` and `r_videosap`. A zero delay synchronization requirement to receive an audio and a video packet from the system at exactly the same point in time can be expressed as follows, taking advantage of Real-Time Estelle's time variables and the capability to compare them:

```
FORALL x,y: TIME; FORALL n: INTEGER; HENCEFORTH (
  (RECEIVING OF r_videosap.UDind_v(n) AT x   AND
   EVENTUALLY RECEIVING OF r_audiosap.UDind_a(n)
      AT (x <=y <= x + ts) )
   IMPLIES (x=y)
);

FORALL x,y: TIME; FORALL n: INTEGER; HENCEFORTH (
  (RECEIVING OF r_audiosap.UDind_a(n) AT x   AND
   EVENTUALLY RECEIVING OF r_videosap.UDind_v(n)
       AT (x <=y <= x + ts) )
   IMPLIES (x=y)
);
```

In this and all following Real-Time Estelle specifications the unit of time constants is determined by the `TIMESCALE` option. We assume that for all future examples the line "`TIMESCALE=milliseconds;`" is included.

The interleaving case can be modeled by the `r_appsap` interaction point of the original SRS specification for reception of both types of messages. Note that in Real-Time Estelle it is not necessary to list a formula for every possible value of n. We use a quantifier to express this in one formula. For the time-bounded version of inter-stream synchronization the bounded delay version of the specification reads:

```
FORALL x,y: TIME; FORALL n: INTEGER; HENCEFORTH (
  (RECEIVING OF r_appsap.UDind(a,n) AT x   AND
   EVENTUALLY RECEIVING OF r_appsap.UDind(v,n)
      AT (x <= y <= x + ts))
   IMPLIES ( y <= x+120)
);

FORALL x,y: TIME; FORALL n: INTEGER; HENCEFORTH (
  (RECEIVING OF r_appsap.UDind(v,n) AT x   AND
   EVENTUALLY RECEIVING OF r_appsap.UDind(a,n)
        AT (x <= y <= x + ts))
   IMPLIES (y <= x +15)
```

```
);
```

8.1.2 Frame Rate

End users describe the perceived moving image quality of a video connection in terms of "high vs. low quality". The technical representation of this characterization relates to the allocated video frame rate, i.e. the number for video frames that can be played out per second at the receiver end of the connection. A rate of 8 frames per second fails to give the impression of a moving picture. For high-quality video transmission, a rate of about 25 frames per second is required. In our specifications we assume that each `UDind` packet in SRS carries one video frame.

SDL/MTL. Frame rates are usually measured by the number of frames per time period, called the observation interval. A certain rate, however, can be achieved by transmitting a number of frames within a very short period of time, and then a few more at the end of the observation interval. However, SDL/MTL does not allow for counting of events in the MTL formula part. This would constitute a non-trivial extension of the logic [29]. We offer an approximation of this requirement by means of a reciprocal consideration: if we want to require that at least 25 frames be transmitted per second, then we assume that this can be reached by transmitting at least one frame every 0.04 sec. We therefore suggest the following exact inter-send time requirement as an approximation of the original frame rate requirement:

$$\Box(OUTPUT(MDind) \supset \Diamond_{\leq 0.04sec} OUTPUT(MDind)).$$

Real-Time Estelle. We make use of the additional expressiveness of Real-Time Estelle to formulate a more lenient requirement on the frame rate. We require that it should be about 25 frames per second. Our first formula reads:

```
FORALL x: TIME; FORALL z:integer; HENCEFORTH (
  SENDING OF r_appsap.UDind[z] AT x LEADSTO
    SENDING OF r_appsap.UDind[z+1] AT (x + 37 <= now <= x + 43)
);
```

However, using only this formula, the frame rate could vary between 23 and 27 frames per second. To further restrict the rate to a value near to 25 frames per second, we give a second formula, which requires that after the z-th frame has been sent, the $z + 24$-th frame should be sent between 997 and 1003 milliseconds later [19].

```
FORALL x: TIME; FORALL z: INTEGER; HENCEFORTH (
  SENDING OF r_appsap.UDind[z] AT x LEADSTO
    SENDING OF r_appsap.UDind[z+24] AT (x + 997 <= now <= x + 1003)
);
```

8.2 Transport QoS requirements

In this section we address QoS related properties at the transport connection layer. We do not strictly abide to the OSI model here, we are happy to consider, for example, the ATM Adaptation Layer a 'transport' layer.

8.2.1 Guaranteed Response of the transport system

Functional system properties are often distinguished into safety and progress properties, and into safety and liveness properties [58,16]. Both classifications are orthogonal. In the SRS example liveness and progress of the service provided by the medium is an important property to infer liveness of the service provided by the SRS system. One such property we wish to express is that if data transmission has been requested by a `MDreq`, then *eventually* we will either see a `MDind` or a `MDrej` message telling us about the success of our request [20]. Note that this is a purely functional property. However, neither SDL not Estelle is capable of expressing liveness properties. We show how Real-Time Estelle and SDL/MTL can also be used to specify these properties for SDL and Estelle specifications. Note that to specify guaranteed response we only need to use the untimed constructs of Real-Time Estelle und SDL/MTL.

SDL/MTL.

$$\Box(OUTPUT(MDreq) \supset \Diamond(INPUT(MDind) \vee INPUT(MDrej))).$$

Real-Time Estelle.

```
HENCEFORTH (
  SENDING OF medsap.MDreq LEADSTO (
```

[19] It should be noted that our specification technique is not stochastic. This specification does not specify an *average* frame rate of 25 frames in the sense that any system satisfying this specification would have a frame rate that converges to 25. Systems may satisfy this specification even if their frame rates converge to slightly less or more than 25, as the reader can easily check.

[20] Strictly speaking, this property is a conjunction of a safety and a liveness property, see [16].

```
    RECEIVING OF medsap.MDind OR
    RECEIVING OF medsap.MDrej
  )
);
```

8.2.2 Delay

Let us assume that the medium service in SRS is a service that provides real-time guarantees on the delivery of transmitted data units. [27] argues, for example, that an HDTV-quality [21] remote surgery system must be based on a telecommunications subsystem that guarantees an end-to-end delay of not more than 1 ms. To specify the QoS requirement of a delay bound of 1 ms on the sending and receiving of data units from the medium in the SRS example we use the following specifications [22].

SDL/MTL.

$$\Box(OUTPUT(MDreq) \supset \Diamond_{\leq 1ms}(INPUT(MDind) \vee INPUT(MDrej))).$$

Note that our SDL/MTL specification does not distinguish multiple instances of the signal types

Real-Time Estelle. Assuming a `TIMESCALE` of `milliseconds` we specify:

```
FORALL x: TIME; HENCEFORTH (
  SENDING OF medsap.MDreq AT x LEADSTO (
    (RECEIVING OF medsap.MDind OR
     RECEIVING OF medsap.MDrej) AT (now <= x + 1)
    )
);
```

Note that for both the SDL/MTL and the Real-Time Estelle specification of SRS we do not require the medium service to be reliable, we only require a positive or negative indication eventually to be given.

---

[21] High Definition Television, a high resolution digital TV standard.

[22] Note that the correctness of this specification hinges upon the fact that the SRS example uses a "stop-and-go" protocol. I.e. after observing an $OUTPUT(MDreq)$ signal the next instance of an $OUTPUT(MDreq)$ signal can only be observed in case an intermittent $INPUT(MDind)$ or $INPUT(MDrej)$ signal has been observed.

8.2.3 Jitter

Delay-related QoS requirements may become more subtle. Successive data units routed through a complex network may be subject to varying delays over time. The ATM service is, as one example, prone to this sort of delay variation [50]. The delay variation may be caused by changing network load which may lead to temporal congestion in network internal ATM switches, or by routing successive cells on different routes. When the application requires continuous media streams to be transmitted it may be necessary to limit the variation in the delay that successive data units experience. Multimedia applications which need to reconstruct continuous signals require data to be delivered within a time interval around the mean value of the transmission delay, depending on the coding scheme used. The delay variance is called *delay jitter* and formally defined as follows: let d_{min} denote the minimal and let d_{max}, $d_{min} < d_{max}$, denote the maximal delay between sending and receiving of a sequence of transmitted data units, then $J = d_{max} - d_{min}$ denotes the delay jitter [23].

Let us assume that the SRS system has some QoS mechanism (which we haven't specified) that guarantees a bound on the delay jitter and at the same time ensures reliable transmission. The following specifications then specify the QoS requirement bounding the delay jitter at the user interface. Let us furthermore assume that the sequence of packets models a video stream within a video conference. For this case, [33] found a maximum acceptable delay of $250ms$ and a maximum jitter of $10ms$. We require a maximum delay bound and set $d_{max} = 200ms$ and assume a minimum delay $d_{min} = 190ms$.

SDL/MTL.

$$\Box(INPUT(UDreq) \supset$$

$$(\neg OUTPUT(UDind)\ \mathcal{U}_{[190,200]}\ OUTPUT(UDind))).$$

Real-Time Estelle. We insert the following time constraint into the `sender-receiver` process specification of Figure 3:

```
FORALL x: TIME; HENCEFORTH (
  RECEIVING OF s_appsap.UDreq AT x IMPLIES (
    HENCEFORTH NOT (SENDING OF r_appsap.UDind
                      AT (x < now <= x + 190))
    AND
    EVENTUALLY SENDING OF r_appsap.UDind AT (now <= x + 200)
```

[23] Note that some absolute delay may be tolerable for some types of traffic like uni-directional broadcast, even when tight delay jitter bounds are crucial for the reasons explained above.

```
  )
);
```

If the communication service can guarantee that there will be no message losses, we can make use of Real-Time Estelle's instance operator in order to rewrite this specification more concisely:

```
FORALL x: TIME; FORALL z:integer; HENCEFORTH (
  RECEIVING OF s_appsap.UDreq[z] AT x LEADSTO
    SENDING OF r_appsap.UDind[z] AT (x+ 190 < now <= x + 200)
);
```

8.3 ATM AAL QoS requirements

8.3.1 Isochronicity

Isochronicity is a characteristic of communication systems supporting multimedia applications. It means that certain communication events, for example sending and receiving of multimedia data units, occur periodically at equally distanced points of time. This is important for continuous media applications that need to have video frames available for playout at isochronous instants in time in order to guarantee a user-perceived moving image QoS. The need for isochronicity depends on the coding scheme in use. Isochronicity is particularly important for simple coding schemes in which samples of the analog signal are taken and sent periodically without pixel-differential encoding and implicitly stored time stamps. Isochronicity is an example for an *intra-stream synchronization* QoS requirement.

Isochronous sending: We refer again to the SRS example and consider isochronous sending of `UDreq` messages from the user to the `Sender` process. Note that this is therefore a specification of environment behavior and not the specification of a QoS requirement.

SDL/MTL. The SDL formalization requires that within the right-open interval of t time units after sending a frame by a `UDreq` primitive it is not allowed to send another `UDreq` message, while the next `UDreq` message has to follow exactly t time units after its predecessor.

$$\Box(INPUT(UDreq) \supset (\neg\Diamond_{<t}INPUT(UDreq) \wedge \Diamond_{=t}INPUT(UDreq))).$$

Real-Time Estelle. The requirement could be formulated similarly to the one in SDL/MTL. However, we again make use of the instance operator and require that two *consecutive* data units arrive at a distance of t time units. Therefore, it is not necessary to forbid the arrival of data units within this interval:

```
FORALL x:TIME; FORALL z:integer; HENCEFORTH (
    RECEIVING OF s_appsap.UDreq[z] AT x LEADSTO
        RECEIVING OF s_appsap.UDreq[z+1] AT (now = x+t)
);
```

Isochronous receiving: On the receiver side, the receiving application process may require to have successive data units available at isochronous moments in time. This now turns out to be a QoS requirement imposed on the service provided by the `receiver` process.

SDL/MTL.

$$\Box(OUTPUT(UDind) \supset$$

$$(\neg\Diamond_{<t} OUTPUT(UDind) \wedge \Diamond_{=t} OUTPUT(UDind))).$$

Real-Time Estelle. Again, we use the instance operator:

```
FORALL x: TIME; FORALL z:integer; HENCEFORTH (
  SENDING OF r_appsap.UDind[z] AT x LEADSTO
    SENDING OF r_appsap.UDind[z+1] AT (now = x + t)
);
```

8.4 QoS Mechanisms

8.4.1 Delay Jitter Compensation

Guaranteeing a bound on the delay jitter of a transmission medium does not yet guarantee isochronous delivery of messages to an application, even if the source is sending data isochronously. In order to compensate the residual delay jitter and to guarantee an isochronous delivery of data units to a user it has been suggested to use a jitter compensation buffer between the network service and the user. In the context of ATM this buffer is often called *playout buffer* [50].

SDL/MTL. Assume that the process `R` in SRS has the functionality of a playout buffer, which can easily be implemented in SDL[24]. Henceforth, `R`

[24] Think of the following SDL process as implementing the buffer: If a message arrives, it will be stored using an SDL `SAVE` primitive. This will be done until the target filling is reached ([50] argues that this is approximately two ATM cells). Then, use a timer and replay the messages from the `SAVE` queue when the timer expires, or `SAVE` incoming messages.

accepts the possibly non-isochronous but jitter-bounded data stream from the Medium service by `MDind` signals. Every signal will be delayed for a minimum time span of t_1 time units. This means that the first data units in a stream will fill the buffer up to a certain threshold number. Then, at latest $t_2 > t_1$ time units after the arrival at the buffer the data units will be delivered to the user by means of a `UDind` signal. The delivery of successive `MDind` signals then occurs isochronously with an inter-signal delivery time of p, which should ideally correspond to the inter-send event time at the sender in order to ensure isochronous traffic with identical inter-send times on the sender and on the receiver side. The jitter compensation requirement for the process `R` then reads[25]:

$$\Box(INPUT(MDind) \supset (\Box_{\leq t_1} \neg OUTPUT(UDind) \wedge \Diamond_{\leq t_2} OUTPUT(UDind)))$$

$$\wedge \Box(OUTPUT(UDind) \supset \Diamond\!\!\!\cdot_{=p} OUTPUT(UDind)).$$

The first conjunct in this formula specifies a property of the playout buffer QoS mechanism, while the second conjunct specifies a QoS guarantee that this mechanism has to provide.

Real-Time Estelle. We follow the approach sketched for SDL and assume that module `receiver` has the functionality of a playout buffer. Unlike SDL, Estelle has no `SAVE` command. Instead, messages can be assigned to variables. Thus, the obvious implementation of the playout buffer is a ring buffer variable where incoming `MDind` messages are stored. A second transition reads the stored data from the ring buffer, encodes them in `UDind` messages and sends them out over the interaction point to the user. The real-time constraints for the receiver's playout buffer are partitioned into two Real-Time Estelle constraints and read as follows:

```
FORALL x: TIME; HENCEFORTH (
  ( RECEIVING OF medsap.MDind AT x FORBIDS
      SENDING OF r_appsap.UDind AT (x <= now <= x + t1))
  AND
  ( RECEIVING OF medsap.MDind AT x LEADSTO
      SENDING OF r_appsap.UDind AT (now <= x + t2))
);

FORALL x: TIME; HENCEFORTH (
```

[25] Note that while the previously sketched SDL implementation of the playout buffer used the SDL timer mechanism in an operational fashion to generate stimuli for replaying saved messages to the user. However, only the conjunction of this operational model with the following MTL formulas guarantees that the resulting model satisfies hard real-time isochronicity bounds.

```
  SENDING OF r_appsap.UDind AT x LEADSTO
    SENDING OF r_appsap.UDind AT (now = x + p)
);
```

The first constraint describes a minimum and a maximum time a packet has to stay within the playout buffer, while the second one ensures isochronicity[26].

8.4.2 Reaction on QoS Violation

The examples that we have shown so far provided specifications of requirements that tell the 'good' system behaviors from the 'bad' ones. If only *one* of the possible executions of a system violates one of the above requirement specifications, then this will invalidate the system with respect to the specification. However, systems will in some cases not become unusable in the event of a violation of some QoS guarantee. Instead, the system will raise an exception condition to indicate the QoS guarantee violation to an operator, and then proceed. We call this mechanism QoS *monitoring*. In the SRS example we require that whenever an `MDreq`, carrying one encoded video frame, is not followed within $200ms$ by either a `MDind` or `MDrej` data unit, which would correspond to meeting the QoS guarantee of positive or negative indication within $200ms$, then the sender process will send a signal `ALARM` within $220ms$ time units of having sent `MDreq`.

SDL/MTL. Assume that in the SRS example the sender process has the capability to indicate the violation of a delay QoS guarantee by the medium using a signal of type `ALARM`. Then the following specification ensures the proper functioning of this monitoring mechanism:

$$\Box(\neg(OUTPUT(MDreq) \supset \Diamond_{\leq 200ms}(INPUT(MDind) \vee INPUT(MDrej)))$$

$$\supset (\Box_{\leq 200ms}\neg OUTPUT(ALARM) \wedge \Diamond_{\leq 220ms} OUTPUT(ALARM))).$$

Real-Time Estelle. We use the keyword `OTHERWISE` to express that we prefer QoS not to be violated, but that there is a possible reaction if it happens.

```
FORALL x: TIME; HENCEFORTH (
  SENDING OF medsap.MDreq AT x LEADSTO (
    ( RECEIVING OF medsap.MDind
      OR
      RECEIVING OF medsap.MDrej )
```

[26] The formulation in Real-Time Estelle uses the abbreviations `FORBIDS` and `LEADSTO`. Note that in the earlier delay jitter example, instead of these keywords, the long forms have been used.

```
    AT (x <= now <= x + 200)
  OTHERWISE
    SENDING OF s_appsap.ALARM AT (x + 200 < now <= x + 220)
  )
  AND HENCEFORTH NOT SENDING OF s_appsap.ALARM AT (now <= x + 200)
);
```

8.4.3 QoS Negotiation

The traffic pattern dynamics in broadband communication systems make it necessary for the involved parties to negotiate and renegotiate QoS guarantees [42]. Assume the sender process in the SRS example has the capability of negotiating an increase in certain QoS guarantees with the underlying medium service. Let us also assume that there is an obvious QoS (re-)negotiation protocol that has been specified between the sender process and the medium: The application sending data via SRS requests an increase in delay bound by sending a `UINCreq` signal which the service forwards to the medium (`MINCreq`). We assume that there is an appropriate network management process maintaining the network resources inside the medium subsystem. The medium either grants the increase (`MINCcon`) or it refuses the increase (`MINCrej`). Both reactions are indicated accordingly to the user. We are not interested in the mechanism itself, but in specifying the effect that a successful renegotiation has. It may be useful to state that successful renegotiation entails a henceforth invariant property to hold, namely the newly established level of QoS guarantee. This is invariant until a new renegotiation is initiated. This may be useful in showing the correctness of other parts of the system that rely on the specified delivery bound.

SDL/MTL. As an example assume that a user was no longer satisfied with the medium delay and asked for a better video quality. The transmission protocols translate this request to a new maximum acceptable delay of $200ms$. We thus require that whenever $\mathit{INPUT}(MINCcon(200ms))$ has been executed, the delivery delay of the medium is henceforth limited to $200ms$, until another, arbitrary $\mathit{INPUT}(MINCcon())$ is observed. Note that we assume that there is a finite number of constants that can appear as an argument to the $MINCcon$ primitive.

$$\Box(OUTPUT(MINCcon(200ms)) \supset$$

$$(\Box((\mathit{INPUT}(\mathit{MDreq}) \wedge \Diamond OUTPUT(\mathit{MDind})) \supset \Diamond_{\leq 200ms} OUTPUT(\mathit{MDind})))$$

$$\mathcal{U}\ OUTPUT(MINCcon())).$$

Real-Time Estelle. A temporal *until* operator is not defined in Real-Time Estelle. The following specification can therefore not express the bounded invariance that holds until renegotiation takes place. The temporal context remains unbounded.

```
FORALL x,y: TIME; HENCEFORTH (
  SENDING OF medsap.MINCcon AT x LEADSTO (
    HENCEFORTH ((RECEIVING OF medsap.MDreq AT (y>=x) AND
        EVENTUALLY SENDING OF MDind) LEADSTO
   SENDING OF MDind AT (now <= y + 200)))
);
```

9 Concluding Remarks

In this paper, we have shown how the standardized FDTs Estelle and SDL could be enhanced in order to be suitable tools for the specification of typical requirements and characteristics of broadband and multimedia systems. The general idea is to describe a system's functional behavior with the standardized part of Estelle or SDL syntax, and the non-functional QoS aspects by variants of real-time temporal logic formulas. In the case of SDL these formulas complement the SDL specification, in the case of Real-Time Estelle they form part of the syntax of an extended Estelle language. We have shown examples of how to apply the two specification techniques to some typical QoS requirements, guarantees and mechanisms.

In this concluding Section we will first present a comparison of Real-Time Estelle and SDL/MTL. Then, we will give an outlook on future work. We discuss the application of our notations to formal validation and verification, and to automatic implementation. Finally, we discuss possible stochastic extensions to our deterministic approaches.

9.1 Comparison of Real-Time Estelle and SDL/MTL

Similarities between the the Real-Time Estelle and the SDL/MTL approach comprise the usage of an language based on communicating extended finite state machines for the basic functional properties, and real-time temporal logic notations for the non-functional QoS properties. We now mention differences between these two approaches.

Syntax: From a syntactical point of view Real-Time Estelle is a language extension, while SDL/MTL uses temporal logic formulas that complement SDL specifications. Consequently, in SDL/MTL two syntactically disjoint specifications are needed to express the overall system requirements while Real-Time Estelle specifications are just one syntactic unit. To favor the one or the other approach requires weighing uniformity of the language vs. modularity and abstraction in the specification. A complementary specification separates operational properties from temporal and real-time properties and hence supports separation of concerns. The Promela language and the XSPIN tool are a practical example for a complementing specification approach: XSPIN will accept Promela specifications of and check Promela models against complementing Linear Temporal Logic formulas [37].

Readability: In the selection of the syntax for its extension part, Real-Time Estelle was guided by simplicity, readability for human readers and similarity to the existing Estelle keyword set. Important design decisions were the selection of only ASCII keywords and the provision of suitable short forms for certain expressions (like `LEADSTO` and `FORBIDS`). The goal of SDL/MTL was mainly to remain close to standardized SDL for the functional specification and to add QoS requirements without needing to change the syntax of the functional specification.

The syntactic approaches we have chosen to account for real-time expressiveness have the nature of case-studies: It is easy to see that one could equally define a complementary specification approach based on Estelle and MTL as one could incorporate a Real-Time Estelle-like real-time syntax into SDL. We hope that our discussions will help those interested in accommodating the SDL and Estelle standards to new needs of expressiveness in choosing adequate syntax (and semantics, we hasten to add).

Semantics: The dynamic semantics of SDL is operationally defined in [41]. We found this semantics definition not suitable for our purposes and therefore exemplified an axiomatic approach to an SDL semantics based on Hoare triples. We applied this approach rigorously to a subset of the full SDL language. A more complete interpretation of Z.100 SDL based on our approach can be given. The semantic definition of Estelle in the standard is less rigorous than the SDL definitions. Again, we propose that the approach chosen for SDL can easily be adapted to defining an axiomatic semantics for Estelle. Note that [13] defines a predicate transformer-based axiomatic semantics for Estelle.

Expressiveness: The semantics of Real-Time Estelle's logic part has been designed with the goal of reaching suitable expressiveness. The logic is similar to Ostroff's RTTL [62]. MTL is less expressive than RTTL. It doesn't allow for counting of events or the use of variables over arbitrary domains. In terms of real-time expressiveness the ability to quantify over time variables allows Real-Time Estelle to express so-called 'non-local timing requirements' that MTL is unable to express [4]. Both Real-Time Estelle and SDL/MTL use future time operators. Others have found past-time operators useful in simplifying specifications [9–11]. While we were happy with the exclusive use of future time operators it should be noted that past time operators can easily be added both syntactically and semantically, and that their introduction does not add significant tractability problems.

Complexity: The increased expressiveness of Real-Time Estelle has to be paid for in terms of decision complexity. Satisfiability of MTL is EXSPACE-complete while it is non-elementary for Real-Time Estelle [4]. This means that one can hope that SDL/MTL specifications may be verified with some effort which is not the case for Real-Time Estelle.

9.2 Verification and Validation

The goal of a formal verification method for QoS requirements is to prove that a system specification S satisfies a set of QoS requirements Q. In particular, S may be the functional specification of a protocol or a service. The QoS guarantees that a service is capable of providing greatly depends on the performance of the underlying communications network. Let P denote a specification of the QoS guarantees implemented by the underlying communications network. Let us consider the SRS example again, and let us assume that the functional behavior of SRS is given as a logic specification $\mathcal{S}$. Assume the system performance to be described by the following minimal response time formula:

$$\mathcal{P} : \Box((\mathit{INPUT}(\mathit{MDreq}) \wedge \Diamond \mathit{OUTPUT}(\mathit{MDind})) \supset \Box_{<t_1} \neg \mathit{OUTPUT}(\mathit{MDind})).$$

Let a QoS requirement on SRS be described by the following formula:

$$\mathcal{Q} : \Box(\mathit{OUTPUT}(\mathit{UDreq}) \supset \Diamond_{\leq t_2}(\mathit{INPUT}(\mathit{UDcon}) \vee \mathit{INPUT}(\mathit{UDrej}))).$$

This gives rise to a verification problem, namely the question, whether based on $\mathcal{S}$ and $\mathcal{P}$ the QoS requirement $\mathcal{Q}$ can at all be satisfied, hence whether the assertion $\mathcal{P} \wedge \mathcal{S} \supset \mathcal{Q}$ holds. Intuitively, the answer depends amongst others on the choice of values for t_1 and t_2. To formally establish this conjecture it is necessary to employ formal verification methods. Amongst the numerous verification approaches in the literature, [1] contains an approach to the formal verification of temporal logic based real-time requirements, and [3] discusses real-time model checking algorithms. RT-Spin [69] is based on the model checking approach in [3].

9.3 Implementation

As we pointed out earlier, formal specifications can have multiple functions in the systems engineering process: they can be abstract requirements models or implementation-biased design specifications. The implementation of a design specification given in Real-Time Estelle needs to be based on a real-time implementation environment since otherwise, no guarantees for the specified real-time constraints can be given. In [25], a method for the automatic implementation of Real-Time Estelle specifications in a real-time operating system environment has been suggested. The general idea is to map each real-time enhanced Estelle module in the specification onto one thread of the operating system and derive the scheduling parameters from the real-time constraints in the specification.

9.4 Stochastic Extensions

[46] distinguishes *deterministic* and *statistical* QoS guarantees. We have silently assumed that QoS can be treated as a deterministic phenomenon. The stochastic nature of some broadband and multimedia systems may make it necessary to express requirements in a stochastic fashion. An example is the requirement that with a probability of p the cell transfer delay in ATM will be less than t time units. There have been a number of approaches that combine temporal logic, real-time and probabilities [2,32]. In these logics formulas do not only need to be satisfied by a timed state execution model, they also need to satisfy accumulated path probabilities in a given Markov chain model.

We informally describe this approach. Intuitively, let $\Diamond^{\geq a}_{\leq t} p$ denote the requirement that with a probability of at least a within the next t time units p will hold. The real-time annotation can be omitted in which case the formula is purely stochastic. This allows us to express the idea of *stochastic reliability*, namely that if a data unit is sent then it will with a probability a with $0 < a \leq 1$ be eventually received:

$$\Box(\mathit{INPUT}(\mathit{MDreq}) \supset \Diamond^{\geq a} \mathit{OUTPUT}(\mathit{MDind})).$$

An interpretation of this requirement in the context of ATM is that the cell loss rate is $< 1 - a$. The more meaningful requirement, however, is that such a cell loss rate will be achieved within a finite interval of t time units, which can be expressed as follows:

$$\Box(\mathit{INPUT}(\mathit{MDreq}) \supset \Diamond^{\geq a}_{\leq t} \mathit{OUTPUT}(\mathit{MDind})).$$

Acknowledgements

The authors wish to thank the anonymous referees for their detailed and helpful comments. Reinhard Gotzhein contributed further to debugging the paper. The research of the second author was in part supported by the Natural Sciences and Engineering Research Council (NSERC) of Canada.

References

[1] M. Abadi and L. Lamport. An Old-Fashioned Recipe for Real Time. In *[21]*, pages 1–27, 1992.

[2] R. Alur, C. Courcoubetis, and D. Dill. Model Checking for Probabilistic Real-time Systems. In J. L. Albert, B. Monien, and M. R. Artalejo, editors, *International Colloquium on Automata, Languages and Programming*, volume 510 of *Lecture Notes in Computer Science*. Springer Verlag, 1991.

[3] R. Alur and D. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

[4] R. Alur and T. A. Henzinger. Logics and models of real-time: A survey. In *[21]*, pages 45–73, 1992.

[5] F. Anger. On Lamport's Interprocess Communication Model. *ACM Transactions on Programming Languages and Systems*, 11(3):404–417, July 1986.

[6] C. Aurrecoechea, A. Campbell, and L. Hauw. A Survey of QOS Architectures. *Multimedia Systems Journal, Special Issue on QoS Architectures*, 1997. To appear.

[7] F. Belina, D. Hogrefe, and A. Sarma. *SDL with Applications from Protocol Specification.* Prentice Hall International, 1991.

[8] S. Ben-David. The global time assumption and semantics for concurrent systems. In *Principles of Distributed Computing*, pages 223 – 231. ACM, 1988.

[9] G. S. Blair, L. Blair, and J. B. Stefani. A specification architecture for multimedia systems in open distributed processing. *Computer Networks and ISDN Systems*, 29:473–500, 1997.

[10] L. Blair, G. Blair, H. Bowman, and A. Chetwynd. Formal specification and verification of multimedia systems in open distributed processing. *Computer Standards and Interfaces*, 17:413–436, 1995.

[11] H. Bowman, G. Blair, L. Blair, and A. Chetwynd. Time versus abstraction in formal descriptions. In R. L. Tenney, P. D. Amer, and M. Ü. Uyar, editors, *Formal Description Techniques, VI*, pages 467–482. Elsevier Science Publishers B.V. (North–Holland), Amsterdam, 1994.

[12] D. Brand and P. Zafiropulo. On communicating finite-state machines. *Journal of the ACM*, 30(2):323–342, Apr. 1983.

[13] J. Bredereke, R. Gotzhein, and F. H. Vogt. Design of a formal Estelle semantics for verification. In *[23]*, pages 153–168, 1993.

[14] M. Broy. Towards a Formal Foundation of the Specification and Description Language SDL. *Formal Aspects of Computing*, 3(1):21–57, 1991.

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

[16] E. Chang, Z. Manna, and A. Pnueli. The safety-progress classification. In *subseries F: Computer and System Science*, NATO Advanced Science Institutes Series. Springer-Verlag, 1992.

[17] K.-T. Cheng and A. S. Krishnakumar. Automatic functional test generation using the extended finite state machine model. In *Proceedings of the 30th Design Automation Conference DAC-93*, pages 86–91, 1993.

[18] C. Courcoubetis, editor. *Computer Aided Verification: Proceedings of CAV'93*, volume 697 of *Lecture Notes in Computer Science.* Springer Verlag, 1993.

[19] 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.

[20] A. M. Davis. *Software Requirements: Objects, Functions and States.* Prentice-Hall, 1993.

[21] J. W. de Bakker, C. Huizing, W. de Roever, and G.Rozenberg, editors. *Real-Time: Theory in Practice*, volume 600 of *Lecture Notes in Computer Science.* Springer-Verlag, 1992.

[22] P. Dembiński and S. Budkowski. Simulating Estelle specifications with time parameters. In Rudin and West [65], pages 265–279.

[23] M. Diaz and R. Groz, editors. *Formal Description Techniques, V.* IFIP Transactions C-10, Proceedings of the Fifth International Conference on Formal Description Techniques. North-Holland, 1993.

[24] E. A. Emerson. Temporal and modal logic. In J. v. Leeuwen, editor, *Handbook of Theoretical Computer Science*, chapter 16. Elsevier Science Publishers B. V., 1990.

[25] S. Fischer. Implementation of multimedia systems based on a real-time extension of Estelle. In Gotzhein and Bredereke [30], pages 310–326.

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

[27] D. Ginsburg. *ATM solutions for enterprise networking.* Addison Wesley, 1996.

[28] R. Gotzhein. Temporal logic and applications – a tutorial. *Computer Networks and ISDN Systems*, 24(3):203–218, 1992.

[29] R. Gotzhein. *Open distributed systems: on concepts, methods, and design from a logical point of view.* Vieweg advanced studies in computer science. Friedr. Vieweg & Sohn Verlagsgesellschaft mbH, Braunschweig/Wiesbaden, Germany, 1993.

[30] R. Gotzhein and J. Bredereke, editors. *Formal Description Techniques IX – theory, application and tools*, Kaiserslautern, Germany, 1996. Chapman & Hall.

[31] A. Hafid and G. v. Bochmann. Quality of Service Negotiation in News-on-Demand Systems: An Implementation. In A. Azcorra, T. D. Miguel, E. Pastor, and E. Vazquez, editors, *Proceedings of the Third International Workshop on Protocols for Multimedia Systems, Madrid, Spain*, pages 221–240, Oct. 1996.

[32] H. A. Hanson. *Time and Probability in Formal Design of Distributed Systems.* PhD thesis, Uppsala University, Sweden, 1991.

[33] D. Hehmann, M. Salmony, and H. J. Stüttgen. Transport services for multi–media application on broadband networks. *Computer Communications*, 13(4):197–203, 1990.

[34] T. A. Henzinger. *The Temporal Specification and Verification of Real-Time Systems.* Phd thesis, Stanford University, Department of Computer Science, August 1991. Also published as Report No. STAN-CS-91-1380.

[35] T. A. Henzinger, Z. Manna, and A. Pnueli. Timed Transition Systems. In de Bakker et al. [21], pages 226–251.

[36] G. J. Holzmann. *Design and Validation of Computer Protocols.* Prentice-Hall International, 1991.

[37] G. J. Holzmann. The verification of concurrent systems. AT&T Bell Laboratories, to be published by Prentice-Hall, 1995.

[38] Information processing systems — Open Systems Interconnection — Estelle: A formal description technique based on an extended state transition model. International Standard ISO 9074, 1989.

[39] ISO/IEC JTC1/SC21. Quality of service - basic framework - working draft #4, July 1994.

[40] ITU-T. Recommendation Z.100: Specification and Description Language (SDL). Geneva, Switzerland, 1993.

[41] ITU-T. Recommendation Z.100: Specification and Description Language (SDL), Annex F3: Dynamic semantics. Geneva, Switzerland, 1993.

[42] ITU-T. Recommendation I.371: Traffic control and congestion control in B-ISDN. Geneva, Switzerland, 1995. Temporary Document.

[43] R. Koymans. *Specifying Message Passing and Time-Critical Systems with Temporal Logic*. PhD thesis, Technical University of Eindhoven, 1989.

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

[45] A. S. Krishnakumar. Reachability and recurrence in extended finite state machines: Modular vector addition systems. In *[18]*, pages 111–122, 1993.

[46] J. Kurose. Open issues and challenges in providing quality of service gurantees in high-speed networks. *ACM Computer Communication Review*, 23(1):6–15, 1993.

[47] L. Lamport. Specifying concurrent program modules. *ACM Transactions on Programming Languages and Systems*, 5(2):190–222, Apr 1983.

[48] L. Lamport. The mutual exclusion problem: Part I – a theory of interprocess communication. *Journal of the ACM*, 33(2):313–326, April 1986.

[49] L. Lamport. The Temporal Logic of Actions. *ACM Transactions on Programming Languages and Systems*, 16(3):872–923, May 1994.

[50] J.-Y. Le Boudec. The asynchronous transfer mode: a tutorial. *Computer Networks and ISDN Systems*, 24:279–309, 1992.

[51] L. Léonard and G. Leduc. An introduction to ET-LOTOS for the description of time-sensitive systems. *Computer Networks and ISDN Systems*, 29(3):271–292, 1997.

[52] S. Leue. QoS specification based on SDL/MSC and temporal logic. In G. v. Bochmann, J. de Meer, and A. Vogel, editors, *Proceedings of Workshop on Distributed Multimedia Applications and Quality of Service Verification*, Montreal, Quebec, Canada, May 1994.

[53] S. Leue. Specifying real-time requirements for SDL specifications – a temporal logic-based approach. In P. Dembiński and M. Średniawa, editors, *Proceedings of the Fifteenth International Symposium on Protocol Specification, Testing, and Verification PSTV'95*. Chapman & Hall, 1995.

[54] 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.

[55] M. T. Liu. Protocol engineering. In M. C. Yovitis, editor, *Advances in Computers*, volume 29, pages 79–195. Academic Press, Inc., 1989.

[56] G. Luo, A. Das, and G. v. Bochmann. Software testing based on SDL specifications with Save. *IEEE Transactions on Software Engineering*, 20(1):72–87, 1994.

[57] N. Lynch and F. Vaandrager. Forward and Backward Simulation for Timing-Based Systems. In de Bakker et al. [21], pages 397–446.

[58] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification.* Springer-Verlag, 1992.

[59] 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.

[60] S. Mørk, J. Godskesen, M. Hansen, and R. Sharp. A timed semantics for SDL. In Gotzhein and Bredereke [30], pages 295–309.

[61] X. Nicollin and J. Sifakis. An Overview and Synthesis of Timed Process Algebras. In de Bakker et al. [21], pages 526–548.

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

[63] J. Quemada and A. Fernandez. Introduction of Quantitative Relative Time into LOTOS. In Rudin and West [65], pages 105–121.

[64] 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.

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

[66] H. Saito, T. Hasegawa, and Y. Kakuda. Protocol verification system for SDL specifications based on acyclic expansion algorithm and temporal logic. In K. Parker and G. Rose, editors, *Formal Description Techniques, IV: Proceedings of the Third International Conference on Formal Description Techniques*, pages 511–526. North-Holland, 1992.

[67] 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.

[68] R. Steinmetz and C. Engler. Human Perception of Media Synchronization. Technical Report 43.9310, IBM European Networking Center, Heidelberg, Germany, 1993.

[69] S. Tripakis and C. Courcoubetis. Extending Promela and Spin for real time. In T. Margaria and B. Steffen, editors, *Tools and Algorithms for the Construction and Analysis of Systems, Proceedings of the Second International Workshop, TACAS'96*, volume 1055 of *LNCS*, pages 329–348. Springer Verlag, 1996.

[70] T. Tsang and R. Lai. Time Estelle: An Extended Estelle Capable of Expressing Multimedia QoS Parameters. In *IEEE Int. Conf. on Multimedia Computing and Systems (ICMCS'97), Ottawa, Canada.* IEEE Computer Society Press, 1997. To appear.

[71] G. v. 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.