QoS Specification based on SDL/MSC and Temporal Logic

Stefan Leue*[1]

*Institute for Informatics, University of Berne.

Revised Version

Abstract. Many telecommunications systems engineers find it convenient to specify functional properties of their systems using state-transition based formal description techniques like SDL or Message Sequence Charts (MSCs). However, the expressiveness of these techniques does not capture Quality of Service (QoS) or Network Performance (Np) requirements because many of these rely on real-time bounds and probability constraints which these FDTs do not allow to express. However, suitably extended temporal logics allow for a description of these requirements. We introduce a method for the integration of functional system specifications given in SDL or MSC with temporal logic based specifications of QoS or Np requirements. We show how SDL and MSC specifications fit together with Temporal Logic specifications. Then we give examples of delay bound, delay jitter, isochronicity, stochastic reliability and stochastic delay bound constraint specifications. We discuss how our method helps in the specification of Np/QoS mapping problems, of QoS negotiation mechanisms, and of QoS monitoring. Finally we hint at methods for the formal verification of QoS and Np specifications.

1. Introduction

In telecommunications systems design there is a long history of the use of formal descriptions for the specification of functional properties of these systems. Functional properties are related to a description of the admissible sequences of events in a system. Many authors have advocated the use of formal description techniques (FDTs) in the design of telecommunications systems. SDL [CCI92a] and Message Sequence Charts (MSCs) [CCI92b] are frequently used for this purpose, and they will therefore play a central role in this paper.

In addition to the functional aspects real-time mechanisms have been used in the design of, in particular,

[1] The work was partly supported by the Swiss National Science Foundation and partly by the Swiss Federal Office for Education and Scientific Research. The work was carried out in the context of the collaboration of the University of Berne and the CEC RACE R2088 Project TOPIC, and in the context of the Δ^2 joint project of the University of Berne and the Swiss Federal Institute of Technology at Lausanne. The results presented in this document do not necessarily represent the opinion of some or all members of the TOPIC consortium.

Correspondence and offprint requests to: S. Leue, Institut für Informatik, Universität Bern, Länggassstr. 51, CH-3012 Bern, Switzerland, e-mail: leue@iam.unibe.ch. Paper presented at: The Montreal Workshop on Multimedia Applications and Quality of Service Verification, Montreal, Canada, June 1994.

protocols in order to ensure first progess of the system, and second in order to detect errors like message losses or unavailability of resources. For the use of the SDL timer mechanism in order to achieve the second goal see for example [BHS91]. However, the asynchronous SDL timer mechanism is unsuited to ensure progress of the system specified [HL92].

Multimedia and high bandwidth telecommunications services are characterised by so-called *Quality of Service* (QoS) requirements which address in particular real-time and probability aspects of the behaviour. Examples are maximum delay bounds, delay jitter bounds, or cell loss requirements for ATM [LB92]. Classical FDTs like SDL or MSC are incapable of expressing this sort of requirements. Therefore, in this paper we present a method for the use of so-called complementary temporal logic specifications. We start with specifications of the 'pure' functional behaviour of communication protocols and services given in SDL or MSC, and add specifications of real-time and probability aspects in suitably extended temporal logics.

Overview. In Section 2 we introduce a state transition model and show how SDL specifications can be interpreted based on this model. This transformation step is similar to the interpretation of SDL specifications as communicating extended finite state machines, for which some suggestions exist in the literature (see Section 2). However, the existing proposals are either informal, incomplete, or they do not adequately capture the SDL semantics, in particular the potentially iterative structure of SDL process transitions and the particular semantics of the SDL `INPUT` statement. In Section 3 we show how temporal logics can be used in combination with SDL specifications. The underlying idea is that both the SDL specification as well as the temporal logic specification constrain the allowable behaviour of the system, and we require both specifications to be satisfied by a system. We say that the temporal logic specification is a *complementary* specification of the basic SDL specification. Temporal logic, however, allows for the specification of state-based properties whereas we are often interested in specifying properties of sequences of events, such as `INPUT` and `OUTPUT` events. We show in Section 3 how these events can be defined in terms of state predicates. In this Section we also give a brief introduction into propositional and metric temporal logic [MP92, AH92], and we introduce informally a probabilistic metric temporal logic. In Section 4 we briefly review previous work on a finite state, state transition based semantics for MSCs, including an explanation how temporal logics can be used in this context.

In Section 5 we specify a range of different real-time constraint based QoS requirements complementing SDL and MSC specification examples. These include service response and message transmission delay bounds, delay jitter bounds, isochronicity related requirements, and requirements on transmission rates. Probabilistic QoS requirements like stochastic reliability (which corresponds to data unit loss rates) are being discussed in Section 6. In Section 8 we exemplify how a some QoS related mechanisms can be specified using our approach, like QoS negotiation and reaction to QoS guarantee violation. Section 7 exemplifies the use of our specification method in the context of QoS/Network Performance (Np) problems. In Section 9 we discuss some issues concerning the formal verification of QoS requirements. We conclude our discussion in Section 10.

Related work. A wide range of literature is available on many of the topics discussed in this paper. We will mention the appropriate references in places where they are relevant. However, some general literature on QoS should be mentioned here. The proposed standardisation of QoS concepts in the ISO/OSI and ODP context is documented in [ISO93]. [Kur93] gives a complementary overview over QoS topics.

2. A State-Transition based Model for SDL Specifications

In this Section we define a rudimentary computational model for SDL specifications, a so-called *Global State Transition Systems.* The computations which these STS represent will later on serve as models for complementary Temporal Logic specifications. The main components of this model are:

- *Process control and data manipulation.* This component describes the local behaviour of an SDL process which consists in transitions between symbolic states. In the course of the transitions the values of variables can be manipulated.
- *Communication.* SDL processes communicate via infinite queues, and each SDL process has exactly one input queue[2]. We describe the local state of an SDL process as the combination of current values for the data variables, the point of local process control, and the state of the input queue.
- *Global 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. We model this concurrency aspect by an interleaving semantics approach which is consistent with the SDL semantics as defined in [CCI92a]. In a given GSS a number of transitions of the individual SDL processes may be enabled. A nondeterministic algorithm decides which of the enabled transitions is selected for execution. The execution of this transition changes the local state of the respective process and, in case an `OUTPUT(X)` statement is executed, the local state of the receiving process by adding the signal `X` to the tail of its input queue. The result is a new global system state.

Related Work. Our definitions here are close to the definitions of state transition systems in [MP92] where they are called *Basic Transition Systems*. They can be seen as a generalisation of *Finite State Machines* (FSM) and *Extended Finite State Machines* (EFSM). A partly formal introduction to FSM and EFSM can be found in [Liu89]. EFSM are usually distinguished from FSM in that they allow the explicit representation of data variables and symbolic control states. Typically, due to the infinite range of data variables EFSM represent infinite state spaces. The modeling of SDL processes as EFSM has been suggested in [BHS91]. However, as we will see later the mapping of SDL process transitions as informally described in [BHS91] is too coarse in order to represent the structure and the complexity of the computation ocurring in the course of an SDL transition. A similar criticism applies to the formalization given in [SHK92]. [LDvB94] contains a formalization of SDL based on FSM, hence without treating data variables over infinite domains. Formalizations of EFSM can be found in [Hol91] (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 [CK93] and [Kri93] (from where we take part of our formalization). [BZ83] describes and formalizes the use of queues to model the collective behaviour of concurrent FSM which communicate asynchronously via queues (there called *protocols*). We use a similar approach when constructing a global state transition system (called *SDL specification* in this paper).

2.1. Process State Transition Systems

The process state transition systems (pSTS) we define here represent an SDL process by a set of symbolic states, a set of program variables (consisting of control and data variables), and by its interactions with the environment (input and output of signals). The 'logic' of an SDL process is encoded in its state transition relation. A state transition relates a current state of the system (involving enabling conditions on the current values of the program variables) and an input signal to a successor state (including an update of the program variables) and an output signal.

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*,

[2] For reasons of conciseness we do not address inter-process communication mechanisms like *viewing* or *remote procedure call*, their treatment within our framework is straightforward.

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. By $s[x]$ we denote the value of variable x in state s. We denote the set of all variables by Σ. Apparently, Σ can be infinite.

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 $\vdash_P$ which relates assertions about the system state to system states for a given pSTS P[3]. In particular, we write $s \vdash p$ iff state s satisfies state-proposition p[4]. Now, in order to relate states s and s' we say that $(s, s') \in T$ iff

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

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 $s_0 \vdash C$, i.e. s_0 is the initial state. In state formulae, 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 $s \vdash en(\tau_k)$ iff $s \models P_k$. For a pair of states (s, s') we say the transition τ_l has been *taken* iff $s \vdash en(\tau_l)$ and $s' \vdash Q_l$. We denote this by $ta(s, s', \tau_l)$

Input queue formally. Let the variables X and Y range over the queues of a pSTS, ie. over sequences of 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.

2.2. Interpreting SDL-Processes as pSTS

We introduce the mapping of an SDL process specification to the components of an pSTS P. We map the SDL states, as indicated by the `STATE` and `NEXTSTATE` keywords, onto elements of S. The statements which are executed in the course of an SDL specification are mapped onto the pre- and postcondition predicates relating two symbolic states. At this stage we consider local pSTS and therefore we only interpret `INPUT` statements, `OUTPUT` statements will be formally interpreted later when we construct a global state transition system out of a set of pSTS.

Formal treatment of `INPUT` statements. Table 1 shows the mapping of an SDL transition to transitions τ_j of the corresponding pSTS. We need to observe the following particularity of the semantics of `INPUT` statements. When executing a transition associated with an `INPUT(X)` statement the process reads the value

---

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

[4] We will not define all details of the relation $\vdash$ formally. For a detailed description of how to define such a relation for a given state transition system we refer the reader to [MP92]. We will concentrate here on the formal definition which are particular for an SDL process based pSTS.

of the head of the input queue and assigns its value, if any, to a local variable with the name `X`[5]. More precisely: it is first checked, whether the signal at the head of the queue is of type `X`, if this is true then the signal is consumed as described above. However, if the signal at the head of the queue does not have the expected type, i.e. it is not of type `X`, then the message is removed from the head of the queue, discarded, and the same `INPUT` statement is re-enabled. Therefore, even though Table 1 only contains one transition we need two transition predicates τ_1 and τ_2 to reflect this mechanism. The SDL transition in Table 2 involves

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

```
STATE S1;
  INPUT(A);
  OUTPUT(B);
  NEXTSTATE S2;
```

Table 1. SDL Transition

the update of the local variable x. The SDL Transition in Table 3 exemplifies the modeling of a decision

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

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

Table 2. SDL Transition with variable assignment

predicate, indicated by the keyword `DECISION`.

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	$\pi = S1 \wedge Q = CX \wedge C \neq A$	$\pi' = S1 \wedge Q' = X$

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

Table 3. SDL Transition with decision predicate

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. This works in all those cases in which an SDL transition represents a finite linear sequence of operations. However, SDL transitions may also be iterative structures, for an example see the loop in the control flow in Table 4. To model this we introduce further symbolic states which correspond to locations in the control flow where the control lies when the process jumps back to a label. In the example in Table 4 we introduced an additional symbolic state `S1-1`, corresponding to label `l1`.

Input/Output labeling of transitions. It is sometimes useful to specify assertions not only on state variables, but referring to interactions with the environment, i.e. concerning input or output of signals that are about to take place or that have just been executed. These communication events occur in the course of state-transitions[6], therefore we need to encode these state transitions in our state proposition language. Technically, we introduce two relations *inlabel* and *outlabel* which label the transitions of the pSTS with the `INPUT` or `OUTPUT` statements according to which of these statements are executed in the course

[5] 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 [LDvB94].

[6] Adopting the terminology of [Lam94] we may say that they represent *actions*.

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	$\pi = S1-1 \wedge \neg D(A)$	$\pi' = S1-1 \wedge A' = A-1$

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

Table 4. SDL Transition with decision predicate and looping transition branch.

of one transition. We omit the straightforward technical construction of this labeling here. If we consider the example in Table 4 we see that for example $\mathit{inlabel}(\tau_3) = \{$`INPUT(A)`$\}$ and $\mathit{outlabel}(\tau_3) = \{$`OUTPUT(B)`$\}$. Now, we augment these labels to state propositions in the following way. We say that $s \vdash \mathit{INPUT}(A)$ iff $(\exists r \in \Sigma, \tau_i \in \mathcal{T})(ta(r,s,\tau_i) \wedge ($`INPUT(A)`$\in \mathit{inlabel}(\tau_i))$ and $s \vdash \mathit{OUTPUT}(A)$ iff $(\exists r \in \Sigma, \tau_i \in \mathcal{T})(ta(r,s,\tau_i)) \wedge$ (`OUTPUT(A)` $\in \mathit{outlabel}(\tau_i))$.

2.3. Global State Transition Systems

SDL Specifications Formally. In the previous section we considered single SDL processes and formalized their state-transition behaviour. SDL specifications, however, consist of collections of concurrent SDL processes. Therefore, we say that an *SDL specification* P is a tuple $P = (P^0, \ldots, P^n)$ where each P^i for $i = 1, \ldots, n$ is a pSTS. P^0 represents the environment. It 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 it may at any instant send any signal of its output alphabet. SDL processes communicate asynchronously via infinite queues. There is one input queue per SDL process. For an SDL specification 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 $A \in I^j$[7]. 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 the previous section, an `INPUT(A)` statement represents an action purely local to an SDL process which in the SDL terminology is often just referred to as *signal-consumption*.

Transition Predicates for OUTPUT statements. As a sending and a receiving process are involved in transitions labeled `OUTPUT` one can not formalize these transitions by state propositions that solely refer to state variable of only one process. Table 5 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.

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	$\pi^1 = S1 \wedge Q^1 = CX \wedge C \neq A$	$\pi'^1 = S1 \wedge Q'^1 = X$

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

Table 5. Formalization of an output statement in an SDL Transition

[7] In SDL this involves a mapping of signal names via signal lists to signal routes which point to the receiving process.

Global System State. Let $P = (P^0, \ldots, P^n)$ denote an SDL specification. 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 $i = 1, \ldots, n$.

Global System State Sequences. We now extend the notions of admissible state sequences to GSS. In the course of each change of the GSS exactly one pSTS changes its local system state. We assume an interleaving model of global system state sequences to model the concurrency of an SDL specification. This means that in a given GSS s a daemon decides nondeterministically which out of all enabled transitions of all pSTS is going to be executed next, giving 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 interpretation of the state propositions *en*, *ta*, *INPUT* and *OUTPUT* extend in the obvious way from pSTS states to GSS.

Satisfaction of an SDL Specification. Based on the above definitions we can now give the definition of a satisfaction relation $\models_{\mathrm{SDL}}$ for SDL specifications. Let P an SDL specification and let Σ_P^ω the set of all infinite sequences of GSS of P. For an $s \in \Sigma_P^\omega$ we write $s \models_{\mathrm{SDL}} P$ iff s is an admissible sequence.

3. Using Temporal Logic for SDL Specifications

Many authors have advocated the use of temporal logics for the specification of communication protocols and services (see for example [WZ92] and [Got92, Got93]). These characterizations are accomplished by specifying constraints on event sequences in terms of an appropriately chosen temporal logic language. For an overview over temporal logics we refer to [Eme90]. In the remainder we will use a temporal logics similar to the logic described in [MP92], called Propositional Temporal Logic (PTL), and extensions based on PTL. However, it should be fairly easy to transfer formulae into other temporal logic framework, for example into the Temporal Logic of Action TLA (see [Lam93]).

A State Proposition Language. When using complementary QoS specifications we need to identify a set of state propositions which we may use in the temporal logic formulae. When determining the set of state propositions one also determines which part of the state information is observable[8]. We will not treat this question in depth here. The determination of the visible state component is mainly a question of the individual specification problem. We assume that the state propositions we use all refer to observable components of the system state, and we use in particular the following state propositions for an SDL specification P

- *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.
- *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.
- Data: we allow the reference to visible data variables and allow standard comparison operators on the variables. However, we require that the resulting expressions remain state propositions.

We allow state formulas to be constructed by using boolean operators between state propositions. E.g., 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.

[8] This corresponds to the determination of the sets $\mu(\tau)$ in [AH92].

3.1. Propositional Temporal Logic

The Propositional Temporal Logic (PTL) we use in this paper is a linear time temporal logic taken from [MP92]. For a formalization of the syntax and semantics of PTL we refer the reader to [MP92]. Let p denote a state predicate. This means that p is constructed from state propositions. We say that the formula $\Diamond p$ holds in a state s iff p holds in s or in some future state. In addition to the standard operators of PTL as defined in [MP92] we define a *strong eventuality* operator $\diamond$ so that $\diamond p$ holds in some future state s[9]. The formula $\Box p$, which denotes a syntactic abbreviation for $\neg\Diamond\neg p$, holds in a state s iff p holds in s and in every successor state of s. Temporal Logic includes the propositional calculus. The formal semantics of PTL define 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.

3.2. Metric Temporal Logic

Real-Time extended temporal logic has been suggested in various places as a suitable tool for the specification of real-time systems (see for example [Hen91], [AL92], [Koy89] and [Ost89]). We apply a variant of these logics called *metrical temporal logic* (MTL) to the specification of QoS requirements[10]. The language of Propositional Temporal Logic (PTL) is a proper syntactic subset of MTL.

Timed Observation Sequences. The models over which we interpret PTL formulae are timed observation sequences $o = o_1, \ldots$ (see [AH92]). Each o_i corresponds to a pair s_i, I_i where s_i is a state and I_i is a numeric interval expression. Let l_i and r_i denote the left and right boundaries of the Interval I_i, then for example the timed observation $(s_i, [l_i, r_i[)$ means that state s_i can be observed in the interval starting with l_i and ending with, but not including, r_i. As we only consider instantaneous state changes the sequence of intervals I_I can be replaced by single time stamps, for example an interval I_i can be replaced by the left interval boundary l_i. We assume sequences l_i to be monotonic. We assume finite precision of our clocks, i.e. we assume that every state change coincides with a click of the clock from which we derive the timed observation. Therefore the set of natural numbers N suffices as domain for the interval expressions [AH92]. We use MTL to specify properties of concurrent systems based on an interleaving interpretation of concurrent state transitions. Assume that s_1, s_2 and s_3 are GSS of an SDL specification. Furthermore, let both $s_1, s_2, s_3, \ldots$ and $s_1, s_3, s_2, \ldots$ be admissible sequences in the untimed model. If we now want to express that both s_2 and s_3 may occur at the same time (have the same time stamp) in any order 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 $l_2 = l_3$. Hence in this interleaving model we assume the sequence l_i to be *weakly*-monotonic [AH92].

MTL language and semantics. MTL contains formulas of the form $\Diamond_I \phi$ which assert that in the current or *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. As an example the formula

$$\Box_{\leq 5}(\neg OUTPUT(A))$$

expresses the property that in all subsequent system states i in which the time stamp T_i is less than or equal to 5 the state proposition $OUTPUT(A)$ is false. In analogy to the satisfaction relation for PTL we write $o \models_{\mathrm{MTL}} p$ iff the sequence o satisfies the MTL formula p.

[9] The formal definition of the semantics of this operator is $s_i \models \diamond p$ iff $(\exists j > i)(s_j \models p)$.

[10] Our introduction will be rather informal and we will not present all possible operators, we restrict ourselves to a minimal subset of the language which we need to carry out our example. For a complete formal definition of the syntax and semantics of MTL we refer the reader to [AH92] and [Hen91, chapter 3.4]

3.3. Probabilistic Metric Temporal Logic

Related work. Some probabilistic extensions to real-time temporal logics have been suggested in the literature. The work in [HJ89] and [Han91] describes an extension of a branching time temporal logic by discrete time (integer time units) and probabilities. The modal operators are labeled by real-time values and probabilities. The logic is based on finite Markov chains. The authors do neither provide an axiomatization nor a decision procedure for their logic. However, a model checking algorithm which permits checking whether a given probabilistic real-time state transition model (a real-time Markov chain) satisfies a given formula of their logic is given. The work described in [ACD91] is close in its scope to the work of [Han91]. It also describes a model checking method for a probabilistic real-time branching time temporal logic. However, the real-time model is based on dense time (real number real-time values).

Probabilistic Metric Temporal Logic. We now give an informal definition of a a probabilistic metric temporal logic (PMTL) for the specification and for reasoning about QoS requirements. A formal model theoretic semantics of this logic can be adopted from the logics described in [HJ89], [Han91] and [ACD91].[11] The logic has the following features:

- The logic will comprise MTL as described in Section 3.2 and it is interpreted over the same class of models.
- In addition PMTL formulae are interpreted over probability labelings of an underlying state-transition model. Let T denote the set of all transitions of a given state-transition system. Then let $\pi : T \to]0;1]$ denote a probability labeling for all transitions in T. The state-transition model together with the probability labeling gives a Markov chain model.
- We interpret PMTL formulae over both the timed observation and the probability labeled state-transition model. The formula $\Diamond^{\geq a} p$ for example expresses that there is a future state such that p holds in this future state and that this state is reachable so that the probability measure is $\geq a$.

In a similar way like for PTL and MTL we write $(T, \pi, o) \models_{\mathrm{PMTL}} p$ iff the sequence o based on the set of transitions T with the probability labeling π satisfies the MTL formula p according to the formal semantics.

3.4. Complementary Specifications

Assume we have an SDL specification P and a set of formulae 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:

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

Similar conditions apply to the use of PTL and PMTL as complemenatary specifications in the context of SDL and MSC.

4. A state/transition Model for Message Sequence Charts

A finite state semantics for MSCs. In [LL94a] [LL94b] we defined a finite control state semantics for Message Flow Graphs (MFGs). Message Sequence Charts (MSC) [CCI92b] are a particular sort of MFG. For an example of an MSC with so-called conditions see Figure 1. The semantics works as follows. The graphical device MSC (left in Figure 1) is first translated into a mathematic object which in [LL94a] has been called an *ne/sig* graph (in the middle of Figure 1). The nodes in the graph represent communication events, and the edges between nodes denote either control flow between nodes (solid line arrows in the ne/sig graph) or

[11] It should be noted that the logics described in these papers rely on *branching time* models.

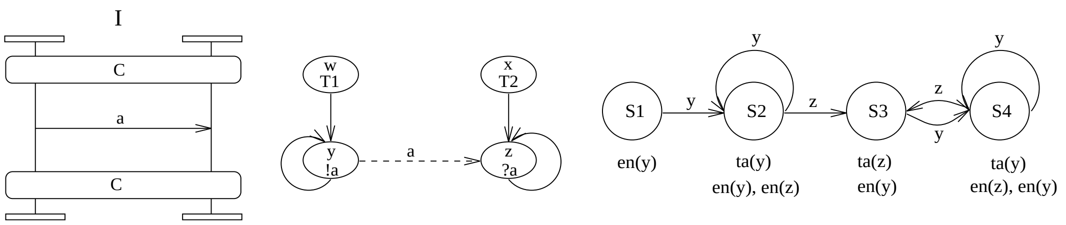

Fig. 1. MSC with corresponding MFG and Global State Transition Graph (taken from [LL94a]).

message flows (dashed line arrows[12]). A global system state (GSS) is now a collection of the control flow and message flow edges, where a control flow edge being in the GSS indicates where the control of the respective process lies, and a message flow edge in the GSS indicates a message sent but not yet received. In the example of Figure 1 the GSS S1 corresponds to the set $\{(w, y), (x, z)\}$. The semantics in [LL94a] [LL94b] defines a set of rules for enabling of events and for a GSS transition relation. In S1 the event y is enabled and when executed the system transits to state $S2 = \{(y, y), (y, a), (x, z)\}$. The result is a global state transition graph as on the right hand side of Figure 1. By defining an acceptance condition the GSTG can be augmented to an ω-automaton [Tho90], for example to a Büchi automaton.

Using PTL, MTL and PMTL for MSC specifications. In [LL94a] [LL94b] we explained how our GSTGs relate to the Manna-Pnueli Basic Transition Systems and how PTL can be used to specify liveness properties for MSCs. The state propositions we use as basic propositions for PTL formulae are the predicates $en(\tau)$ and $ta(\tau)$ where τ is a transition in the GSTG. We label transitions with the names of the communication events which cause the transition to happen, so in the GSTG in our Example the transition from $S1$ to $S2$ is labeled by y. It should be noted that for example the state proposition $ta(y)$ holds in several different states ($S2$ and $S4$), namely all those states that can be entered by executing a y event. If the transition y represents a send (receive) event of a message of type a, then we sometimes write $!a$ ($?a$) instead of $ta(y)$[13]. If we were for example to require that in our example a send event will eventually be followed by a receive event, which is a liveness requirement that prevents the system from looping on state $S2$ forever, we would write in PTL

$$\Box(!a \supset \Diamond ?a).$$

The interpretation of MTL and PMTL formulae is a straightforward extension of the interpretation of the formulae over SDL specifications.

5. Specifying QoS: Delays

Figure 2 presents the example of a simple *Sender/Receiver Service* (SRS) specified using MSCs. The example works as follows. A user `U1` of the service requests the transmission of some data by sending a `UDreq` signal to the sender process `S` which in turn requests the transmission of the data from a medium service `M` by sending a `MDreq`. The medium service is unreliable. However, in case the transmission is successful the medium service will deliver the data to the receiver process `R` by means of an `MDind` message, and the receiver delivers the data to the user process `U2`. Although the medium service is unreliable we nevertheless assume that it is capable of reliably indicating to the sender process by means of an `MDcon` signal whether the data has been delivered successfully to the receiver process, or by an `MDrej` that this is not the case. Successful delivery will be indicated to the service user `U1` by an `UDcon` signal, and unsuccessful delivery by an `UDrej` signal. The example does not reflect any particular known real example from the telecommunications, however as pointed out in [LB92] an ATM adaption layer service may offer the same functionality as the medium service

[12] In [LL94a] [LL94b] we also gave a semantics for synchronous communication.
[13] In [LL94a] [LL94b] we formally define $!a$ and $?a$ to denote the *event types* of a communication event y.

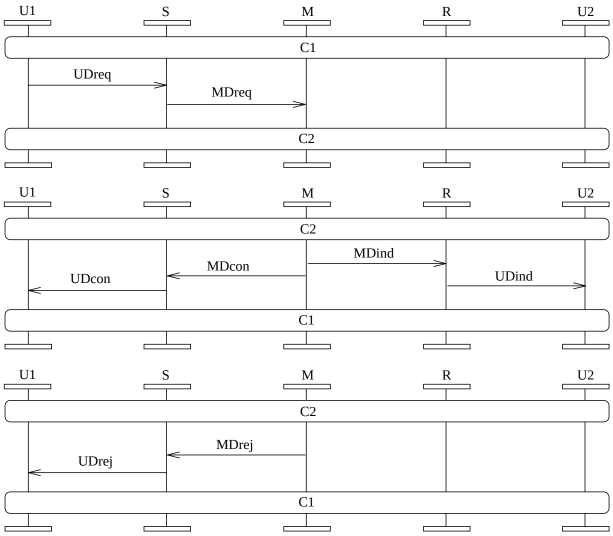

Fig. 2. MSC Specification of SRS example.

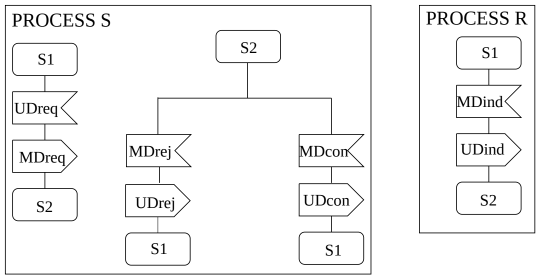

Fig. 3. SDL Specification of SRS example.

in SRS. The SDL specification presented in Figure 3 presents a similar service. However, we focus on the specification of the behaviour of the sender `S` and receiver `R` processes.

Related work. For an overview on specifying real-time constraints using MTL we refer the reader to [AH92]. In [HL92] we discussed the unsuitability of the asynchronous SDL timer mechanism for the specification of hard real-time constraints. [Don93] describes aspects of the JVTOS service using MSCs, and describes QoS measurements in TLA [Lam93]. [MS93] addresses in particular the question how timing constraints attached to MSCs can be semantically interpreted, and devises the application of a verification algorithm to real-time attributed MSCs.

5.1. Delay bounds on SRS

The appropriateness of the SDL service may depend on the satisfaction of a number of real-time constraints which we discuss here.

Service response delay bound. A reasonable liveness requirement for the SRS service is that if the service process has received a `UDreq` it will eventually indicate either an `UDcon` or a `UDrej` signal to the service user in order to indicate successful or unsuccessful delivery of data. We describe this requirement by

$$\Box(!\mathit{UDreq} \supset \Diamond(?\mathit{UDcon} \vee ?\mathit{UDrej})).$$

However, many real-time applications may require an indication to be given within a real-time bound of for example t_1 time units. In PTL this requirement reads

$$\Box(!\mathit{UDreq} \supset \Diamond_{\leq t_1}(?\mathit{UDcon} \vee ?\mathit{UDrej})).$$

Service processing delay bound. The sender process `S` may need some time to process the user data before sending out a `MDreq` to request data transmission from the medium service. The service processing delay bound requirement reads

$$\Box(?\mathit{UDreq} \supset \Diamond_{\leq t_2} !\mathit{MDreq}).$$

Message transmission delay bound at service interface. Depending on the communication mechanism used at the service interface between processes `S` and `M` the transmission of a message between both may consume time. In order to limit the transmission time to at most t_3 time units we write

$\Box(!MDreq \supset \Diamond_{\leq t_3} ?MDreq)$.

Medium transmission delay bound (Np). The process `M` is an abstraction for the whole communication subsystems which we use in order to transfer the data. One may wish to constrain the time between the events of the reception of the `MDreq` signal and the output of the `MDind` signal to t_4 time units, but only if the transmission is successful.

$\Box((?MDreq \wedge \Diamond !MDind) \supset \Diamond_{\leq t_4} !MDind)$

This requirement also constrains the medium service *not* to deliver messages to process `R` after t_4 time units after sending.

Minimal medium service response time. In a verification context it may also be interesting to state that between two events there is a minimum time that will always pass. The following formula states that if after the request the data will eventually be successfully delivered by the medium service by issuing a `MDind` signal, then this will happen within t_5 time units.

$\Box((?MDreq \wedge \Diamond !MDind) \supset \Diamond_{\geq t_5} !MDind)$

The SRS example is based on MSCs with asynchronous communication. However, with the obvious exception of the message transmission delay bound requirement all other requirements apply analogously to the same example with synchronous communications, which some may find more appropriate for the sorts of service interfaces specified here. The requirements referring to events belonging to the M process can be considered as so called *network performance* (Np) requirements.

5.2. Delay variation: Jitter

Delay jitter. Subsequent data units routed through a complex network may be subject to varying delays over time. The delay variations may be caused either by the network management changing the routes which subsequent data units are using through a multi-hop network, or by variations of the background load of the network. The ATM service is, as one example, prone to this sort of delay variation [LB92]. However, in particular 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} 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. We use the SRS example specified in SDL (see Figure 3) here to exemplify the specification of jitter constraints, similar formulae would apply to the SRS example specified by MSCs. We assume that d_{min} and d_{max} are known constant values. The requirement bounding the delay jitter for the user interface service can then be specified by the formula

$\Box(INPUT(UDreq) \supset (\Box_{\leq d_{min}} \neg OUTPUT(UDind)) \wedge (\Diamond_{\leq d_{max}} OUTPUT(UDind)))$.

5.3. Isochronicity

Isochronicity is a characteristics of many multimedia applications. The isochronicity we refer to means that events, for example sending and receiving of data units, occur periodically at equally distanced points of time. The example formulae given here again refer to the SDL specification of the SRS example.

Isochronous sending and receiving. Isochronous sending is a characteristic of a traffic source. It is characteristical for simple coding schemes for audio or video data where samples of the analogous signal are taken and sent periodically. The characterization of isochronous sending reads

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

On the receiving side the receiver may require to have subsequent data units available at isochronous moments in time. This may be expressed in a way very similar to the isochronous send characterization, namely as

$$\Box(\mathit{INPUT}(\mathit{UDind}) \supset (\Diamond_{\leq t}\neg \mathit{INPUT}(\mathit{UDind}) \wedge \Diamond_{=t}\mathit{INPUT}(\mathit{UDind}))).$$

Jitter compensation. Guaranteeing a bound on the delay jitter does not yet guarantee isochronous delivery of messages to a user, 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 is often 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 (see [LB92]). At this point we leave the SRS example as underlying model to a certain degree. We assume that the process `R` also has the functionality of a playout buffer, as described for example in [LB92][14]. The playout buffer functionality of `R` is the following. `R` accepts the possibly non-isochronous but jitter-bounded data stream from the Medium service by `MDind` signals and delays every data unit a fixed time d which is chosen so that there are usually a few data units stored in the buffer. The delivery of subsequent `MDind` signals then occurs isochronously with an inter signal delivery time of p, which ideally should correspond to the inter send event time at the sender. The jitter compensation requirement for the process `R` then reads

$$\Box(\mathit{INPUT}(\mathit{MDind}) \supset \Diamond_{\leq d}\mathit{OUTPUT}(\mathit{UDind})) \wedge \Box(\mathit{OUTPUT}(\mathit{UDind}) \supset \Diamond_{=p}\mathit{OUTPUT}(\mathit{UDind})).$$

5.4. Rates.

Rates like throughput are usually measured by the number of data units processed per time period. The temporal logic we have proposed so far does not permit the counting of events. Event counting requires a non-trivial extension of the logic[15]. However, it may be useful to specify that the time interval in between two events of some particular type, e.g. the sending of events, is restricted to a time span t. This is a reciprocal consideration compared to the number of events per time unit based rate specification. For example, in order to specify a constraint on the output rate of the medium service used in the SRS example we may require that the time between two successive *MDind* events is limited to be less than or equal to t_6.

$$\Box(\mathit{OUTPUT}(\mathit{MDind}) \supset \Diamond_{\leq t_6}\mathit{OUTPUT}(\mathit{MDind})).$$

It may be more appropriate to call this requirement the bounded *inter-send-time* requirement instead of calling it a throughput requirement.

6. Specifying QoS: Probabilistic requirements

The SRS example with retransmission: SRSR. The SDL specification presented in Figure 4 represents a variation of the SRS service of Figure 3. In this example the sender process may upon receipt of a `MDrej` signal decide non-deterministically[16] whether he attempts a retransmit (transition from state `S3` to `S2`) or whether he indicates the impossibility of successful transmission (from `S3` to `S1`). We call this example the *simple sender/receiver service with retransmission* (SRSR).

---

[14] It is a fairly easy exercise to model the functional aspects of a playout buffer in SDL.

[15] For an example of how to incorporate event counting in Temporal Logics see [Got92, Got93].

[16] Non-deterministic behaviour choices have been introduced into the SDL-92 standard [CCI92a] [Tur93] [FO92]. SDL transitions may be triggered by the `INPUT(NONE)` keyword (we call it a *non-deterministic* transition) which means that the SDL process makes a nondeterministic choice between all non-deterministic transitions plus those transition which are in a given state enabled due to a regular `INPUT` statement. The treatment in the framework of pSTS is fairly easy, a 'non-deterministic' transition has no term in the pre-condition referring to the processes input queue Q, only the control as expressed by π has to lie at the right point.

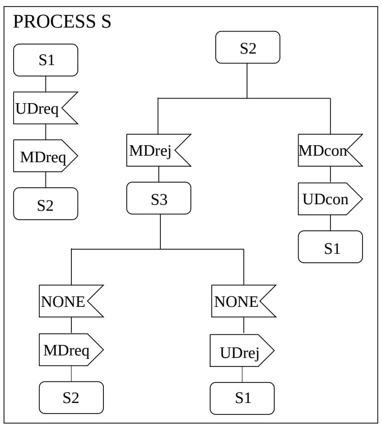

Fig. 4. SDL Specification of SRSR example.

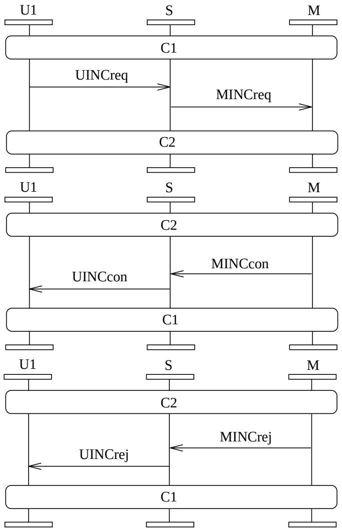

Fig. 5. MSC Specification of QoS negotiation.

Stochastic reliability. The deterministic reliability constraints are only satisfied by systems for which in every run every data unit which has once been sent will eventually be received. However, many of the applications we envisage as users of high speed communications services will tolerate a certain delay in the delivery of data units as well as a certain quantity of packets not being delivered. Non delivery may mean that either data units are lost due to an unreliable underlying service or due to network congestion, or that data units are discarded because they arrive too late to be useful in the application context. Therefore we introduce the notion of *stochastic reliability.* The requirement is that if a data unit is sent then it will with a probability a with $0 < a \leq 1$ be eventually received. The formalization of this requirement, when applied to the medium service in in the SRS or SRSR examples is

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

An interpretation of this requirement in the context of ATM is that the cell loss rate is $< 1 - a$.

Stochastic delay bound over finite intervals. We have not yet considered the aspect of time in combination with stochastic reliability. The requirement that a data unit will *possibly* be delivered in the future, is an insufficient guarantee for many applications. The much more important requirement, however, is that with a certain probability the data unit will be delivered within a certain time span. For the medium service in the SRSR example we require that if a packet has been sent it will with a probability of $\geq a$ be received within t time units.

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

In the context of ATM this may mean a limitation of the cell loss rate over a fixed period of time.

7. Network performance to QoS mapping

It is interesting to investigate the impact of the network performance (Np) on the user QoS. We consider this relationship here from a particular point of view. Assume that we are given a specification of $\mathcal{N}$ of the Np, specification $\mathcal{S}$ of the service or the protocol which we investigate, and a specification $\mathcal{Q}$ of the user QoS requirements. We now ask whether $\mathcal{N}$ together with $\mathcal{S}$ can ensure that $\mathcal{Q}$ can be satisfied, or more formally:

$$\mathcal{N} \wedge \mathcal{S} \supset \mathcal{Q}.$$

Example SRSR. We consider the Np/QoS mapping problem in the context of the SRSR example. Assume the specification to be given as an SDL specification $\mathcal{S}$ (see Figure 4). Furthermore, assume the Np requirement to be specified as a stochastic reliability requirement

$$\mathcal{N} : \Box(OUTPUT(MDreq) \supset \Diamond^{\leq 1-a} INPUT(MDrej)).$$

Finally, let the user QoS requirement be given as a delay bound requirement

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

We conjecture that the described SRSR is unable to satisfy the condition $\mathcal{N} \wedge \mathcal{S} \supset \mathcal{Q}$. This is a consequence that the underlying service is unreliable, which implies that the SRSR protocol may need to retransmit data units an unbounded number of times before the data is delivered successfully. Therefore, any time bound t in $\mathcal{Q}$ may be exceeded. To formally establish this conjecture it is necessary to either employ theorem proving or model checking techniques.

8. Specifying QoS-mechanisms

In this Section we show how the method of QoS specification which we described in the previous Sections can be applied to the specification of QoS related mechanisms, in particular QoS negotiation and reaction on QoS guarantee violation.

8.1. QoS negotiation.

Figure 5 describes a QoS negotiation scenario. Assume that this specification is somehow related to the specification of the SRS example in Figure 2[17]. The functioning of the negotiation is quite obvious. The user `U1` requests an increase in bandwidth by sending a `UINCreq` signal which the service forwards to the medium (`MINCreq`) (it is assumed that there it is processed by the appropriate network management process). The medium either grants the increase (`MINCcon`) or it refuses the increase (`MINCrej`). Both reactions are indicated accordingly to the user. The following formula limits the constraining impact of the response time requirement on the medium service by only requiring the QoS guarantee to be satisfied if the QoS level has been granted (by the medium issuing the `MINCcon` signal).

$$\Box(!MINCcon \supset \Box((?MDreq \wedge \Diamond !MDind) \supset \Diamond_{\leq t_4} !MDind)))$$

8.2. Reaction on QoS violation.

The specifications so far distinguish a systems' behaviour to belong either to the category of compliant behaviours, or not. However, it may be useful to specify a reaction on the violation of QoS requirement without requiring that the violation invalidates the behaviour against the system specification. Let us look at the SRS example again and let us assume that we *monitor* (or, if preferred to say so, *measure*) the response time behaviour of the medium service. Assume that we want the monitor to rise an `ALARM` signal at most t_8 time units after it has detected that the medium service does not respond by either `MDind` or `MDrej` within t_7 time units. We specify this as

$$\Box(\neg(OUTPUT(MDreq) \supset \Diamond_{\leq t_7}(INPUT(MDind) \vee INPUT(MDrej))) \supset \Diamond_{\leq t_8} OUTPUT(ALARM)).$$

It is obvious that the specification of the system may not contain, in conjunction, the delay bound requirement

$$\Box(OUTPUT(MDreq) \supset \Diamond_{\leq t_7}(INPUT(MDind) \vee INPUT(MDrej))),$$

as this would lead to an inconsistent specification.

[17] The example has been inspired by an example of QoS negotiation in the context of ATM based virtual private networks given in [GSH94].

9. Verification of QoS requirements

The goal of a formal verification method for QoS requirements is to prove that the specification of a system S satisfies a set of QoS requirements Q. In particular, S may be the specification of a protocol or a service and may include the guarantees provided by the underlying network (the *network performance*). Q may be the specification of QoS requirements that the System specified in S is expected to guarantee. The verification methods available are *formal verification* and *model checking*.

Formal verification. We translate the specification S into a set T_S of temporal logic formulae, unless S is already specified in temporal logic[18]. It then remains to prove that

$$T_S \supset Q.$$

Formal verification requires a proof system for the particular temporal logic calculus used. In case the underlying temporal logic is a decidable logic the proof can be fully automatized. If this is not the case manual or machine supported reasoning is necessary.

Model checking. We take the state transition model M_S of of the system S and prove formally, that this model satisfies the QoS requirements Q, formally

$$M_S \models Q.$$

Model checking amounts essentially to an exploration of the state space of the system M_S. State exploration is a well known technique in the field of protocol validation (see for example [Hol91]). Algorithms for combined real-time and probabilistic model checking based on temporal logic formulae have been described in [HJ89], [Han91] and [ACD91].

10. Conclusion

We described a method for the specification of real-time and probability constraint based QoS and Np requirements. Starting point was an analysis of SDL specifications. We mapped SDL specifications to global state transition systems and we showed, how systems states and state transitions can be described in terms of logic formulae over state propositions. Next we connected temporal logic specifications to SDL and MSC specifications. The temporal logics we used were standard propositional and metric temporal logic. The probabilistic metric temporal logic we used is a dialect of existing probabilistic real-time temporal logics.
In the second part of the paper we exemplified our method with a set of general examples for QoS and Np requirements in the context of functional specifications. Examples included delay bounds, delay jitter, isochronicity, inter-send-time bounds, stochastic reliability and stochastic delay bounds. We then showed how QoS mechanisms can be specified in the framework of our method, in particular QoS negotiation and QoS monitoring. Finally we briefly discussed methods for the formal verification of QoS/Np specifications.
Further work will address a rigorous formalization of PMTL as well as a more profound investigation of formal verification aspects. We will also investigate methods to reconcile the two worlds of state-machine and temporal logic based specifications in order to reduce the number of languages which the specifier has to learn.

Acknowledgements

The author wishes to thank Jorg Liebeherr for initial discussions on QoS as well as Peter Ladkin and Reinhard Gotzhein for helpful commentary on an earlier draft of this paper.

[18] The translation of an SDL specification into a set of temporal logic formulae is an extension of the logic based description of SDL transitions as described in Section 2.

References

[ACD91] R. Alur, C. Courcoubetis, and D. Dill. Model checking for probabilistic real-time systems. In *International Colloquium on Automata, Languages and Programming*, 1991.

[AH92] R. Alur and T. A. Henzinger. Logics and models of real-time: A survey. In J. W. de Bakker et al., editor, *Real-Time: Theory in Practice*, volume 600 of *Lecture Notes in Computer Science*, pages 45–73. Springer-Verlag, 1992.

[AL92] M. Abadi and L. Lamport. An old-fashioned recipe for real time. In J. W. de Bakker et al., editor, *Real-Time: Theory in Practice*, volume 600 of *Lecture Notes in Computer Science*, pages 1–27. Springer-Verlag, 1992.

[BHS91] F. Belina, D. Hogrefe, and A. Sarma. *SDL with Applications from Protocol Specification*. Prentice Hall International, 1991.

[BZ83] D. Brand and P. Zafiropulo. On communicating finite-state machines. *Journal of the ACM*, 30(2):323–342, Apr 1983.

[CCI92a] CCITT. Recommendation Z.100: CCITT Specification and Description Language (SDL). CCITT, Geneva, 1992.

[CCI92b] CCITT. Recommendation Z.120: Message Sequence Chart (MSC). CCITT, Geneva, 1992.

[CK93] 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.

[Don93] A.J.M. Donaldson. Specification of quality of service measurement points in JVTOS. Master's thesis, University of Stirling, Scotland, U.K., September 1993.

[Eme90] 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.

[FO92] O. Faergemand and A. Olsen. FORTE-92 tutorial on new features in SDL-92. In M. Diaz and R. Groz, editors, *Fifth International Conference on Formal Description Techniques, Participant's Proceedings: Tutorials*, 1992.

[Got92] R. Gotzhein. Temporal logic and applications - a tutorial. *Computer Networks and ISDN Systems*, 24(3):203–218, May 1992.

[Got93] 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.

[GSH94] J.-P. Gaspoz, T. Saydam, and J.-P. Hubaux. Object-oriented specification of a bandwidth management system for ATM-based virtual private networks. Unpublished manuscript, submitted for publication, March 1994.

[Han91] H. A. Hanson. *Time and Probability in Formal Design of Distributed Systems*. PhD thesis, Uppsala University, Sweden, 1991.

[Hen91] T. A. Henzinger. *The Temporal Specification and Verification of Real-Time Systems*. Report no. STAN-CS-91-1380, Stanford University, Department of Computer Science, Aug 1991.

[HJ89] H. Hansson and B. Jonsson. A framework for reasoning about time and reliability. In *Real Time Systems Symposium*, pages 102–111, 1989.

[HL92] D. Hogrefe and S. Leue. Specifying real-time requirements for communication protocols. Technical Report IAM 92-015, University of Berne, Institute for Informatics and Applied Mathematics, 1992.

[Hol91] G. J. Holzman. *Design and Validation of Computer Protocols*. Prentice-Hall International, 1991.

[ISO93] ISO/IEC JTC 1/SC21 WG1. Quality of service framework, working draft # 3. ISO/IEC, Nov. 1993.

[Koy89] Ron Koymans. *Specifying Message Passing and Time-Critical Systems with Temporal Logic*. PhD thesis, Technical University of Eindhoven, 1989.

[Kri93] A. S. Krishnakumar. Reachability and recurrence in extended finite state machines: Modular vector addition systems. In C. Courcoubetis, editor, *Computer Aided Verification, 5th International Conference, CAV'93*, volume 697 of *Lecture Notes in Computer Science*, pages 111–122. Springer Verlag, 1993.

[Kur93] J. Kurose. Open issues and challenges in providing quality of service gurantees in high-speed networks. *ACM Computer Communication Review*, pages 6–15, 1993.

[Lam93] L. Lamport. The temporal logic of actions. *ACM Transactions on Programming Languages and Systems*, 1993. To appear. Revised Version of [Lam94].

[Lam94] L. Lamport. The Temporal Logic of Actions. *ACM Transactions on Programming Languages and Systems*, 16(3):872–923, May 1994.

[LB92] J.-Y. Le Boudec. The asynchronous transfer mode: a tutorial. *Computer Network and ISDN Systems*, 24:279–309, 1992.

[LDvB94] 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.

[Liu89] M. T. Liu. Protocol engineering. In M. C. Yovitis, editor, *Advances in Computers*, volume 29, pages 79–195. Academic Press, Inc., 1989.

[LL94a] P. B. Ladkin and S. Leue. What do Message Sequence Charts Mean? In R. L. Tenney, P. D. Amer, and M. Ü. Uyar, editors, *Formal Description Techniques, VI*, IFIP Transactions C, Proceedings of the Sixth International Conference on Formal Description Techniques. North-Holland, 1994. To appear.

[LL94b] P.B. Ladkin and S. Leue. Interpreting message flow graphs. *Formal Aspects of Computing*, 1994. To appear.

[MP92] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification.* Springer-Verlag, 1992.

[MS93] N. Meng-Siew. Reasoning with timing constraints in message sequence charts. Master's thesis, University of Stirling, Scotland, U.K., August 1993.

[Ost89] J. S. Ostroff. Real-time temporal logic decision procedures. In *IEEE Real-Time systems Symposium*, pages 92–101, 1989.

[SHK92] H. Saito, T. Hasegawa, and Y. Kakuda. Protocol verification system for SDL specifications based on acyclic expansion algorithm and temporal logic. In K. R. Parker and G. A. Rose, editors, *Formal Description Techniques, IV*, IFIP Tranactions C, pages 511–526. Elsevier Science Publishers B. V. (North-Holland), 1992.

[Tho90] W. Thomas. Automata on infinite objects. In *Handbook of Theoretical Computer Science*, chapter 4, pages 132–191. Elsevier Science Publishers B. V. (North-Holland), 1990.

[Tur93] K. J. Turner, editor. *Using Formal Description Techniques.* John Wiley & Sons, 1993.

[WZ92] H. B. Weinberg and L. D. Zuck. Timed Ethernet: Real-time formal specification of Ethernet. In W. R. Cleaveland, editor, *CONCUR '92*, volume 630 of *Lecture Notes in Computer Science*, pages 370 – 385. Springer Verlag, 1992.