Timed MSCs - an Extension to MSC'96

Ina Schieferdecker[1], Axel Rennoch[1], Olaf Mertens[2]

[1] GMD Fokus,
Hardenbergplatz 2, 10623 Berlin
e-mail: {Schieferdecker, Rennoch}@fokus.gmd.de

[2] Technische Universität Berlin,
Einsteinufer 25, 10587 Berlin
olafbahg@zrz.linux.tu-berlin.de

Abstract

This paper presents a new approach for extending Message Sequence Charts (MSCs) with time concepts. The approach is mainly aimed at using MSCs in the context of performance testing. The time extensions offer means to formally specify test purposes for performance testing as well as to describe time extended execution traces of distributed systems tests. Although already a number of time extensions has been proposed for MSCs, these are mainly aimed at describing timing requirements and not time measurements, which are needed for performance testing.

1. Motivation

MSCs are widely used to describe functional requirements and/or functional behavior of distributed systems as the temporal ordering of asynchronous message exchange between process instances of a system, that may also communicate with the environment. MSCs were mainly developed by ITU-T for the design of telecommunication systems (in conjunction with SDL specifications), but they are used for the description of distributed systems in general. MSCs are used for abstract, first shot service descriptions, for functional requirements specifications, as a basis to steer the execution of a functional simulation, for describing test purposes in conformance testing, for the description of interfaces of systems components as well as for the specification of use cases in object-oriented design.

MSCs were standardized in 1992 for the first time, what lead to a fast growing number of MSC users and researchers. The upcoming standard for MSCs - MSC'96 - includes operators for the behavioral combination of MSCs and high-level MSCs for an hierarchical description of MSCs, but still has no means to describe real-time systems. MSCs contain only a concept of timers to describe the functionality of a two case behavior after setting a timer - either the timeout occurs or the timer is reset. The timeout period has no quantitative meaning.

However, in today's distributed systems not only the functional correctness but also their timeliness and performance are of importance. Examples are Quality-of-Service requirements in multimedia communications or requirements for real-time systems.

Therefore, specification techniques should address timing issues as well. So far, timing issues were considered mainly in behavioral specification techniques based on finite state machines or process algebras as well as in requirements specifications based on temporal logics. Investigations on how to extend MSC with time concepts have been started only recently.

2. Time Concepts for Message Sequence Charts

Basic objects in an MSC are processes, messages, actions, conditions, and timers. Concepts for the functional composition of these objects include coregions, generalized ordering, process creation and termination, and inline expressions. Other concepts, such as MSC composition and decomposition, MSC references, and high-level MSCs, are mainly used for structuring purposes.

MSCs [4] offer with the timer concept only restricted means for the description of time dependency. In fact, the semantics of MSCs has no time concept, so that the timeout periods of timers only serve as comments (annotations) with no semantic meaning.

This paper aims at defining a time extension for MSCs that can be used in the context of performance testing, i. e. timed MSCs are aimed at the description of test purposes for performance tests. Hence, we have to be able to describe timing constraints (for example, to describe test purposes for maximal delay tests between request and response in order to impose timed requirements) and time measurements (for example, to describe delay measurements between request and response to impose timed monitoring).

Before presenting our approach of timed MSCs, let us consider the related work of introducing time into MSCs [1], [3], and [14]. [1] considers a subset of basic MSCs (with message exchange only). Timing constraints are expressed by means of time-quantified timers and timing delay intervals. Timer periods can be set to a positive integer valued periods. Timing delay intervals can be of the form [a,b], [a,b), or (a,b] with $a \in N$ and $b \in N \cup \{\infty\}$. Timing delay intervals define minimal and maximal delays for a message flow or for a control flow in a process. The following events can be used to delimit the control flow of a process: start of a process, end of a process, output and input of a message, and setting, resetting or timeout of a timer. The semantics is defined by means of timed traces that are consistent with the (visual) partial order of events in the MSC and with the timing constraints of timers and timing delay intervals. A specific attention in [1] is given to the timing analysis of such MSCs: it investigates the timing consistency and process divergences in timed MSCs specifications. The approach in [1] is a base for the considerations in this paper, although we have to extend the time concepts such as the use of timing intervals as well as to introduce timed measurements.

MSC/RT has been defined in [14] for the specification and verification of real-time requirements for real-time systems used in avionics, cars or in communication systems. It concentrates on basic MSCs and defines timing requirements for events (output and input of a message) and for actions. A timing requirement consists of a relation ($<, \leq, =, \geq, >$), a time (a rational number), and possibly of a tolerance (to express some uncertainty). We will re-use the idea of timing requirements for actions, which describe actions that may last for a certain period of time.

[3] proposes PMSCs as an extension to MSC'96 for the specification of various performance aspects including performance and resource requirements and the specification of available resources. A subset of MSC'96[1] is used. The extensions are enclosed as special annotations in MSC comments in order to make PMSC downward compatible. Since the time extensions are of interest only for a timed MSC approach, let us concentrate on that. PMSCs consider all events such as actions and message output and input to be non-instantaneous and time-consuming. Timed intervals between the beginning and/or the end of events can be defined in a manner that is similar to [14] (except that the tolerance has been omitted). We will not follow the approach of considering all events to be non-instantaneous, because that would complicate the semantics definition[2].

This paper follows also the approach of introducing quantitative timing into a specification technique by connecting time-quantified information to specific instantaneous events in the specification. Events of an MSC that might be enhanced with time requirements are

- the start, creation and termination of a process,
- the output and input of a message,
- the loss or generation of a message,
- the start and end of an action,
- a condition, and
- the set, reset, and timeout of a timer.

We consider only two "activities" in an MSC to be non-instantaneous: that is the "activity" of sending and receiving a message and the "activity" of executing an action. That decision was mainly inspired by the fact that it is sufficient to have one concept for the description of time-consumption in the control flow of a process and in the communication flow between processes.

As discussed above, time extensions for MSCs differ in the possibilities to define time periods between MSC events. The most general approach would support the assignment of time periods to any of the events as well as to any mixture between them. In a first attempt, we decided to restrict to time assignments to:

- the output and input of a message,
- the start and end of an action,
- a condition, and
- the set, reset, and timeout of a timer.

In addition, we consider the start, creation and termination of a process to be urgent and to have no delay. The synchronization on conditions is considered to be urgent in the sense, that any process that is involved in a condition is able to wait as long as needed, but once the condition is met by all involved processes it occurs

1. The subset consists of so called plain MSCs (which are basic MSC with instance decomposition, but without MSC references and conditions) and of reference expressions (to express sequences, loops, parallel composition, alternatives, and optionals of plain MSCs).
2. In fact, [3] skips a semantics for PMSCs.

immediately without any delay. An action can be defined to be non-instantaneous. Finally, a process can be enforced to idle for a given time period.

The approach allows us to model following characteristics of real-time systems:

- time consuming procedures,
- idle periods,
- quantified timeouts,
- timed message delivery, and
- hard and soft deadlines.

Our approach goes beyond the previously discussed related works in the sense that we do not restrict to a subset of MSCs. In addition, we do not only offer features for the specification of timing constraints but also for the specification of timed measurements. However, that is induced by our specific needs to apply timed MSCs in the context of performance testing.

3. The Timed MSC Approach

TMSCs uses a predefined time domain T (either a discrete or dense one). In addition, a time unit of measurement such as s (for seconds), ms (for milliseconds), etc. can be defined. All time expressions refer to this time unit. However, the time unit has no impact on the semantics of a TMSC (in terms of the timed traces of the events), but serves as a scaling factor and a hint to system developers. The default assumptions for a TMSC are a discrete time domain with seconds as the unit of measurement.

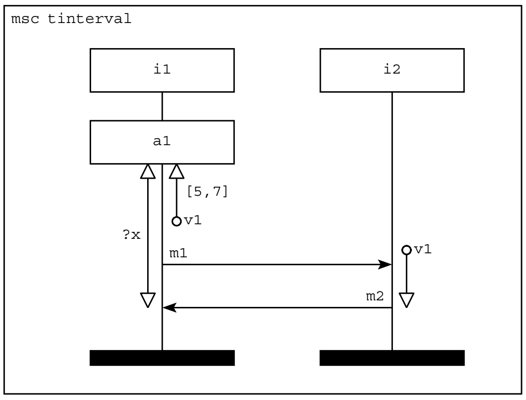
3.1. Timed Intervals

Timed intervals are used either to require a certain delay between events or to measure the delay. We denote this as the requirement and measurement mode, respectively.

Timed intervals contain the lower and/or the upper bound and may be open or closed, i.e. [a,b], [a,b), (a,b], or (a,b) with $a \in T$, $b \in T \cup \{\infty\}$, and $a \leq b$ being a closed, left-closed, right-closed, and open interval, respectively.

If the minimal delay is not given it equals per default 0. If the maximal delay is not given, it equals per default ∝. For example, `(,5)` means `(0,5)` and `[2,)` means `[2,∝)`. In addition, all events, where no time interval is given, have an undetermined delay in between `[0, ∝)`.

Time measurements are defined by the use of a time variable, where this variable is implicitly declared: `?var` means, that the delay between the identified events should be measured and the measured valued should be assigned to `var`.



```
msc tinterval; inst i1, i2;
    instance i1;
        action a1 begininterval v2 ?x
                  begininterval v1 [5,7];
        out m1 to i2;
        in m2 from i2 endinterval v2;
    endinstance;
    instance i2;
        in m1 from i1;
        out m2 to i1 endinterval v1;
    endinstance;
endmsc;
```

Fig. 1 Timed Intervals

The graphical representation of an interval uses a vertical line with two arrows that indicate the starting and the finishing event for this interval. In the case that the events belong to different processes or to different MSCs (when used in conjunction with HMSCs), the time interval can be divided into two parts, which are logically connected via a label.

Fig. 1 contains two timed intervals. The left is in measurement mode. Variable x is assigned the delay between action a1 and input of message m2. The right timed interval is in requirement mode and is split between instance i1 and i2. It requires that the delay between the end of action a1 and the output of message m2 is in between 5 and 7.

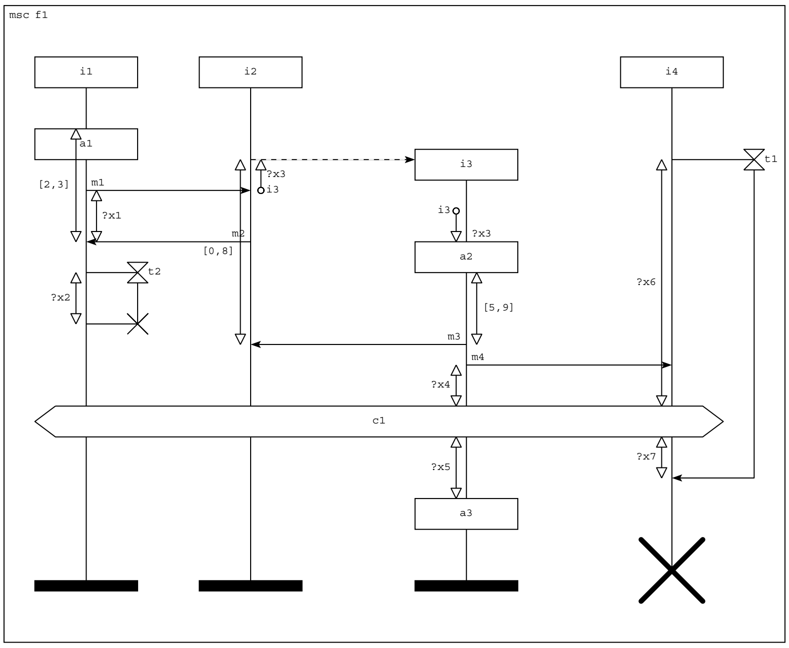


Fig. 2 Usage of Timed Intervals

The textual representation of a timed interval consists of the keyword begininterval, which denotes the instantaneous event that starts the interval, and of the keyword endinterval of the instantaneous event, which ends the interval. Each timed interval has a unique identifier. The identifier is followed either by the interval bounds or the measurement variable. The textual representations for the MSC is also given in Fig. 1.

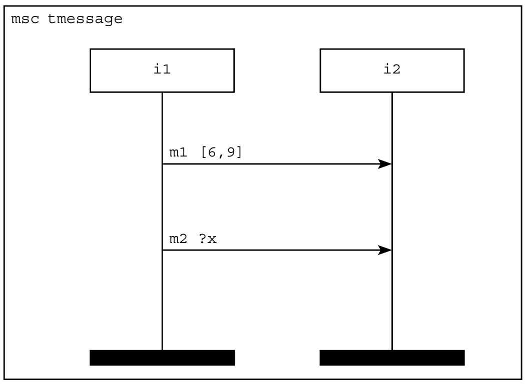
A number of different possibilities to use timed intervals in a flexible manner is shown in Fig. 2. It contains timed intervals between

- start of an action and input of a message (instance i1, timed interval [2,3]),
- output and input of messages (instance i1, timed interval ?x1),
- start and reset of a timer (instance i1, timed interval ?x2),
- creation of a process and input of a message (instance i2, timed interval [0,8]),
- creation of a process and start of an action (instance i2 and i3, timed interval ?x3),
- end of an action and output of a message (instance i3, timed interval [5,9]),
- output of a message and condition (instance i3, timed interval ?x4),
- condition and start of an action (instance i3, timed interval ?x5),
- setting a timer and condition (instance i4, timed interval ?x6), and
- condition and timeout (instance i4, timed interval ?x7).

3.2. Timed Messages

Timed messages are an extension of messages to describe either a delay between output and input of that message or to declare a time measurement for it. Similar to timed intervals, a requirement and a measurement mode exist. A timed message in requirement mode is followed by an interval containing the lower and upper bound for the delay. For example in Fig. 3, m1[6,9] means that the message is delivered at most after 6 and at least until 9. In measurement mode, a timed message is followed by ?var. With m2 ?x in Fig. 3, the delay of message delivery m2 is assigned to the time variable x.

A message without any time extension is considered to be in requirement mode with undetermined delay, i.e. with interval [0, ∝).



```
msc tmessage; inst i1, i2;
    instance i1;
        out m1 [6,9] to i2;
        out m2 ?x to i1;
    endinstance;
    instance i2;
        in m1 [6,9] from i1;
        in m2 ?x from i2;
    endinstance;
endmsc;
```

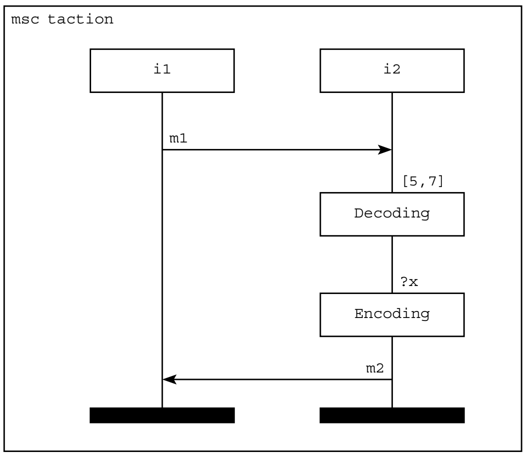
Fig. 3 Timed Messages

3.3. Timed Actions

Actions as the representation of internal computations of a process are considered to be non-instantaneous. Analogous to timed intervals or timed messages, a timed action is either in requirement or in measurement mode. In requirement mode, the interval defines the lower and upper bounds for the duration of an action.

For example, the timed action in Fig. 4 has a computation period in between `5` and `7`. In measurement mode, a time variable is defined that gets assigned the duration of that action.

An action without time extension is considered to be in requirement mode with interval `[0, ∝)`.

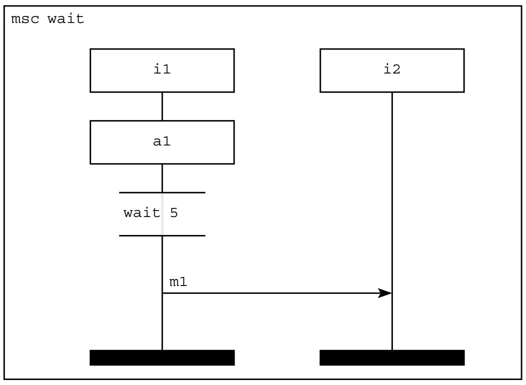


```
msc taction; inst i1, i2;
    instance i1;
        out m1 to i2;
        in m2 from i2;
    endinstance;
    instance i2;
        in m1 from i1;
     action ‘Berechnung’ [5,7]
        out m2 to i1;
    endinstance;
endmsc;
```

Fig. 4 Timed Actions

3.4. Timed Wait

The timed wait concept is used to describe idling of a process for a given time. The idle period can be defined exactly or may have a lower and upper bound. For example the timed wait in Fig. 5 defines an idle period of `5` (what is a shorthand notation for `[5,5]`).



```
msc twait; inst i1, i2;
    instance i1;
        action 'a1';
        wait 5;
        out m1 to i2;
    endinstance;
    instance i2;
        in m1 from i1;
    endinstance;
endmsc;
```

Fig. 5 Timed Wait

3.5. An Example Timed MSC Specification

The alternating bit protocol serves as an example for the use of Timed MSCs. It is a well-known, small-sized protocol what makes it specifically useful for research papers.

The alternating bit protocol is a uni-directional communication protocol, which transfers data from the sender to the receiver. It assumes an unreliable underlying transmission service, where data may be lost and delayed, but data are not misinserted, duplicated, or corrupted. Each received data is explicitly acknowledged by the receiver. A data packet is re-sent if the timer expires (no acknowledgment has been received). Data packets and acknowledgments are explicitly numbered with `0` or `1`, what allows the sender to identify the appropriate acknowledgment for a sent packet. The sender has a sending window of size `1`. Only if the sent data is acknowledged by the receiver correctly, the next data packet is sent. The sender uses the variable `sn` that contains the bit of the last sent packet. The variable is alternated after each successful data transmission. The receiver uses variable `en` to carry the bit of the next awaited packet. This variable is alternated after successful packet reception and delivery.

This paper presents two versions of describing the alternating bit protocol with Timed MSCs. The first version in Fig. 7 in the Appendix uses three processes: `Sender`, `Netzwerk` and `Empfaenger`. The high-level MSC defines the causal relationship between the MSCs. The processes are initialized with `sn= 0` and `en= 1`. The variable `n` contains the number of the actual packet `data` or of the actual acknowledgment `ack`. The timed interval `i1` defines the maximal delay between sending a packet and receiving the appropriate acknowledgment to be `5`. The delay between sending a packet by the `Sender` and receiving it by `Empfaenger` is measured in the variable `x` by means of the timed interval `i2`.

Another version of the alternating bit protocol is given in Fig. 6 in the Appendix. In contrast to the first version, the underlying transmission service is not explicitly represented. Instead, it uses inline expressions to define alternative behavior and lost message to represent data losses during transmission. The measurement between sending and receiving a packet is now represented by a timed message `data`.

4. The Semantics of Timed MSCs

This section discusses an on-going work on the semantics definition of TMSCs. It presents the general ideas only. The semantics of TMSCs is defined on the basis of their textual representation and on the work presented in [6] and [8][1].

The time concepts of TMSC are defined by E-LOTOS [5] expressions, which have a well-defined semantics.

1. Care has to be taken when using the textual representation instead of the graphical one, what was pointed out in [7]. The current mapping from the graphical to the textual representation as defined by [4] leads to anomalies where MSCs with different graphical representations (and different semantics) have the them textual representation. That occurs in particular when messages are crossed. Our solution to this problem is to reflect the relationship between output and input of a message (as it is done in the graphical representation by the message arrow) explicitly in the textual representation. That can be achieved via a unique identification of each message output. For lack of space, the details are not presented here.

E-LOTOS offers four constructs to express real-time capabilities of behaviors

- a time type (discrete or dense) with addition and comparisons on times,
- a wait operator to introduce delays,
- an extended communication operator that is sensitive to delay, and
- urgency for behaviors that cannot delay. Urgency is expressed by means of internal actions, exception raising, and termination actions.

Please consider the simple example below that is taken from [5]:

```
loop forever in
    loop forever in tick endloop [> wait(1);
    loop forever in tock endloop [> wait(1)
endloop
```

defines a behavior that performs any number of `tick` during the first time interval, at time `1` control is handed over, and any number of `tock` is performed until time `2`.

	TMSC	**Semantics in E-LOTOS**
Timed Intervals	`[a,b]` between event `e1` and event `e2`	`e1; e2 @?t [(t ge a) andalso (t le b)]`
	`(a,b]` between event `e1` and event `e2`	`e1; e2 @?t [(t gt a) andalso (t le b)]`
	`[a,b)` between event `e1` and event `e2`	`e1; e2 @?t [(t ge a) andalso (t lt b)]`
	`(a,b)` between event `e1` and event `e2`	`e1; e2 @?t [(t gt a) andalso (t lt b)]`
	`?x` between event `e1` and event `e2`	`e1; e2 @?t`
Timed Messages	`m[a,b]`[a]	`out !m; in !m @?t [(t ge a) andalso (t le b)]`
Timed Actions	`g[a,b]`[a]	`start !g; end !g @?t [(t ge a) andalso (t le b)]`
Timed Wait	`wait([a,b])`[a]	`var t: Time in ?t = any Time [(t ge a) andalso (t le b)]; wait (t);`
Timers	`set T(a)`	`set !T`[b]
	`reset T`	`reset !T`
	`timeout T`	`set !T; ( timeout !T @!timeout(T) [set_again !T> reset !T; set !T; raise(set_again !T) )`

Table 1 E-LOTOS Timing Constraints for the Time Concepts of TMSCs

a. The other intervals are defined in the same manner as the intervals for timed messages
b. In addition, the timeout value `a` of `T` is stored in a record and can be accessed via the function `timeout(T)`.

The semantics of TMSCs is defined by a constraint-oriented E-LOTOS specification, that contains separate behavior expressions for the

- temporal order of the events in a process,
- the output before input ordering of messages, and
- the timing constraints for each timed construct of the TMSC[1].

The behavior expressions for expressing the temporal order of events and the output before input ordering adopts the approach of [8] with corrections that are due to [7]. The behavior expressions are parallel composed and synchronize in common events to meet the additional constraints. The timing constraints for the time concepts of TMSCs are given in Table 1.

The time constraints use the `@P` annotation, which matches the pattern `P` to the time at which the event happens. The time is measured relative to the time when the event was enabled. Output and input of a message are represented by separate events `out` and `in` respectively, that are parametrized with the unique identifier of the message and possibly with message parameters. A timed action is represented by a pair of start and end event, since it is non-instantaneous. Similar to timed messages, these events carry the unique identifier of the action. A timed wait is represented by the E-LOTOS wait construct. It uses a time variable that is set to a time value from the corresponding interval. The timer semantics is defined by means of `set`, `reset`, and `timeout` events. It uses the suspend/resume operator that allows the resume of an interrupted behavior. The resume is executed whenever the exception that is defined inside the operator occurs. The exception `set_again` is used here, to resume to the timeout period whenever a timer is set after a previous reset.

Further work is needed to develop the complete semantics definition of TMSCs. We will take into consideration the ongoing work at ITU-T on the semantics definition of MSC'96.

5. Conclusions

This paper presents a new approach for timed MSCs that is mainly oriented at the use of Message Sequence Charts in the area of performance testing. TMSCs offer means to describe timing requirements but also timed measurements. The latter is the main difference to other proposals on time extensions for MSCs.

Front end tools for TMSCs were already developed [12]. They include a graphical editor (written in Tcl/Tk under Solaris and Linux), a scanner and parser (written in lex and yacc), the abstract syntax (written in Kimwitu), and transformers between the concrete instance-oriented syntax, the concrete event-oriented syntax, and the abstract syntax of TMSCs.

In the future, we will complete the semantics definition for TMSCs and will concentrate on simulators for TMSCs and test case generators from SDL/TMSCs specifications.

References

[1] H. Ben-Abdallah, S. Leue: Expressing and Analyzing Timing Constraints in Message Sequence Charts Specifications.-Technical Report 97-04, Electrical and Computer Engineering, University of Waterloo, April 1997.

[2] C. Facchi. Formal Semantics of Time Sequence Diagrams. Technical Report TUM-I9540, Technical University Munich, 1995.

[3] N. Faltin, L. Lambert, A. Mitschele-Thiel, F. Slomka: PMSC - Integrating Performance into Message Sequence Chart.- Proc. of FBT'97 (in this issue), Jun. 1997.

[4] ITU-T. Z.120 - Message Sequence Chart (MSC'96) - Working Draft, Jan. 97.

[5] ISO/IEC: Enhancements to LOTOS (E-LOTOS).- ISO/IEC JTC1/SC21/WG7: Working Draft on Enhancements to LOTOS, Jan. 97.

[6] P.B. Ladkin and S. Leue. Interpreting Message Flow Graphs. Formal Aspects of Computing 7(5), p. 473-509, Sept./Oct. 1995.

[7] P.B. Ladkin and S. Leue. Comments on a Proposed Semantics for Basic Message Sequence Charts. The Computer Journal, 37(9), January 1995.

[8] S. Mauw. The Formalization of Message Sequence Charts. Computer Networks and ISDN Systems , 28(12):1643-1657, 1996.

[9] S. Mauw and M.A. Reniers. An Algebraic Semantics of Basic Message Sequence Charts. The Computer Journal, 37(4):269-277, 1994. Also appeared as: Computing Science Notes 94-17, Department of Computing Science, Eindhoven University of Technology, April 1994.

[10] S. Mauw and M.A. Reniers. High-Level Message Sequence Charts. Draft version. Accepted for the Eighth SDL

1. The only exceptions for this rule are the timed wait and set and reset of a timer that are not separated from the causal ordering of events. However, a timer is represented by a separate constraint to define its timeout and reset semantics.

Forum, 1997, France.

[11] S. Mauw and E.A. van der Meulen. Specification of Tools for Message Sequence Charts. In ASF+SDF'95, Amsterdam, 1995.

[12] O. Mertens: A graphical Editor for Timed MSCs.- MSc thesis, Electrical Engineering, Technical University Berlin, May 1997 (in german only).

[13] E. Rudolph, P. Graubmann, J. Grabowski: Tutorial on Message Sequence Charts .- Computer Networks and ISDN Systems, Vol. 28, p. 1629-1641, 1996.

[14] Ch. Schaffer: MSC/RT: A Real-Time Extension to Message Sequence Charts (MSCs).- Technical Report TR140-96, Johannes Kepler University of Linz, 1996.

Appendix

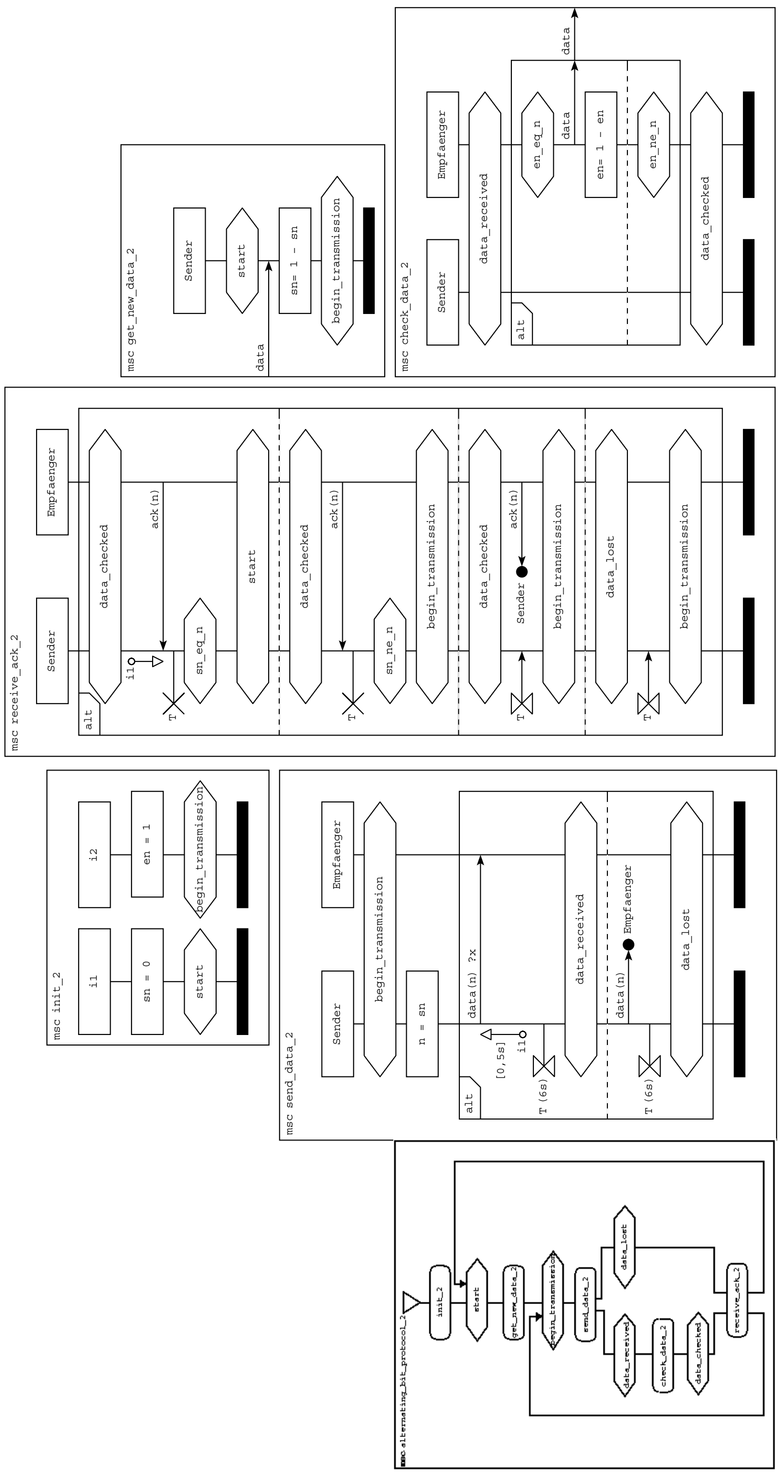


Fig. 6 Alternating Bit Protocol - Second Version

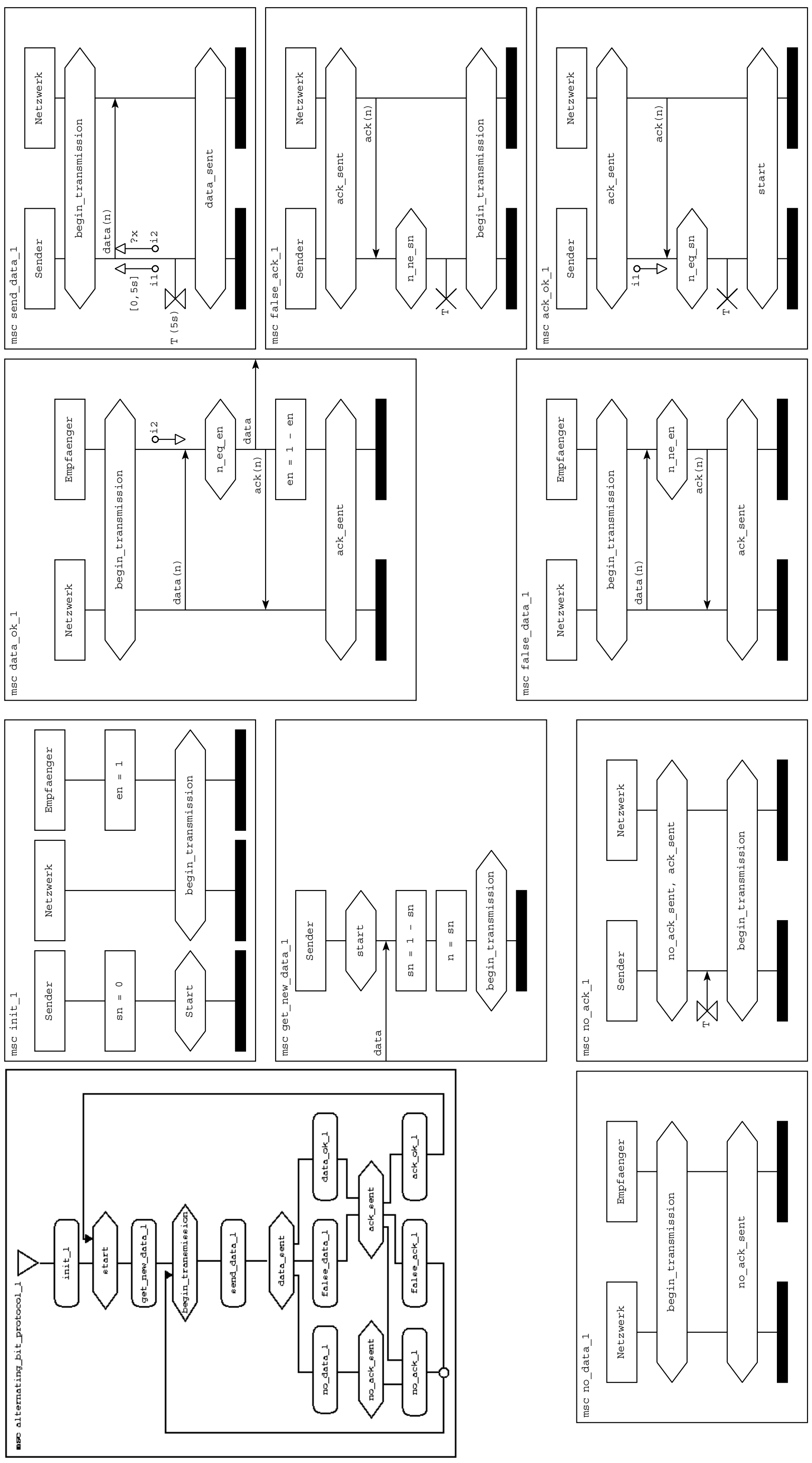


Fig. 7 Alternating Bit Protocol - First Version