Specifying Real-Time Requirements for Communication Protocols[1]

Dieter Hogrefe Stefan Leue[2]

University of Berne
Institute for Informatics and Applied Mathematics
Länggassstrasse 51
CH-3012 Bern, Switzerland
hogrefe@iam.unibe.ch, leue@iam.unibe.ch

IAM 92-015

July 1992

[1] Also submitted for publication.

[2] The work of this author was supported by the Swiss National Fund.

Abstract

Real-time constraints play an important role in the specification of communication protocols. We present three practical real-time specification methods, SDL, real-time extended LOTOS and Message Sequence Charts in combination with real-time temporal logic, and exemplify their application to the specification of real-time aspects in communication protocols. We discuss their suitability for different types of real-time constraints and observe that different specification styles are suitable for different application areas.

CR Categories and Subject Descriptors: C.0 Systems specification methodology; C.2.2 Protocol Verification; C.2.4 Distributed Systems [Computer-Communication Networks] C.3 Real-time systems [Special-Purpose and Application-Based Systems]; D.2.1 Requirements/Specifications [Software Engineering]; D.2.2 Methodologies [Software Engineering]; F.3.1 Specifying and Verifying and Reasoning about Programs [Logics and Meanings of Programs]; F.4.1 Mathematical Logic
General Terms: Design, Languages
Additional Key Words: SDL, LOTOS, Message Sequence Charts, Metrical Temporal Logic, Real-time requirements; Temporal Safety and Temporal Liveness Properties

Chapter 1

Introduction

The description of communication protocols requires the description of a class of behaviour constraints which refer to real-time. The requirements we envisage consist of timeout requirements, which also occur in the description of conventional communication protocols, as well as timing requirements which require the timely delivery of some communication service.

In this paper we restrict ourselves to the consideration of timeout mechanisms. We consider the investigation of formal description techniques for other kinds of real-time requirements a task for further investigation. We discuss various practical specification techniques and their real-time specification mechanisms which we apply to a common example: the timeout mechanism of the INRES protocol connection establishment phase (see [Hog91]).

INRES connection establishment. We informally describe the considered example as follows. We consider three communicating processes, an *initiator-user*, an *initiator* and a *responder* process. The initiator-user requests the connection establishment from the initiator by issuing a `ICONreq` service promitive (SP), which causes the initiator to send a `CR` protocol data unit (PDU) to the responder to request connection establishment from the responder. The responder may respond in two different ways to that request. It may confirm the connection establishment by sending a `CC` PDU to the initiator, which may in turn signal this by a `ICONconf` SP to the user. Alternatively, the responder may reject the connection establishment by sending a `DR` PDU to the initiator which is indicated by a `IDISind` SP to the user. If connection establishment was confirmed, the responder will subsequently send a `DR` PDU to close the connection, which is indicated to the user by a `IDISind` SP. We also permit the initiator to spontaneously reject the connection establishment request to its user by issuing a `IDISind`, if it does not receive a response from the responder. At this point real-time comes into play. To avoid an undetermined waiting of the initiator to signal to its user, that the connection establishment was not

successful due to missing response from the responder, we specify a *real-time bound* for the time after which he is required to issue the `IDISind` SP.

Synopsis. Our subsequent considerations will present and discuss real-time specifications of the INRES connection establishment example. We choose three languages for the formulation of the examples: SDL, LOTOS and Message Sequence Charts (MSCs). Our choice is justified by the fact that all three techniques are widely used specification methods in industial practice, as well as in telecomminication standardisation committees. SDL posesses real-time expressiveness, in the case of LOTOS we discuss two different real-time extensions. For MSCs, which possess no real-time expressiveness, we combine this technique with real-time temporal logic. We will finally compare the associated specification styles and conclude with directions for further research.

Chapter 2

Real-time specifications

This section introduces some real-time terminology.

Real-time control systems. There is no unique definition of a real-time system in literature. Koymans (see [Koy89]) defines *time-critical systems* as systems which are required to react timely to the stimuli of the asynchronously working environment. Ostroff (see [Ost89]) talks about *real-time discrete event processes* as processes which must satisfy certain hard real-time constraints to ensure safe operation. Their correctness depends on the timely delivery of results, not only on their logical correctness. Finally, Parnas (see [FP88]) describes *hard-real-time systems* as systems which must supply information according to real-time constraints. We subsume these definitions under the term *real-time control systems* which we define to perform control functions for the environment under timing restrictions. Communication protocols are required to supply services under time constraints, we therefore consider them as real-time control systems of a particular kind.

Types of real-time constraints. There is no generally accepted classification of real-time requirements in literature, but typical cases are enumerated in different places (see for example [Koy89, chapter 6], [LV90, p. IX], [Hen91] and [FP88]). We present a subset of the types of real-time constraints which can be found in literature.

1. A *bounded invariance* condition requires that once triggered, a condition continuously holds for a certain amount of time. Conditions of this kind specify for example general program invariance properties, but their satisfaction is only required over a limited period of time.

2. A *bounded omission* condition requires that something 'bad' does not happen within a certain amount of time. This type of requirement is sometimes specified by timer mechnisms.

3. A *bounded response* (maximal distance) condition requires something 'good' to happen within a certain amount of time. A typical representative of this type is the bounded response requirement for the timely reply to requested services as it is specified in many communication protocols.

4. *Hard* real-time constraints may under no circumstances be violated without invalidating the purpose of the system. Examples are real-time control systems like flight control or medical live supporting control systems.

5. As opposed to that *soft* real-time constraints refer moreover to a level of service. They may be slightly violated without invalidating the system's purpose, and they are usually coupled to probabilistic values. Examples are audio-video transmissions, where a certain percentage of late delivery of information is tolerated, or electronic mail distribution, where timely delivery for 'most' messages is guaranteed.

We subsume bounded invariance and bounded omission under the name *temporal safety* and bounded response under the name *temporal liveness*. We would like to point out that the presented classes are not necessarily *disjoint*.

Chapter 3

Real-time in SDL

Real-time is introduced into SDL (see [XI92]) by providing a *timer* mechanism[1]. This mechanism is semantically explained as follows: A *timer* can be set in the course of a state transition by the SDL command `set`. This is accomplished by synchronously reading a *global time* value, called `now`, from a *global time process*. A *time distance* value is added to `now`, this yields the *timeout* value. We call a process which sets a timer the *timed* process. The set value is kept by a *timer process*, which is instantiated and *activated* by the timed process when setting the timer. The timer process runs independently and asynchronously from the timed process. The timer process continuously compares its timer value with the global time value. When the timer value is reached or exceeded, 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. The timed process may now consume the timer signal from its input queue and react accordingly. Timers may also be *reset* in which case the timer process becomes *deactivated* and a timer signal, which might already have reached the input queue, is removed from the input queue (this means that the pure queue startegy is violated in this case). The reset may be caused explicitly by a `reset` command or implicitly by consumption of the timer signal.

Example

In Figure 3.1 we present an SDL specification of the INRES connection establishment example, in which we make use of the timer mechanism. We consider an *initiator* and a *responder* process, the initiator plays the central role. The behaviour of the responder seems to be obvious, we therefore omit commenting its behaviour. The initiator transits from the `disconnected` state to the `wait` state upon reception of a `ICONreq` SP from the service user. In the course of this transition a `DR` PDU is sent to the responder and the timer `T` is set to the value of the global time `now` plus the time distance value `t`. It should

[1] For a very instructive presentation of the SDL timer mechanism see [BHS91], p. 168.

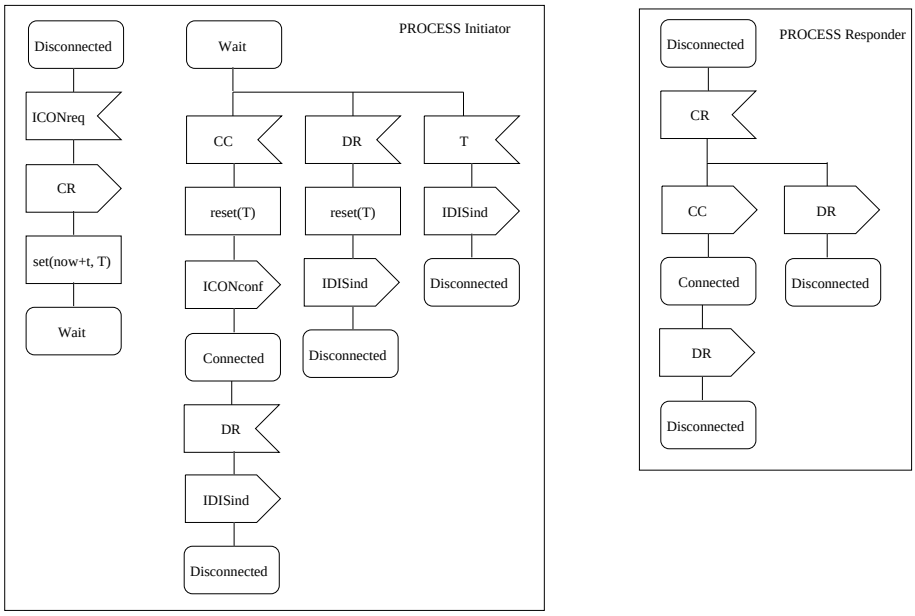

Figure 3.1: SDL specification of the INRES connection establishment

be noted that the state transition from `disconnected` to `wait` is considered to be atomic. When in `wait` state the initiator may

- receive a `CC` PDU from the responder, which indicates that the responder accepts the connection establishment, and it may then transit to state `connected`,
- or it may receive a `DR` PDU which indicates a rejection of the connection request by the responder,
- or it may receive a timer signal `T` which indicates that neither `CC` not `DR` arrived within `t` time units after the timer was set.

After the reception of `DR` or `T` signals the initiator indicates closing of the connection or unsuccessful connection establishment respectively by issuing a `IDISind` service primitive to the Initiator-user.

Discussion

We critique the following aspects concerning the timer mechanism in SDL.

- The timer mechanism allows specifying temporal safety properties. In our example we specify the safety property that the initiator does not spontaneously issue a `IDISind` SP without having received a `DR` PDU before the timer expires.

- However, processes receive the timer signals through their input queue. No assertion can be made concerning the time it takes to consume all events in the queue which arrived earlier than the timer signal.
- The interaction between the input queue and the process is asynchronous. Even if a timer signal has arrived when the input queue of the timed process was empty it cannot be guaranteed how long it will take for the timed process to actually consume the timer signal and react accordingly. However, as the SDL semantics makes a liveness assumption it is guaranteed that the signal will eventually be consumed.

The above consideration shows that it is not possible to express temporal liveness by the SDL timer mechnism. It is not possible to express that within a certain time-span an event, like the issuing of `IDISind`, is required to happen. SDL is therefore not suited for the specification of time-critical control systems.

To overcome the disadvantages arising from the asynchronous timer mechanism in SDL we suggest an *interrupt* mechanism which ensures synchronous consumption of timer signals as the timer expires. This implies an elemination of the timer consumption mechanism via a queue. Though such an extension would render the SDL semantics more complicated it seems necessary to introduce these concepts in order to specify hard real-time constraints for communication protocols[2].

[2]For a discussion of the need for synchronous interrupt mechanisms in real-time control systems see [FP88]. A synchronous clock mechanism is also used for the real-time language Conic ([Ost89]).

Chapter 4

Real-time extensions to LOTOS

Timed LOTOS

One drawback of standard LOTOS is that there is absolutely no notion of real-time. The timeout in the INRES protocol is simulated with the i event without specifying any value for the timer. This disadvantage gave rise to extensions of LOTOS with real-time. Two of these approaches shall be presented in the following [QAF90], [vHTZ90].

Assignment of time restrictions to the occurrence of events

Quemada introduces time into LOTOS by assigning timing restrictions to the occurrence of events. This approach gives precise timing to the events of a specification. Each event happens at one instance of time, which is appended to it as a numerical value. For example `a3` represents that the event shall happen at instance 3. This time value is always relative to the previous event or the initial instant of a behaviour if there is no previous event. The operator ";" is used for prefixing in time an action to a behaviour, thus `a2;b3` represents that `a` will occur 2 units of time after the initial instant and that `b` will occur 3 units of time after `a`.

The time attribute to an event may also be a set of time values. This allows for example specifying events with no precise timing but timing from a range of possible time values as will be shown in an upcoming example.

An implicit global clock exists which keeps track of the current time in the system. A behaviour generates a transition system. The following example shows graphically a transition system with a global time count `ck`.

The interleaving of timed events is not a triviality since proper time merge must be done. In LOTOS interleaving generates all possible combinations of individual behaviours, but when timing is added not all the combinations make sense. Moreover, Quemada shows that different meanings of interleaving are possible in the timed context.

Consider for example the evolution of `(a;stop|[]|b;stop)` which is shown in Fig. 4.1

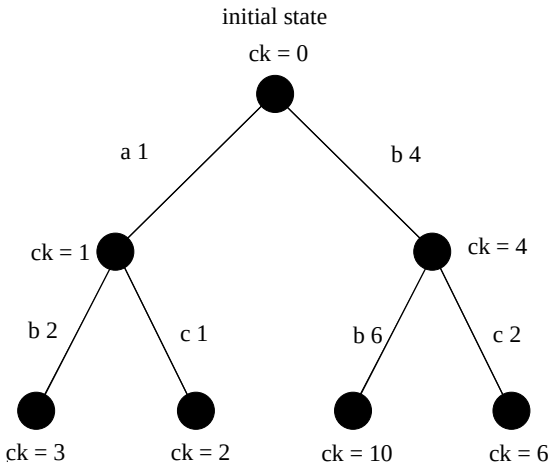

Figure 4.1: Timed transition system with time count

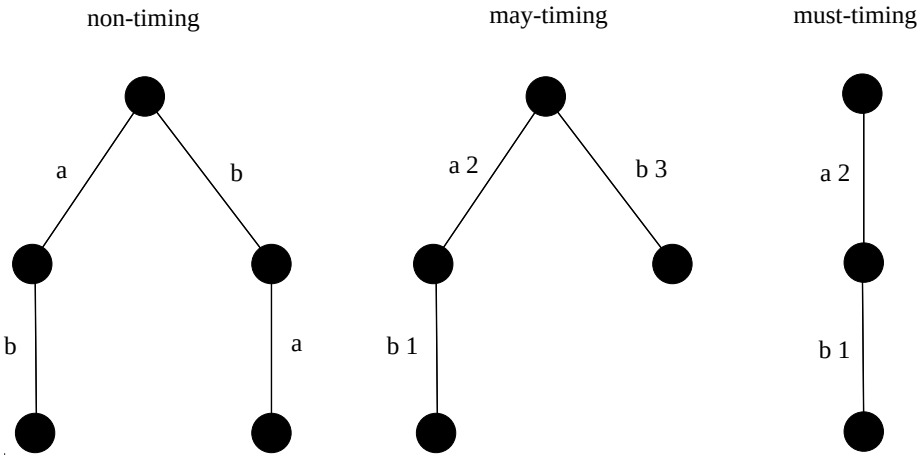

Figure 4.2: Non-timing, may timing and must timing.

for the *non-timed case*. Let us now assign timing to this behaviour in the following way: `(a2;stop|[]|b3; stop)`. This composition has the following intuitive meaning: event `a`, which must occur 2 units of time after the initial instant, interleaved with event `b`, which must occur 3 units of time after the initial instant. Only two of the evolutions given in Fig. 4.2 make sense, called *may timing* and *must timing*.

In "may timing", the first branch represents the case where `a` occurs at instant 2 and `b` at instant 1 after a has occured. In the second branch, as `a` may occur only at instant 2 if `b` has occured at instant 3, `a` will never occur.

In "must timing", the branch b3 is completely eliminated. Here a is obliged to occur before b, b will never occur if a doesn't. This interpretation of timing has better compositional properties as the composition of two timed systems evolves as one naturally would expect. The timeout in the INRES protocol can now be modelled more precisely.

```
process Example [ICONreq,IDISind,CR,DR,CC,ICONconf] :=
          ICONreq;CR;(CC;ICONconf;DR;IDISind;Example
                    []DR;IDISind;Example
                    []i 5;IDISind;Example)
endproc
```

The example contains a number actions with no timing assigned to them. This is a shorthand originally not proposed by Quemada. According to [QAF90] events which are allowed at any instant of a given time interval have to be labelled with {lower_limit ... upper_limit}. If a given event may occur at any instant in the future {0 ... no_limit} should be assigned to it. Therefore the event ICONrec should be labelled and read ICONrec {0 ... no_limit} in the example above, since it represents a waiting situation. Similarly all other events except i should be labelled in the same way because it doesn't matter at which instant exactly they occur. As a shorthand these labels are omitted in our example.

Extension by clocks

The extension presented in the previous section is particularly suitable for the definition of time distances between successive events. The specification of timing conditions on nonsuccessive events without timing intermediate events is not possible with this approach without using additional constraint processes.

An elegant way of handling this situation has been proposed by [vHTZ90]. By the introduction of clocks it becomes possible to measure durations between ordered, but not necessarily immediately successive events and to influence the occurrence of events.

In Clock Extended LOTOS or CELOTOS as it is called in [vHTZ90] two (predefined) data types are required, one for clock identifiers and one for durations. In the following example the clock C is used to measure the time between the CR event and the CC, DR or i events.

```
process Example [ICONreq,IDISind,CR,DR,CC,ICONconf] :=
          ICONreq;CR<|start(C)|>;
                    (CC[read(clock(C))<5];ICONconf;DR;IDISind;Example
                    []DR[read(clock(C))<5];IDISind;Example
                    []IDISind[read(clock(C))≥5];Example)
```

```
endproc
```

In the example the events `CC` and `DR` are only offered to the environment if the clock `C` is less than 5. The event `IDISind` is only offered if `C` is greater or equal to 5.

In CELOTOS an implicit clock table is maintained by each process. The table passes from one behaviour expression to the subsequent behaviour expression. In a similar way the clock table is handed to a process instantiation, hide expression, local definition, etc. In a choice, the clock table passes to the subsequent behaviour of the selected event.

A bit more complicated is the treatment of parallelism and disabling. Here it is important to understand that the only means of communication between independent processes is via gates. It is undesirable to change this. Therefore there should be no way that a process can influence the clock table of another process.

If the behaviour expression is a parallel expression, both processes get their own copy of the clock table. These clock tables are independent from there on. If one process restarts a clock, this influences its own clock table only. Thus no means of communication are provided between the parallel processes.

If a behaviour expression disables another expression as in `g;(B1[>B2)`, both behaviour expressions `B1` and `B2` get a copy of the clock table as it is after the event on `g`. The reason is that in LOTOS no communication is possible between `B1` and `B2`. If a clock table would be passed from `B1` to `B2`, starting a timer might be used as a means of communication.

In case of the enabling, `B1` $\gg$ `B2`, also both processes get their own copy of the clock table. But in this case values, like duration values can be passed from `B1` to `B2` and lead to clock setting in `B2`. This is possible since the exit in `B1` has the full capability of an interaction. The values defined in the exit are associated with value identifiers in the accept of `B2`.

Discussion

Although the clock the two presented real-time extensions for LOTOS does not depend on arbitrary delay caused by asynchronous signal queues like in SDL the real-time expressiveness is not greater than the expressiveness of SDL. The epressiveness is restricted to temporal safety: the `IDISind` SP is not issued earlier than 5 time units after the `CR` PDU is sent unless preceded by the reception of `CC` or `DR` PDUs. After the passage of 5 time units the issuing of `IDISind` is enabled, and the implicit LOTOS liveness ensures its eventual execution, but nothing can be asserted about how long it takes until `IDISind` will actually be issued. We conclude that temporal liveness is not expressible in the presented variants of LOTOS.

Chapter 5

Real-time properties for MSC specifications

Message Sequence Charts (MSCs) are widely used in industry as a specification method for systems of asynchronlusly communicating processes. MSCs focus on the description of the message and control flow of the considered processes, data and real-time aspects are not directly specified. In previous work ([GHL+92], [LL92b]) we formalised this method by providing an automaton semantics. We also showed how additional safety and liveness properties for the specified systems could be formulated by the application of temporal logic (see [LL92b]). We will now present a specification of the considered INRES example and we will thereafter use real-time extended temporal logic to specifiy the required real-time behaviour. This means that we do not intend to present a real-time extension to MSCs but that we suggest the combination of well known specification techniques to obtain the desired result, the specification of real-time constraints for a system specified by MSCs.

Example

In figure 5.1 we present a specification of our INRES connection establishment example by means of MSCs. The specification is divided into four MSCs. MSC1 describes the connection request by the initiator-user and the emission of a `CR` PDU to the responder. MSC1 ends with a label, namely `C2`, which indicates possible continuations of the control flow of the involved procecsses. We call those labels *conditions*. MSCs may be continued at conditions with the same name. We observe that there are three possible continuations from this point on:

- MSC2 describes a successful connection establishment and a regular disconnect afterwards. A continuation is possible at condition `C1` of MSC1.

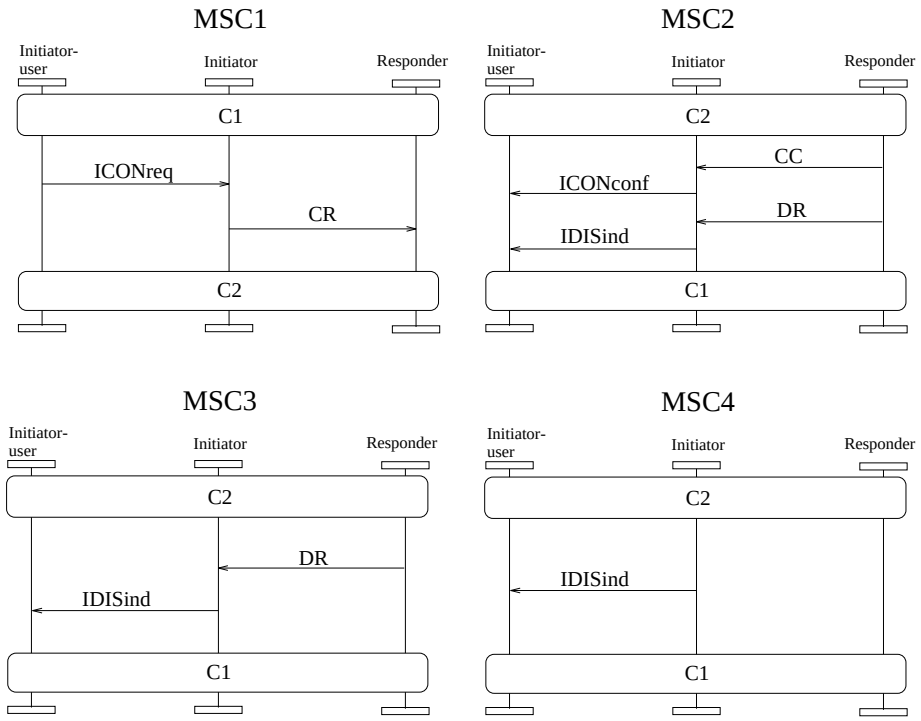

Figure 5.1: MSC specification of INRES connection establishment

- MSC3 describes the situation in which the responder rejects the connection establishment, a continuation is possible at condition `C1` of MSC1.
- Finally MSC4 describes the situation in which the initiator does not receive an answer upon his connection request and therefore indicates the unsuccessful connection establishment by a `IDISind` service primitive.

Real-time constraints

As already mentioned we previously introduced an operational model as semantics for MSC specifications. In this model we translate MSCs to global state automata which generate all possible interleavings of the communication events specified by an MSC specification. In [LL92a] we showed how temporal logic is well suited to specify liveness properties for MSC specifications. We will now suggest a temporal logic based method to specify real-time properties for MSCs. We will first informally introduce some terminology which is in much more detail explained in [LL92a].

Computations and state predicates The global state automaton $\mathcal{S}$ which we obtain from the MSC specification transits between global system states[1]. Its transitions are

[1] Manna and Pnueli use the term *Basic Transition System* in [MP91] to describe these systems.

labeled by the systems *communication events*, for a message of type p we denote the corresponding *send* event by $!p$ and the *receive* event by $?p$. Communication in MSCs is *asynchronous*, therefore sending and receiving are distinct events. We interpret the system's behaviour over infinite state sequences $\sigma = s_0, s_1, \ldots$. We refer to s_i as the *i-th position* of σ. A state sequence σ is called a *computation* of a considered System $\mathcal{S}$, iff s_0 is an initial state of $\mathcal{S}$ and every consecutive pair s_i, s_{i+1} corresponds to a state transition in $\mathcal{S}$. We say that a transition a is taken at position i of some computation if the corresponding system $\mathcal{S}$ transited from s_{i-1} to s_i by a. We define the state predicate $ta(a)$ to be true at position i if a is taken at that position.

Propositional Temporal Logic. We define a Propositional Temporal Logic (PTL) in the usual way, following [MP90]. The language has the state predicate *ta(a)* as its only basic assertion, includes the Boolean connectives (we use just $\neg$ and $\vee$ for simplicity), and the temporal operators $\Diamond$ (eventually), $\Box$ (henceforth) and $\mathcal{S}$ (since). As syntactic abbreviations we introduce the notations $\Box p \stackrel{\triangle}{=} \neg\Diamond\neg p$ and $\Diamond\!\!\!-p \stackrel{\triangle}{=} true\ \mathcal{S}\ p$. The semantics are as usual, for a definition we refer the reader to [MP90].

Real-Time 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], [Koy89] and [Ost89]). We apply a variant of these logics called *metrical temporal logic* (MTL) to our example[2]. PTL is a proper subset of MTL. 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. It is straightforward to define a satisfaction relation for timed computations and MTL formulas. A computation which does not satisfiy a set of MTL formulas given as specification is not a computation of the specified system. If such a computation is the trace of some implemented system, then the implementation is not conform to the specification.

Real-time constraint for INRES connection establishment. We restrict the set of possible computations for the INRES system as it is specified in example 5.1 in the following way. Any behaviour of the system as it is specified by the MSCs is acceptable. However, considering the behaviour of the initiator we want to require the following real-time behaviour. We anchor the real-time requirements at the point where the initiator sends the `CR` PDU. At first we require that if within t time units after sending `CR` the

[2]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 [Hen91, chapter 3.4]

initiator will issue a `IDISind` SP only if preceded by a the reception of a `CC` or a `DR` PDU.

$$\Box(ta(?ICONreq) \supset \Box_{\leq t}(ta(!IDISind) \supset \Diamond\!\!\!-(ta(?CC) \vee ta(?DR))))$$

We call this requirement a *temporal safety* requirement. Furthermore we require, that if the initiator has not received a `CC` or a `DR` PDU within t time units it will eventually issue a `IDISind` PDU to indicate unsuccessful connection establishment.

$$\Box(ta(?ICONreq) \supset ((\Box_{\leq t}\neg(ta(?CC) \vee ta(?DR)) \supset \Diamond(ta(!IDISind))))$$

The above requirement can be classified as temporal safety combined with a general liveness assumption for the occurrence of `!IDISind`.

Discussion

The above specification describes the system beahviour in pretty much the same way like the SDL and the LOTOS specifications. It is specified that when the point of time t has not yet been reached, the initiator does not issue a `IDISind` SP without being explicitly requested to do so by receiving a `DR` PDU. Once t time units have expired, we require the eventual issuing of a `ISIDind` SP without specifying how long it will take untill it is issued.

As opposed to SDL and LOTOS the expressiveness of real-time temporal logic is not restricted to temporal safety requirements. In addition to the above specification we may also require, that the `IDISind` SP is issued after $t' > t$ time units if no `CC` or `DR` PDU has been received beforehand.

$$\Box(ta(?ICONreq) \supset ((\Box_{\leq t}\neg(ta(?CC) \vee ta(?DR)) \supset \Diamond_{<t'}(ta(!IDISind))))$$

The above requirement can be classified as a temporal liveness requirement.

Chapter 6

Discussion and concluding remarks

We presented real-time specifications of the INRES protocol connection establishment phase using three different practical specification methods: SDL, real-time extended LOTOS and MSCs in combination with real-time temporal logic.

We investigated the real-time expressiveness of the different specification methods and found out that both real-time extensions of LOTOS and the timer mechnism of SDL are equally expressive. Both allow expressing temporal safety properties but not temporal liveness. We then considered MSCs which do not posses real-time expressiveness. We provided that expressiveness by using real-time temporal logic in combination with the MSC specification. It was then possible to express temporal liveness by means of real-time temporal logic formulas. The combination of real-time temporal logic on the one hand side and SDL and LOTOS on the other hand side in order to express temporal liveness is a topic for further research.

We also observe that there are two specification styles associated with the specification techniques used.

- SDL and real-time extended LOTOS are *mechanism* oriented. They specify real-time behaviour by describing the mechanisms which guarantee timely behaviour. In SDL the timeout mechanism is implicitly anchored in the semantics and explicitly specified in the syntax. The variants of LOTOS we considered use global clocks or process depending clocks in the semantics, and explicit timing restrictions for events or clock access operations in the syntax.

- As opposed to the mechanism oriented specification style we consider the MSC specifications in combination with the real-time temporal logic to represent a *requirement* oriented specification style. We showed that operational semantic models can easily be obtained from MSC specifications. Based on these temporal logic based real-time

requirements can be formulated.

We claim that the operational specification style is more geared towards providing support for the implementation phase of the communication protocol engineering process, as the semantics of these specification techniques explain how real-time systems could be implemented. The requirement oriented approach on the other hand is better suited to enhance verification, validation and testing of communication protocols[1]. Model theoretic semantics for real-time temporal logics allow describing a system by the set of traces it generates. A system generating a trace which is not in this set is therefore not a conforming implementation of the specification.

Further work will also lead us to a more detailed analysis of real-time requirements typical for high performance communication protocols. This analysis will include the investigation of the relation of real-time requirements and other non-functional requirements like quality of service or throughput requirements. On these grounds we will devise one or more suitable specification styles for the specification of high performance protocols.

[1] For verification and validation see for example [Hen91], for testing see for example [RG89].

Bibliography

[BHS91] Ferenc Belina, Dieter Hogrefe, and Amardeo Sarma. *SDL with Applications from Protocol Specification*. Prentice Hall International, 1991.

[FP88] S. R. Faulk and D. L. Parnas. On synchronisation in hard-real-time systems. *Communications of the ACM*, 31(3):274–287, mar 1988.

[GHL+92] J. Grabowski, D. Hogrefe, P. Ladkin, S. Leue, and R. Nahm. Conformance testing - a tool for the generation of test cases. Project Report, project contract no. 233, funded by Swiss PTT, University of Berne, may 1992.

[Hen91] Thomas 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.

[Hog91] Dieter Hogrefe. OSI formal specification case study: The INRES protocol and service. Technical Report IAM-91-012, University of Berne, 1991.

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

[LL92a] P. B. Ladkin and S. Leue. An analysis of message sequence charts. Technical Report IAM 92-013, University of Berne, Institute for Informatics and Applied Mathematics, 1992. Also submitted for publication.

[LL92b] P. B. Ladkin and S. Leue. An automaton interpretation of message sequence charts. Technical Report IAM 92-012, University of Berne, Institute for Informatics and Applied Mathematics, 1992. Also submitted for publication.

[LV90] P. B. Ladkin and F. H. Vogt. Proceedings of the Berkeley workshop on temporal and real-time specification. Technical Report TR-90-060, International Computer Science Institute, Berkeley, California, 1990.

[MP90] Z. Manna and A. Pnueli. A hierarchy of temporal properties. In *Proceedings of the 9th Annual ACM Symposium on Principles of Distributed Computing*, pages 377–408. ACM Press, aug 1990.

[MP91] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems*, volume 1, Specification. Springer-Verlag, 1991.

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

[QAF90] J. Quemada, A. Azcorra, and D. Frutos. TIC: A timed calculous for LOTOS. In S. T. Vuong, editor, *IFIP Formal Description Techniques, II*, pages 195–209. Elsevier Science Publishers B. V. (North Holland), 1990.

[RG89] Rami R. Razouk and Michael M. Gorlick. A real-time interval logic for reasoning about executions of real-time programs. In *Proceedings of the ACM SIGSOFT'89 Third Symposium on Software Testing, Analysis and Verification*, pages 10–19, dec 1989.

[vHTZ90] Wilfried H. P. van Hulzen, Paul A. J. Tilanus, and Han Zuidweg. Lotos extended with clocks. In S. T. Vuong, editor, *Formal Description Techniques, II*, pages 179–191. Elsevier Science Publishers B. V. (North-Holland), 1990.

[XI92] CCITT SG XI. *Recommendation Z.100: Specification and Description Technique*. CCITT, 1992. Geneva.