Semantics for Timed Message Sequence Charts via Constraint Diagrams

Volker Grabowski, Cheryl Dietz and Ernst-Rüdiger Olderog
University of Oldenburg, Department of Computer Science
P.O. Box 2503, 26111 Oldenburg, Germany

Abstract

For Timed Message Sequence Charts we define a novel semantics in terms of Constraint Diagrams [3], a graphic notation for real-time properties stated in the Duration Calculus [10]. For the subset of untimed Basic Message Sequence Charts we prove consistency of the new semantics with the standard process algebra semantics due to Mauw and Reniers [6]. Based on this semantic study we comment on concepts and formalization of Timed Message Sequence Charts.

Keywords

Timed Message Sequence Charts, Constraint Diagrams, Semantics, Real-Time Requirements

1 INTRODUCTION

Graphical notations are important for the communication between application experts and computer science experts. That's why Petri nets, state charts or SDL diagrams are so popular in engineering applications. These well understood notations all serve to display state transition models of reactive systems.

Less well understood is how more abstract bahavioural requirements can be described graphically. For the application area of telecommunication Message Sequence Charts (MSCs) developed in the context of SDL have become a standard. While MSCs originally displayed only typical individual communication traces of reactive systems, various extensions of this basic idea are currently under development. The extensions deal with features like inline expressions, co-regions, and real-time. There is a danger that with all these extensions the intuitive appeal and clarity of the original MSCs gets lost. This can be seen for example in the increasingly sophisticated process algebra semantics of MSCs and its extensions.

Therefore it is interesting to look also into other approaches of graphic formalization of behavioural requirements. One such approach is based on the concept of *timing diagram* informally used in hardware design. Starting from this concept Schlör and Damm have developed the notion of a *symbolic timing diagram* to express temporal logic properties of reactive systems [9]. These diagrams have been applied in several industrial projects as requirement specifications for StateMate designs.

Also derived from timing diagrams and motivated by the work of [9] are *Constraint Diagrams* (CDs) [3]. They have been designed to specify real-time requirements of systems by relating the time dependencies of different time dependent observables. Their formal semantics is based on the *Duration Calculus* [10, 5], a real-time logic and calculus based on interval temporal logic [8]. Thus CDs can be seen as a graphic notation for a subset of Duration Calculus.

In this paper we present a new semantics for Timed MSCs using CDs (Section 4). The definition is not trivial

*This work was partially funded by the Leibniz Programme of the German Research Council (DFG) under grant Ol 98/1-1.

because the views of MSCs and CDs on systems are different: whereas MSCs have a communication-oriented view CDs have a state-oriented view. Our semantics will bridge this gap by exploiting the duality between communications (events) and states and by introducing a suitable time dependent observable recording the communication traces. We demonstrate that for Basic MSCs our CD semantics looks very much alike the original MSC but for MSCs with more advanced features more complex auxiliary CDs are needed.

Our semantics is of course just a definition. How does it relate to the already existing process algebra semantics for Basic MSCs [6] ? In Section 5 we show a consistency result of the new semantics w.r.t. the original semantics.

We believe that this semantic study sheds new light on MSCs and comment in the conclusions on current proposals for extending MSCs.

2 TIMED MESSAGE SEQUENCE CHARTS

Message Sequence Charts (MSCs) describe the asynchronous communication between components of concurrent systems and with an environment. Therefore it uses a set of so-called *instances* representing the single components. Graphically, an instance is a vertical line where events occurring during the execution are ordered. An MSC thus provides an event-oriented view on the behaviour of a system. Time evolves from top to bottom. The system's environment is reflected though a bordering line around the chart.

Instances communicate with each other and with the environment via asynchronous transmission of messages. The entities involved thus need not be ready for communication at the same time. Sending and receiving of messages is temporally distant. Communications are represented by arrows between the instances (resp. an instance and the bordering line for the environment). The MSC *M1* in Figure 1(a) includes two examples for communications. First, instance *I1* sends a request to instance *I2*. Afterwards, *I2* transmits an acknowledge to *I1*. The intervals beside parts of instances and arrows describe temporal requirements for the execution. Allowing the formulation of such requirements yields an extension of the MSC language, as explained in the following. Elements of the extended language are called *Timed MSCs*.

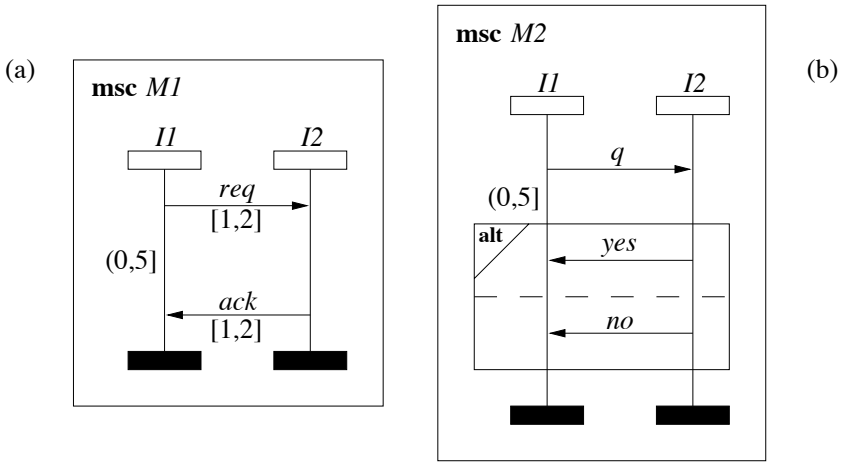

Figure 1 Two MSCs

Similar to the approach of R. Alur, G.J. Holzmann and D. Peled [2], we consider time requirements. On the

one hand, they can appear after an the occurrence of an event restricting the time passing until the next event. On the other hand, they restrict the temporal distance between send and receipt of a message. Time conditions are given by intervals where the lower bound represents the minimal time and the upper bound the maximal time allowed between two events. In the graphical representation they are written next to the corresponding section of an instance or an arrow. This can be seen in the sample MSC *M1*. There two time conditions for the duration of transmitting a request or acknowledge and one time condition restricting the time between sending the request and receiving the acknowledge on instance *I1* appear. The conditions associated to the arrows thus mean that transmitting the message may take between one and two time units. The interval $[0, 5)$ requires that the acknowledge hat to arrive at most five time units after the request was sent.

More complex specification can be built from these basic elements using *inline expressions*. They provide several operators for combining single MSCs. Among these in this article just alternative composition is considered. An example of its usage is found in Figure 1(b). After sending the question q from *I1* as answer yes or no can arrive. The MSC *M2* illustrates the versatility of time conditions. The requirement given through the interval $(0, 5]$ restricts the occurrence time of the next event independently from the decision which operand in the alternative is chosen.

The MSC language standardized in 1996 includes some more concepts, such as state conditions, internal actions, co-regions and high-level MSCs. They are not considered here as they play no role for discussing the temporal aspects.

3 CONSTRAINT DIAGRAMS

Requirements concerning the behaviour of reactive systems often consist of an assumption and a commitment part. This assumption/commitment style is embodied in CDs. The idea of this graphical language is illustrated with the safeness requirement for a register accessed by a writer and a reader in Figure 2: whenever a read (r) occurred non-overlapping with a write (w) with written value $wv = k$, this last written value is returned (read value $rv = k$). The concept of 'non-overlapping' is expressed here by the arrow between the two lines stating that the read phase (r) occurs after the write phase (w).

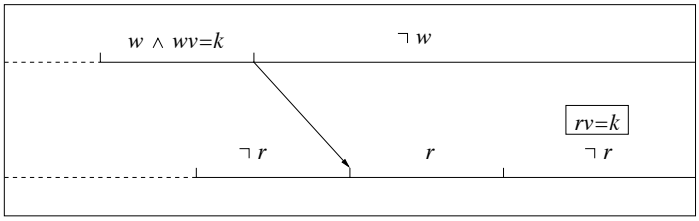

Figure 2 Safeness requirement

As in MSCs, different lines characterise the behaviour of components of a system considered. The state-based view in CDs is reflected in annotations of phases on these lines with Boolean expressions. E.g. the first line in the safeness requirement stands for a behaviour where the writer at the beginning behaves arbitrarily (indicated by the dashed part of the line), then writes the value k and afterwards leaves the write phase. The diagram leaves it open how a state change e.g. from write to non-write mode is achieved. This makes CDs a fairly abstract way of specifying requirements.

The phases on lines are furthermore determined by time requirements indicated by intervals; if no particular

length is supposed, i.e. a length in $(0, \infty)$ is considered, no annotation is made. Time dependencies between components are introduced by arrows decorated with intervals between the lines.

All commitments are highlighted by boxes. They may also appear in the past of assumptions in a CD which allows non-operational requirements to be formulated. An example for this phenomenon is given in Figure 3. It gives a dual requirement to safeness considered above: Whenever the value k is read, it must have been written before.

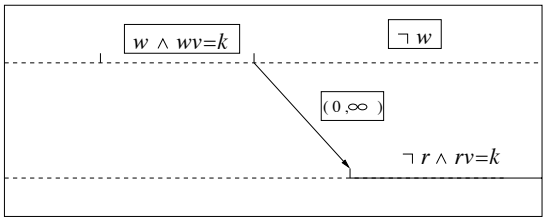

Figure 3 Correct read requirement

To give the reader a taste of the semantics of CDs defined in [3], we informally explain the Duration Calculus formulas corresponding to the example CD. Lines in CDs are characterized through so-called sequence formulas like

$$\ell = \epsilon_1; (\lceil w \wedge wv = k \rceil \wedge \ell = \epsilon_2); \lceil \neg w \rceil$$

for the line describing behaviour of the write component; where $F_1; F_2$ (read 'F_1 chop F_2') denotes an interval where first F_1 and then F_2 holds, e.g. $\ell = \epsilon_1$ means that the length of the subinterval considered is the value of the real variable ϵ_1 and $\lceil w \wedge wv = k \rceil$ means that the Boolean expression holds throughout the respective subinterval. Similarly, the semantics of the line for the reader is

$$\ell = \delta_1; (\lceil \neg r \rceil \wedge \ell = \delta_2); \lceil r \rceil; \lceil \neg r \lceil \ .$$

Constraints described with arrows are semantically captured with inequations like

$$\delta_1 + \delta_2 \geq \epsilon_1 + \epsilon_2$$

for the arrow between the lines for write- and read component.

The overall semantics of the CD is an implication between formulas describing assumptions and formulas for commitments. Thus, the example CD has the semantics

$$\begin{array}{l} (\ \ell = \epsilon_1; (\lceil w \wedge wv = k \rceil \wedge \ell = \epsilon_2); \lceil \neg w \rceil \\ \wedge\ \ell = \delta_1; (\lceil \neg r \rceil \wedge \ell = \delta_2); \lceil r \rceil; \lceil \neg r \rceil \\ \wedge\ \delta_1 + \delta_2 \geq \epsilon_1 + \epsilon_2) \\ \Longrightarrow \ell = \delta_1; (\lceil \neg r \rceil \wedge \ell = \delta_2); \lceil r \rceil; \lceil \neg r \wedge rv = k \rceil \ . \end{array}$$

The relation between CDs and MSCs is discussed in the conclusions.

4 THE NEW SEMANTICS

The aim of this section is to give a semantics for Timed MSCs that captures all the temporal aspects of the extended language. As a formal foundation, real-time interval logic is a natural choice. But instead of directly assigning a formula to each MSC, it is interpreted by a set of CDs which collectively describe the intended behaviour of the chart. This way, we take advantage of the clarity of graphical formalisms as opposed to the complexity of temporal formulas and it allows a comparison of the expressiveness of MSCs and CDs. Since the CD-semantics of an MSC M shows how to express M by several CDs, it is also often called the *(CD-)translation* of M.

For lack of space, we can't present the entire semantics definition (as featured in [4]). Instead, we confine ourselves to explaining the basic ideas behind this definition by discussing the semantics of the two sample MSCs in Fig. 1. Since the translation of MSCs without coregions and inline expressions has a quite simple structure, we start with $M1$. So, how could $M1$ be expressed by CDs?

First of all, it is obvious that MSCs and CDs are based on very different approaches to specifying a system. MSCs concentrate on the communication structure of a system. To this end, they specify the outputs and inputs of all the components. Thus, an MSC renders an event-oriented view of a system. CDs follow a different way to describe the intended behaviour. They focus on the states the components assume during execution. Therefore, we introduce a state variable tr, called the *trace variable*, which holds at any given point in time the sequence of events that have happened up to this moment. It acts as a link between events and states. Since CDs focus on states, it is necessary to record the state of every instance during execution. It is not possible to determine the value of tr directly (i.e. without any state information). Hence, for every instance I in the MSC we need a local state variable X_I (the term "local" indicates that X_I doesn't contribute to the observable behaviour).

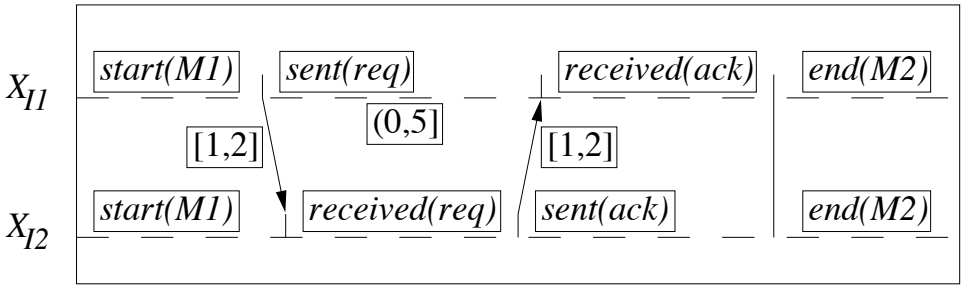

Figure 4 $Main(M1)$

The evolution of the local variables is determined by the CD $Main(M1)$ (Figure 4), which contains no assumptions. (MSCs are thus interpreted as specification for one execution. They could also be viewed as constraints that must be fulfilled whenever the initial state is reached during execution. This view is taken for handling inline expressions and is explained below.) The execution of each instance begins in the initial state $start(M1)$ and terminates in the final state $end(M1)$. Events in $M1$ correspond to state transitions in $Main(M1)$. Though the actual value of the X_I's has no influence on tr, for clarity, we chose an intuitive name for each state (apart from control states such as $start(M1)$ and $end(M1)$) that indicates which event has happened last. Time conditions and other dependencies resulting from communication between instances are modelled by arrows in a straightforward manner.

Several CDs are required to correctly describe the behaviour of tr, called the *trace-CDs*. Due to space limita-

tions, we describe them only briefly without display. For every instance I, the CD $TraceChange_I$ ensures that every state transition at X_I leads to an appropriate modification of tr. The trace variable has to be expanded by the corresponding event. Accordingly, the CD $StateChange_I$ guarantees that every modification of tr is caused by an appropriate state transition. By these CDs a one-to-one relationship between events and state transitions is achieved. For technical reasons, two more CDs are necessary. The diagrams $TraceType(M1)$ and $TraceMon$ make sure that tr evolves correctly. They demand that tr contains only events out of $M1$ and that tr is monotonic. These trace-CDs correspond to a set of trace properties needed in the Consistency Theorem in Section 5.

This completes the semantics of $M1$. Obviously, the graphical structure of $M1$ and $Main(M1)$ are quite similar. This is not always the case. As mentioned above, MSCs that feature coregions or inline expressions complicate the translation. To illustrate the resulting problems, we examine the semantics of MSC $M2$ from Figure 1 which features an inline expression. The trace-CDs don't differ from the ones for $M1$. For the CD $Main(M2)$, however, this is not the case.

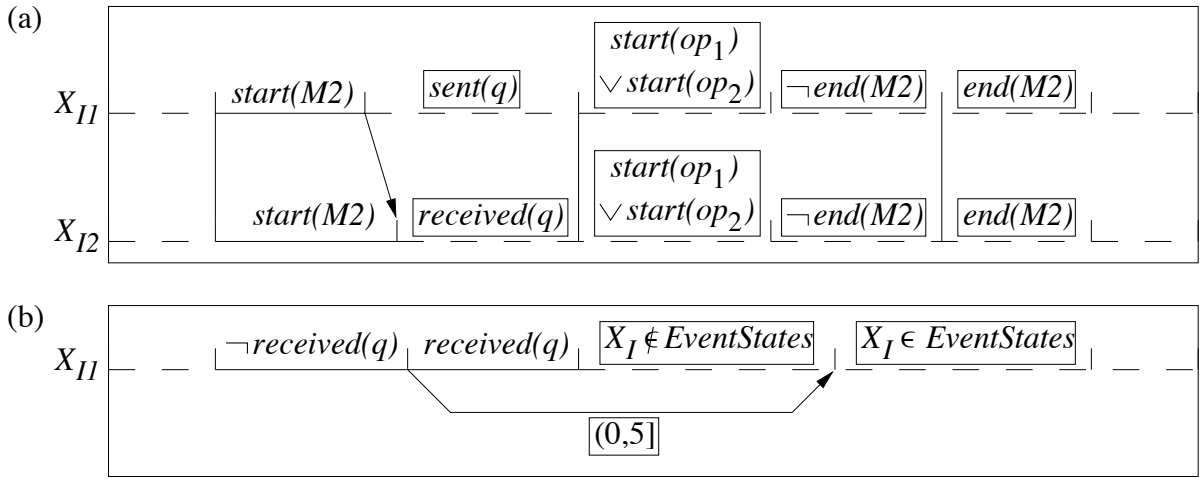

Figure 5 (a) $Main(M2)$, (b) time-CD

The occurrence of an inline expression increases the complexity of the semantics definition. Obviously, the execution of the alternative composition can't be described by one single CD since there is more than one possible behaviour. Because the operands of inline expressions are again MSCs, the semantics of arbitrary MSCs has to be defined inductively. As for $M1$, the behaviour outside of inline expressions is described by the CD $Main(M2)$ which can be seen in Figure 5(a). There is one striking difference between $Main(M1)$ and $Main(M2)$. While $Main(M1)$ contains no assumption and therefore describes an unconditional execution, the CD $Main(M2)$ only applies if all instances are in the initial state $start(M2)$. Additionally, $Main(M2)$ doesn't require the final state to be stable forever like in $Main(M1)$. These differences are caused by the inductive definition. Since every MSC is potentially an operand of an inline expression, the translation has to take into account that the chart is possibly only a small part of a much bigger specification. There is another detail that attracts the attention: the state transitions at the beginning and the end of the execution of the inline expression are connected by vertical lines which indicate synchronization of the involved components. This means that the alternative composition is connected with the rest of the chart via strong sequential composition. We discuss this topic in detail in the conclusion (Section 6).

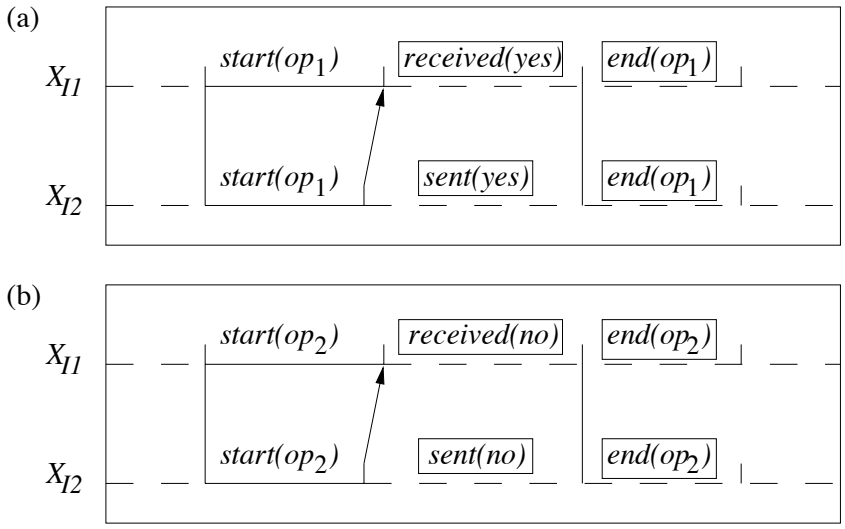

Figure 6 (a) $Main(op_1)$, (b) $Main(op_2)$

This leaves the operands of the alternative composition in $M2$ to be modelled correctly. As indicated, at this point of the semantics definition the induction starts. Each operand is treated like a single MSC, i.e. for operand i a CD $Main(op_i)$ is defined like before for $M1$ and $M2$. Thus, the translation of $M2$ has to be augmented by two more CDs $Main(op_1)$ and $Main(op_2)$ (which can be seen in Figure 6). There are no additional trace-CDs needed for the operands since they contain the same instances as $M2$. The CDs $Main(M1)$ and $Main(M2)$ describe the complete execution of the operands but they don't say when it has to take place, i.e. at what point it starts and when it ends. To connect the operands with the rest of the system, two more CDs are needed for every operand j which describe the state transition from the preceding state to $start(op_j)$ and from $end(op_j)$ to the state following the inline expression. Like the trace-CDs, these diagrams were omitted here.

The time condition in $M2$ is immediately followed by an inline expression. This means that it doesn't refer to a unique pair of events. Instead it limits the amount of time between the sending of q and the reception of the answer which can be yes or no. To model the time condition correctly, another CD is called for. When the preceding event (the output of q) has happened, the next one has to occur within the specified time interval. This is achieved by the CD in Figure 5(b). The set $EventStates$ contains all local states that correspond to MSC events. The occurrence of the next event is modelled by the state transition between the phases $X_I \notin EventStates$ and $X_I \in EventStates$.

Eventually, the initial state and the final stability of the system which were left out of $Main(M2)$ due to the inductive definition have to be specified by additional CDs. The CD $start(M2)$ states that the execution of $M2$ begins in $start(M2)$. For every instance I, the CD $EndStable_I(M2)$ guarantees that the system is stable once it reached the final state $end(M2)$.

The semantics definition of $M2$ is now complete. It illustrates the general approach to translating an MSC into CDs. The semantics of $M1$ demonstrates just an interesting special case. An MSC M without inline expressions and coregions has a very simple semantics consisting only of the trace CDs and one more CD which combines $main(M)$ and the CDs $start(M)$ and $EndStable_I(M)$.

5 CONSISTENCY RESULT

Definining a new semantics for a language always leads to the question whether the chosen approach is appropriate. To justify our definition, we proved its consistency with the standard process algebra semantics by Mauw and Reniers [6] which assigns a suitable process term to every MSC. Generally speaking, it states that the set of values for tr that are allowed by the CD semantics is equal to the set of prefixes of traces described by the process term (under certain general properties for traces, such as 'the trace only grows').

Theorem 1 (Consistency) *For an MSC M, let $[\![M]\!]$ denote its CD semantics and let $pref(M)$ be the set of all prefixes of traces described by the process term semantics. Then the following equivalence holds:*

$$[\![M]\!] \Leftrightarrow (\lceil tr \in pref(M) \rceil \wedge TraceProperties)\,.$$

Proof. *See [4].* □

Recall that tr is a free observable in the new semantics $[\![M]\!]$. Thus the Consistency Theorem states that every trace tr which satisfies the CD semantics $[\![M]\!]$ is a member of $pref(M)$. Due to the nature of the semantics of CDs, which just guarantees that an initial part of the specified behaviour takes place, this is the strongest possible result. On the other hand, the theorem also states that for every trace in $pref(M)$ there exists an interpretation for tr satisfying the *TraceProperties* which fulfills the CD semantics $[\![M]\!]$.

6 CONCLUSIONS

In this extended abstract we sketched a real-time semantics for MSCs that formalizes constructs like messages with time conditions. For Basic MSCs [6] our CD semantics looks very much alike the original MSC but for MSCs with more advanced features more complex auxiliary CDs are needed.

One crucial point of the presented semantics is synchronization of the instances at the beginning and the end of an inline expression. Thus, an inline expression is connected with the rest of the chart via strong sequential composition. Why didn't we employ its counterpart, the weak sequential composition, like Mauw and Reniers did in [7]? Attempts to model the weak sequential composition with CDs proved to be quite complex, presumably due to the state-based nature of CDs. For instance, several problems arise when combining alternative and weak sequential composition. While the alternative describes a global choice which generally involves several instances, the weak sequential composition allows certain instances to be ahead of others in execution. As a result, the first instance to reach the alternative composition (during execution) has to inform the others about which choice he took, so that they can follow him. Fig. 7 shows an example for this problem (the boxes in this MSC denote internal actions). The process $P1$ may enter the alternative composition before the message m has arrived at $P2$. If, for instance, $P2$ receives m after $P1$ has executed the action a, $P2$ has to execute the action b next. In this case, $P2$ must follow the decision previously made by $P1$. As a consequence, some kind of synchronization between $P1$ and $P2$ has to take place. However, this synchronization process is by no means displayed in the graphical notation and thus defeats its intuitive comprehensibility. In addition, the MSC doesn't reflect the complete communication structure of the resulting implementation since all of the communications needed for synchronization are missing.

The MSC in Figure 8 demonstrates another problem caused by the weak sequential composition. Since the weak sequential composition doesn't require any synchronization between instances, $P1$ can send the message m an arbitrary number of times without it ever reaching its destination. Such a behaviour is obviously unrealistic.

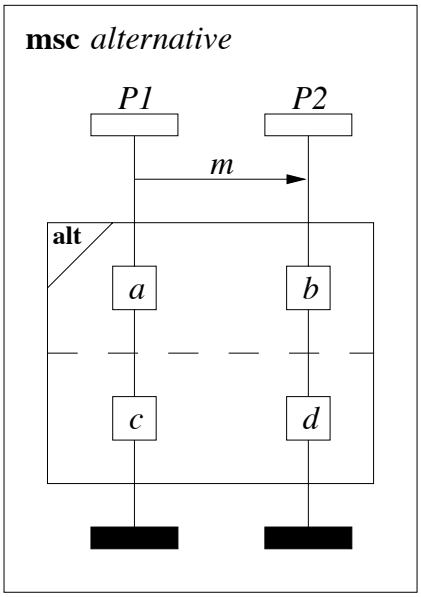

Figure 7 MSC *alternative*

No matter how the communication between instances is finally implemented, it will only be possible to buffer a limited amount of messages.

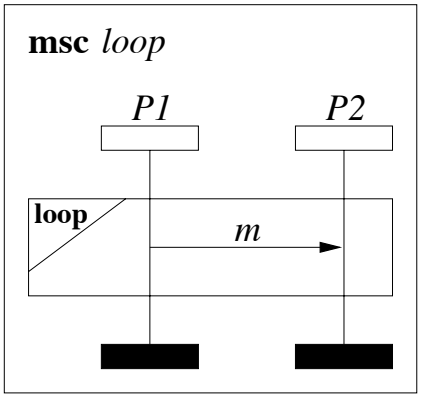

Figure 8 MSC *loop*

Ben-Abdallah and Leue encountered similar problems while designing a tool for system development from MSCs [1]. They solved them by considering only MSCs with certain properties (for example, demanding that instances only communicate through handshaking to avoid process divergence). However, we feel that the main problem is the choice of weak sequential composition as an operator. Hence, we decided to use the strong sequential composition for our definition of the MSC semantics.

Finally, we'd like to point out some aspects that distinguish CDs from MSCs. Both formalisms operate on a different level of abstraction and are based on opposing approaches to specifying a system (event-oriented versus state-based). CDs leave it open how a state change in one instance is achieved. This makes CDs a fairly abstract way of specifying requirements independently from a certain communication paradigm. In CDs, arrows do not indicate communication as in MSCs but time distance between events; the effect of communications is left implicitly in a CD like in the interrelation between the Boolean expressions $wv = k$ and $rv = k$ in Figure 2. What is explicit in that CD is the temporal distance between sending and receiving a message. Moreover, commitments may also appear in the past of assumptions in a CD allowing non-operational requirements to be formulated which is not possible with MSCs. Nevertheless, MSCs can be viewed as a subclass of CDs. It will be worthwhile to consider extensions of MSCs inspired by concepts used in Constraint Diagrams.

REFERENCES

[1] H. Ben-Abdallah; S. Leue: MESA: Support for szenario-based design of concurrent systems. In B. Steffen, editor, *Tools and Algorithms for the Construction and Analysis of Systems*, LNCS 1384, Springer, 1998.

[2] R. Alur; G. J. Holzmann; D. Peled: An Analyzer for Message Sequence Charts. In T. Margaria and B. Steffen, editors, *Proceedings of the 2nd International Workshop on Tools and Algorithms for the Construction and Analysis of Systems (TACAS95)*, Passau, Germany, 1996.

[3] C. Dietz: Graphical Formalization of Real-Time Requirements. In B. Jonsson and J. Parrow, editors, *Formal Techniques in Real-Time and Fault-Tolerant Systems*, LNCS 1135, Springer, Uppsala, Sweden, 1996.

[4] V. Grabowski: Graphische Spezifikationsformalismen: Message Sequence Charts und Constraint Diagrams. Masters Thesis (in German), FB Informatik, Univ. Oldenburg, Dez. 1997.
`http://theoretica.informatik.uni-oldenburg.de/diplom.html`

[5] M. R. Hansen; C. Zhou: Duration Calculus: Logical Foundations. *Formal Aspects of Computing*, 9, 1997, 283–330.

[6] S. Mauw; M.A. Reniers: An Algebraic Semantics of Basic Message Sequence Charts. *The Computer Journal* 37, No. 4 (1994).

[7] S. Mauw; M.A. Reniers: High-Level Message Sequence Charts. In A. Cavalli and A. Sarma, editors, *SDL'97: Time for Testing – SDL, MSC and Trends*, Proceedings of the eigth SDL Forum, Elsevier Science Publishers B.V., Evry, France, 1997.

[8] B. Moszkowski: A Temporal Logic for Multi-Level Reasoning about Hardware. *IEEE Computer*, 18(2), 1985, 10–19.

[9] R. Schlör; W. Damm: Specification and Verification of System Level Hardware Designs using Timing Diagrams. *Proceedings of the European Conference on Design Automation*. Paris, France, 1993.

[10] Zhou Chaochen; C.A.R. Hoare; A.P. Ravn: A calculus of durations. *Information Processing Letters*, 40/5, 1991, 269–276.