Disrupt and Interrupt in MSC: Possibilities and Problems

Th. Cobben, A. Engels
Eindhoven University of Technology,
P.O. Box 513, NL–5600 MB Eindhoven, The Netherlands.
engels@win.tue.nl

Abstract

Disrupt and interrupt are a possible extension of the MSC language. However, such an extension is not without risks. Before it can be done, there must be a decision about the semantics that are used to implement it, and even if that is going to be done, they might well have unwanted consequences. In this paper we show the most important choices in implementing these constructs, propose a semantics and show a few of the problems that disrupt and interrupt might cause.

Keywords
MSC, disrupt, interrupt, language extensions, semantics

1 Introduction

The language of MSC (Message Sequence Chart) has been extended with various operators recently [1], and semantics for these have been proposed [2]. However, a number of additional expansions for the language have already been proposed, and might be introduced in the next version of the language, which is to be accepted in the year 2000. Two of these are disrupt and interrupt. Our opinion is that the semantics of a new construct should be well thought out before it is added to the language. For disrupt and interrupt this paper attempts to make such a pre-introductory semantic overview. We will show the most important of the many choices that have to be made, and will show some of the problems that might occur if these operators are introduced.

We feel that, in general, it is a bad thing to let the language grow too fast or too large. The MSC'96 language has not been thoroughly researched. It would be better, in our opinion, to have a solid, stabilized semantics for the existing language, and if possible also for the proposed extensions, before the language is extended. There are other reasons for restraint in the adoption of new features as well: If features are introduced too quickly, tool builders will have problems keeping up, having too many features runs the risk of groups of users using only subsets of the language, thus diminishing the advantage that using one single languages has. Another problem is that a large number of features greatly increases the chance that unforeseen interactions between them lead to unwanted or unexpected behaviour.

We do not intend to claim that additions to the language have to be avoided at all costs. Far from that, some additions are certainly useful, and not having any innovation whatsoever will be even more certain to kill the language's applicability than a too generous addition of new features would. However, new features should only be introduced when there is a wish for inclusion by a large number of users, and a well-defined semantics for it.

Even though disrupt and interrupt are much wanted by a number of users, there does not seem to be a consensus about their exact meaning. Intuitively, interrupt means something like 'while doing something, wait for a while to do something else, then do the first thing again'. Likewise, disrupt is something like 'stop doing something to do something else instead'. However, from these intuitions many different implementations can be, and in fact have been, created. There are a large number of choices to be made, and each of them will influence what the meaning of disrupt and interrupt in MSC will be. All such choices have to be made in a unique way, although this may well mean that some, or even many, users will not have 'their' interrupt or disrupt in the language. One could argue that the popularity of disrupt and interrupt might thus be smaller than it seems. In this paper we will show a few of the most important choices to be made.

1.1 Acknowledgments

We would like to thank Sjouke Mauw, Michel Reniers, Loe Feijs and Jos Beaten for their support, both technical and editorial, in compiling this document.

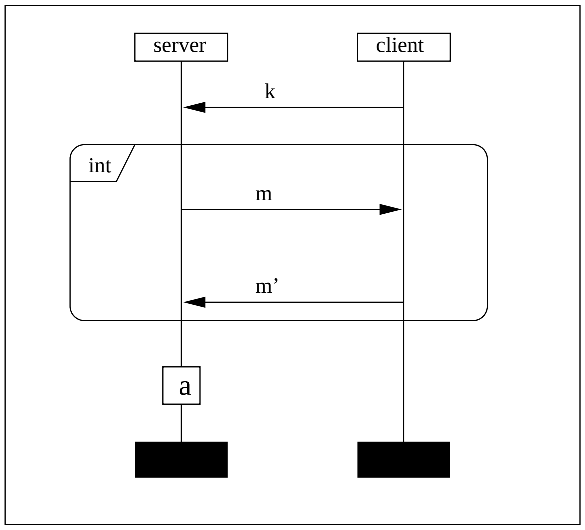
2 Syntax

If disrupt and interrupt would be included in the language, there would not only be a need for a semantics, but for a syntax as well. These two subjects are of course not independent. Semantics that fit well with a certain syntax can be clumsy or illogical when combined with another syntax, and vice versa.

The main distinction here is between *local* and *global* interrupt (or disrupt). The difference here is the period during which the disrupt or interrupt can take place. In a local interrupt or disrupt, the disrupt or interrupt can only take place at a single point in time, while a global interrupt or disrupt can do so at any time during a given period.

We would like to stay as close as possible to the existing MSC syntax, so we will choose to adapt a construct that is already in the language. The most logical choice is of course using inline expressions. This leads to a possible syntax as shown in Figure 1.

In Figure 1 we see examples for both local and global syntax. In the left a local interrupt is displayed. When all instances are at the point in time just before the interrupt, the main line of flow (presented by the part of the MSC outside the inline expression) can be interrupted if, and only if, all instances (at this case two) are at the point in time just before it.



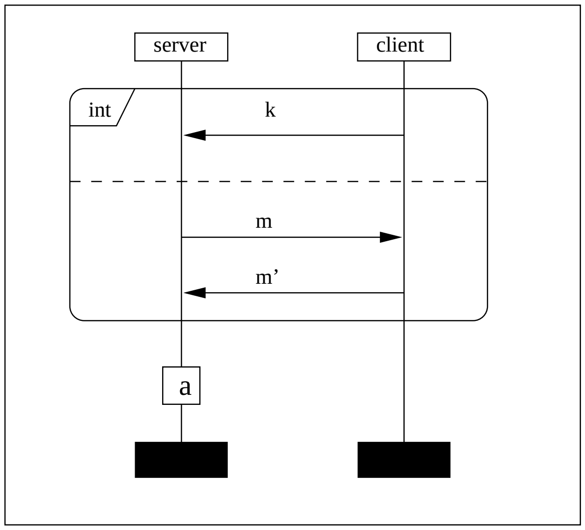


Figure 1: Proposed syntax for (left) local and (right) global interrupt

The right MSC shows our proposed syntax for interrupt, would the global variant be chosen. As it is important here to specify at which times an interrupt is possible, we now have a two-component inline expression. The lower part is equal in function to the inline expression in the left example, giving the interrupting sequence. The interrupt can take place at any time that all instances are somewhere in the time described by the upper part of the expression.

Looking more precisely at what happens in the local case, we find that the MSC can in fact have only two essentially different behaviours:

- Not doing the interrupt.
- Doing the interrupt, and doing it at exactly at the time given.

But that is a semantics we already have! The *opt*-construct has exactly this same meaning - when something is placed in an 'optional' inline expression it can either be done at that precise moment, or not at all. Likewise a local disrupt could be replaced by an 'exception'.

Such an equality has advantages and disadvantages. The advantage is, that a semantics is easily found, and will cause no problems with the rest of the language, or at least no problems that were not already there. The disadvantage is, that adding such a disrupt or interrupt will not give any added functionality. That way, the language would be made larger without making it any more powerful. As enlargening the language is on itself undesirable, although often necessary, such extensions should be avoided as much as possible.

Because local interrupt and disrupt are easily implemented semantically, but global interrupt and disrupt are not, we will take a look at the semantics of the latter in the following two sections.

3 Semantics

3.1 Semantic choices

In the global semantics of disrupt and interrupt a number of other decisions have to be made. Each of these will influence the resulting semantics. We see at least the following:

1. Can an interrupt take place only once, or any number of times?
2. In the second case, can the system be interrupted more that once between any two actions?
3. If yes, can one interrupt interrupt the other?
4. Can an interrupt or disrupt take place before the first and/or after the last action of what is interrupted?
5. Does an interrupt or disrupt all instances at the same time, or is it enough that all instances are interrupted or disrupted at some time? This point will be explained in more detail below, as it is an important choice, and the most obvious answer might well be the wrong one.

The last point above deserves some extra discussion. At first thought it might seem that the first interpretation is more natural - an interrupt or disrupt should work on the whole system, or at least on all instances on which it is specified, at once. However, when we look at a quasi-practical example, this might not be as obvious. See for example Figure 2. It models a Telnet-protocol: The server sends two (packets of) messages to the client. The client can check whether the server is still alive by sending an *ayt*-signal ('are you there?'), to which the server answers by saying '*yes*'.

Now if we regard the interrupt as interrupting all instances at once, then the sending of the ayt-message will block the action of the server. However, how is the server in a practical case to know that the ayt has been sent? It only notices this upon its receipt, so it is logical to assume it will not be blocked before that. Likewise, the server does not know when the 'yes' is received, only when it is sent. Thus letting the server be interrupted during all of the period leads to some possibly unwanted extra causalities. The more logical choice might be to let each process be interrupted separately, that is, each process has to do the interrupting actions at some point without doing anything else in between, but they do not have to do it all at the same time. When one would choose to have all instances interrupted or disrupted at the same time, one would in fact introduce synchronization points, which are contrary to the nature of MSC as it has been practiced until now.

For the other questions our prefered answers are as shown below. However, we do not feel very strongly about these questions, and if it appears that other choices would be closer to the wishes of the users, they are the ones whose opinions have to prevail.

1. Any number of interrupts.
2. More than one interrupt between two events possible.
3. No interrupts interrupting interrupts.
4. Interrupts and disrupts possible at the given times.

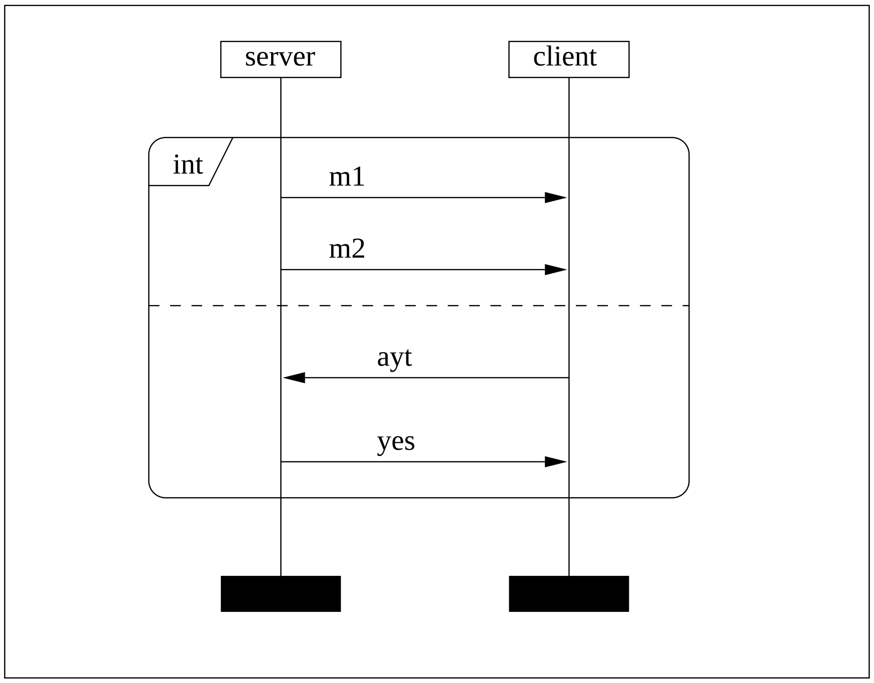


Figure 2: Example interrupt MSC: 'Are you there'-protocol

3.2 Semantics

Under the choices given in the last section we will try to create an operational semantics for the for the disrupt and interrupt operators. The official semantics for MSC'92 [3] was made in process algebra [4], but the new semantics for MSC'96 [2] for a large part are made operationally only. In a preliminary version of this paper [5] we have already given the semantics for the case that disrupts and interrupts happen simultaneously.

We will define operators $\rhd$ and $\blacktriangleright$ to denote the interrupt and disrupt respectively. That is, $x \rhd y$ is 'x (possibly) interrupted by y' and $x \blacktriangleright y$ 'x (possibly) disrupted by y'.

When could $x \blacktriangleright y$ do an action a? Well, there are in fact two possibilities: Either x does the action, or y does it. In the first case, the resulting expression can still be disrupted. In the second case, all instances, except the one on which a takes place do not have to have been disrupted yet. They have to be disrupted at some time, but that time can be somewhere in the future, and upto that point can still do actions of x. To describe this situation we introduce the forced disrupt, $|\!\blacktriangleright$. $x |\!\blacktriangleright y$ can do x, but must at some time in the future be disrupted by y. However, this is not enough yet: The action of y that has already been taken forbids any actions on the same instance of x to be taken. That is, some events of x may be done before being disrupted, but others have already been disrupted. Therefore we add to the forced disrupt a set of instances S that have already been disrupted. $x |\!\blacktriangleright^S y$ can now do any action from y (provided y could do it), but actions from x only if they happen on an instance not in S. We will be giving operational semantics for both $\blacktriangleright$ and $|\!\blacktriangleright$, although it would be possible to make the semantics without using $\blacktriangleright$, as it can be eliminated through the equation $x \blacktriangleright y = x |\!\blacktriangleright^{\varnothing} y \mp x$.

For the interrupt $\rhd$ we also define such a forced interrupt $\rhd\!\!\!|\,$, but in this case we cannot get away with re-defining the interrupt, as we need to keep the possibilities of further interrupts.

We have to check the behaviour of our operators for three operational modifiers: $x\downarrow$, which is true iff x can terminate, $x \xrightarrow{a}$, which gives the result of doing an a on x, and $x \overset{a}{\cdots\!\!\rightarrow}$, which gives the result of x permitting a. The latter is used when an action that, in the semantic description of the MSC, is after x, can happen before x despite of that. In this case, only those parts of x remain that do not contain any action on the same instance as the 'allowed' action. Any trace of x that would contain such an action is removed,

First termination. $x \blacktriangleright y$ can terminate in two ways: Either x terminates, or y disrupts x and then terminates without doing any action. For $x|\blacktriangleright^S y$ to terminate it is necessary and sufficient for y to terminate, while $x \triangleright y$ can terminate by just x terminating. Finally, for $x|\!\triangleright^S y$ to terminate, both x must be ready and y must have no interrupting actions left, so $x|\!\triangleright^S y$ only terminates if both x and y terminate. Thus:

$$\frac{x\downarrow}{x \blacktriangleright y \downarrow} \qquad \frac{y\downarrow}{x|\blacktriangleright^S y\downarrow} \qquad \frac{x\downarrow}{x \triangleright y\downarrow} \qquad \frac{x\downarrow, y\downarrow}{x|\!\triangleright^S y\downarrow}$$

$$\frac{y\downarrow}{x \blacktriangleright y \downarrow}$$

Then doing a step. When could an action, a, be taken by $x \blacktriangleright y$? There are in fact two possibilities: either x takes the step, after which a disrupt of course could still take place, or y disrupts, and takes the step. In the second case, we would get into a forced disrupt situation, where the instance on which a took place, which is denoted $I(a)$ is already disrupted.

In a forced disrupt $x \blacktriangleright^S y$ x could only take the action a if the instance on which a takes place was not already disrupted, that is, if $I(a) \notin S$. y can of course always occur, and if it does then necessarily its instance must be disrupted as well.

With the interrupt $x \triangleright y$ we again see two possibilities. Either the action can be done by x, and nothing particularly shocking happens, or it can be done by y. In the latter case we get into a forced interrupt again. However, we will have to keep the 'old' interrupt as well, because x could be interrupted a second time.

$x|\!\triangleright^S y$ is the most difficult one in this aspect. x can do the step if $x \notin S$, but it can also do it if y allows a. This is, because in this case the instance on which x takes place has done all it has to do, so it is not interrupted anymore, and can do steps from the main execution (x) again. Steps from y work just like the former cases.

There is another complicating factor here: We can have just one possible step for a given action from a given expression, because all our choices are deterministic. Thus when both x and y can do a given step, we have to include a special case. Taken together, this leads to:

$$\frac{x \xrightarrow{a} x', y \not\xrightarrow{a}}{x \blacktriangleright y \xrightarrow{a} x' \blacktriangleright y} \qquad \frac{x \xrightarrow{a} x', y \not\xrightarrow{a}, I(a) \notin S}{x|\blacktriangleright^S y \xrightarrow{a} x'|\blacktriangleright^S y}$$

$$\frac{y \xrightarrow{a} y', x \not\xrightarrow{a}}{x \blacktriangleright y \xrightarrow{a} x|\blacktriangleright^{\{I(a)\}} y'} \qquad \frac{y \xrightarrow{a} y', x \not\xrightarrow{a}}{x \blacktriangleright y \xrightarrow{a} x|\blacktriangleright^{S\cup\{I(a)\}} y'}$$

$$\frac{x \xrightarrow{a} x', y \xrightarrow{a} y'}{x \blacktriangleright y \xrightarrow{a} x' \blacktriangleright y \mp x|\blacktriangleright^{\{I(a)\}} y'} \qquad \frac{x \xrightarrow{a} x', y \xrightarrow{a} y', I(a) \notin S}{x|\blacktriangleright^S y \xrightarrow{a} x'|\blacktriangleright^S y \mp x|\blacktriangleright^{S\cup\{I(a)\}} y'}$$

$$\frac{x \xrightarrow{a}, y \xrightarrow{a} y', I(a) \in S}{x|\blacktriangleright^S y \xrightarrow{a} x|\blacktriangleright^S y'}$$

$$\frac{x \xrightarrow{a} x',\, y \overset{a}{\not\longrightarrow}}{x \triangleright y \xrightarrow{a} x' \triangleright y} \qquad \frac{x \xrightarrow{a} x',\, I(a) \notin S}{x |\!\triangleright^S y \xrightarrow{a} x' |\!\triangleright^S y}$$

$$\frac{x \xrightarrow{a} x',\, I(a) \in S,\, y \overset{a}{\cdots\longrightarrow} y''}{x |\!\triangleright^S y \xrightarrow{a} x' |\!\triangleright^S y''}$$

$$\frac{y \xrightarrow{a} y',\, x \overset{a}{\not\longrightarrow}}{x \triangleright y \xrightarrow{a} (x |\!\triangleright^S y') \triangleright y} \qquad \frac{y \xrightarrow{a} y',\, x \overset{a}{\not\longrightarrow}}{x |\!\triangleright^S y \xrightarrow{a} x |\!\triangleright^{S \cup \{I(a)\}} y'}$$

$$\frac{x \xrightarrow{a} x',\, y \xrightarrow{a} y'}{x \triangleright y \xrightarrow{a} x' \triangleright y \mp (x \triangleright y) |\!\triangleright^{\{I(a)\}} y'} \qquad \frac{x \xrightarrow{a} x',\, y \xrightarrow{a} y',\, I(a) \notin S}{x |\!\triangleright^S y \xrightarrow{a} x' |\!\triangleright^S y \mp x |\!\triangleright^{S \cup \{I(a)\}} y'}$$

$$\frac{x \xrightarrow{a} x',\, y \xrightarrow{a} y',\, I(a) \in S,\, y \overset{a}{\not\cdots\longrightarrow}}{x |\!\triangleright^S y \xrightarrow{a} x |\!\triangleright^S y'}$$

$$\frac{x \xrightarrow{a} x',\, y \xrightarrow{a} y',\, I(a) \in S,\, y \overset{a}{\cdots\longrightarrow} y''}{x |\!\triangleright^S y \xrightarrow{a} x' |\!\triangleright^S y'' \mp x |\!\triangleright^S y'}$$

Finally the permission relation. This relation has been introduced in the semantics of MSC to describe the possibility that in the expressio $x \circ y$ events of y can go before events of x. However, this can only be done if no events on x are on the same instance as the event taking place, or are otherwise forced to go first. This can depend on choices that are made within x. In such a case those choices that would have made the event taking place impossible are subsequently disallowed. Thus we get the relation $x \cdots\rightarrow x'$, which denotes that x by permitting an event from another (later) term is reduced to x'.

For $x \blacktriangleright y$ this immediately leads to problems. There are two possibilities here: Either x has permitted the event, or y has permitted it. However, in the second case, those events of x that would have permitted it, can still take place. That is, x may perhaps not take place in full, but it can still do those actions that are not forbidden by the event just having been allowed. This is not a simple removal of choices as it was with the permission relation for other MSC operators. Here those parts of x that would normally be disallowed by the permission of the events can still take place upto the place where the permission would actually be impossible.

To see how we can deal with this, it is good to look at the forced disrupt $x|\blacktriangleright^S y$. Here, in order for an event a to be permitted, it necessarily has to be permitted by y. On the other hand, whether or not x permits it is not interesting - any beginning of a trace in x can happen as long as it does not contain any events that are disallowed by the permission of a, that is, as long as it does not contain any events on the same instance $I(a)$, independent of whether or not they are part of a complete trace that would have allowed a. Thus we see that, if y permits an event am, changing into y'', $x|\blacktriangleright^S y$ permits that event, and changes into $x|\blacktriangleright^{S\cup\{I(a)\}} y''$. This looks strange, as this is independent of whether or not x permits a. The reason is that x cannot terminate anyway, as it will be interrupted by y at some time. Because of this it does not matter whether x, or even the trace taken, actually permits a, as long as that part of the trace that is actually taken does so. The SOS-rules for permission by $x \blacktriangleright y$ now follow through the equality $x \blacktriangleright y = x \mp x|\blacktriangleright^{\varnothing} y$.

For $x \triangleright y$ to permit a it suffices that x does so. If y does not allow a, the process cannot be interrupted anymore, if it does it still can. $x|\!\triangleright^S y$, finally, can permit an event only if both x and y do so. Note that it does not matter in this case whether or not $I(a)$ is added to S, as all events on $I(a)$ are 'sieved out' by the permission of a anyway. This leads to the following:

$$\frac{x \overset{a}{\cdots\longrightarrow} x'',\ y \overset{a}{\not\cdots\longrightarrow}}{x \blacktriangleright y \overset{a}{\cdots\longrightarrow} x''}$$

$$\frac{x \overset{a}{\not\cdots\longrightarrow},\ y \overset{a}{\cdots\longrightarrow} y''}{x \blacktriangleright y \overset{a}{\cdots\longrightarrow} x|\blacktriangleright^{\{I(a)\}}\ y''}$$

$$\frac{y \overset{a}{\cdots\longrightarrow} y''}{x|\blacktriangleright^{S}\ y \overset{a}{\cdots\longrightarrow} x|\blacktriangleright^{X\cup\{I(a)\}}\ y''}$$

$$\frac{x \overset{a}{\cdots\longrightarrow} x'',\ y \overset{a}{\cdots\longrightarrow} y''}{x \blacktriangleright y \overset{a}{\cdots\longrightarrow} x'' \mp |\blacktriangleright^{\{I(a)\}}\ y''}$$

$$\frac{x \overset{a}{\cdots\longrightarrow} x'',\ y \overset{a}{\not\cdots\longrightarrow}}{x \triangleright y \overset{a}{\cdots\longrightarrow} x''}$$

$$\frac{x \overset{a}{\cdots\longrightarrow} x'',\ y \overset{a}{\cdots\longrightarrow} y''}{x \triangleright y \overset{a}{\cdots\longrightarrow} x'' \triangleright y''} \qquad \frac{x \overset{a}{\cdots\longrightarrow} x'',\ y \overset{a}{\cdots\longrightarrow} y''}{x \triangleright^{S}\ y \overset{a}{\cdots\longrightarrow} x'' \triangleright^{S}\ y''}$$

4 Problems in the Semantics

One thing is obvious from the shown semantics: They are complicated. This is a bad point, as it increases the chance that people will write down an MSC that semantically has a meaning different from what they intend. Another point is that interrupt and disrupt may well not have certain desirable properties in their semantics, such as stratifiability [6]. This kind of properties may not cause worries to most users, but not having them will mean that the semantics are much harder to understand from a theoretical point of view.

We see a number of other problems as well, but many of those are, in some form or other, also present in the present semantics of MSC'96. Here we will mention a few.

First, because a message could be disrupted after it is sent but before it is received, when it is sent, we do not know whether it will be received or not. That way, an MSC could end its behaviour while there are still unreceived messages out.

Secondly, certain unwanted behaviours could enter an MSC in unexpected ways. Take for example the MSC in Figure 3. The choice whether or not to disrupt the transmission of m to transmit m' seems to be made by the instance to the right, but if the left one does local action a suddenly the right one cannot choose to do m' anymore. Thus a seemingly local choice [7] suddenly appears to be non-local. The same would be the case if the disrupt would be an interrupt, of course.

Another strange case is shown in Figure 4. There seems to be nothing strange about this MSC, but there is a hidden message overtaking here. Semantically in this MSC the message m can be overtaken by m', or even m' can be overtaken by m! Now, in some cases this might be logical, but in others it is very illogical, for example if the medium between the two instances works as a FIFO-buffer [8].

When disrupt and interrupt are combined with gates, we get into really bizarre situations. It is not clear what the semantics of an MSC like the one in Figure 5 are, or even what they should be.

In all these cases we run the risk that users write down an MSC which according to the official semantics has a meaning that differs from the intended one.

5 Conclusions

Interrupt and disrupt can be introduced into MSC in various ways. These choices have to be made very carefully, because a language construct that is not understood in the same way by all users and other people concerned will cause many more problems than it solves. Restriction in the inclusion of new features is advisable from a more general point of view too.

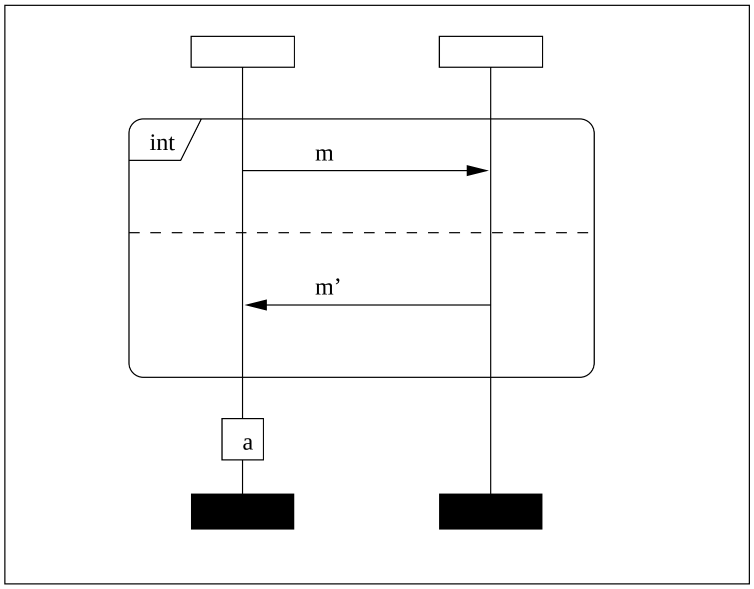


Figure 3: Example of unexpected non-local choice

If interrupt and disrupt are indeed to be included in the language, the first choice is whether a local or a global interrupt is taken. A local interrupt has the advantage of being semantically simple and easily understood, but on the other hand it does not really add anything to the language, so one could say it is nothing but syntactic sugar. A global interrupt on the other hand is semantically quite complicated, which can lead to unclarities or counter-intuitive results. There are also a number of additional choices to be made. Although adding interrupt and disrupt to MSC is certainly possible, their usefulness has to be doubted.

6 *

5 REFERENCES

[1] ITU-TS. Message Sequence Chart (MSC). Recommendation Z.120, ITU-TS, Geneva, May 1996.

[2] S. Mauw J.M.T. Cobben, A. Engels and M.A. Reniers. Formal semantics of message sequence charts. Technical Report CSR 97-17, Eindhoven University of Technology, 1998. to appear.

[3] ITU-TS. Algebraic semantic of Message Sequence Charts. Recommendation Z.120 Annex B, ITU-TS, Geneva, 1995.

[4] S. Mauw and M.A. Reniers. An algebraic semantics of basic Message Sequence Charts. *The Computer Journal*, 37(4):269–277, 1994.

[5] Th. Cobben and A. Engels. Disrupt and interrupt in MSC. internal document TD-L17, ITU, 1997.

[6] M.A. Reniers. (title to be announced). Master’s thesis, Eindhoven University of Technology, 1998? to appear.

[7] Hanêne Ben-Abdallah and Stefan Leue. Syntactic detection of process divergence and non-local choice in Message Sequence Charts. In Ed Brinksma, editor, *Tools and Algorithms for the Construction and Analysis of Systems*, number 1217 in Lecture Notes on Computer Science, pages 259–274. Springer Verlag, 1997.

[8] A. Engels, S. Mauw, and M.A. Reniers. A hierarchy of communication models for Message Sequence Charts. In T. Mizuno, N. Shiratori, T. Higashino, and A. Togashi, editors, *Formal Description Techniques and Protocol Specification, Testing and Verification, Proceedings of FORTE X and PSTV XVII ’97*, pages 75–90. Chapman & Hall, 1997.

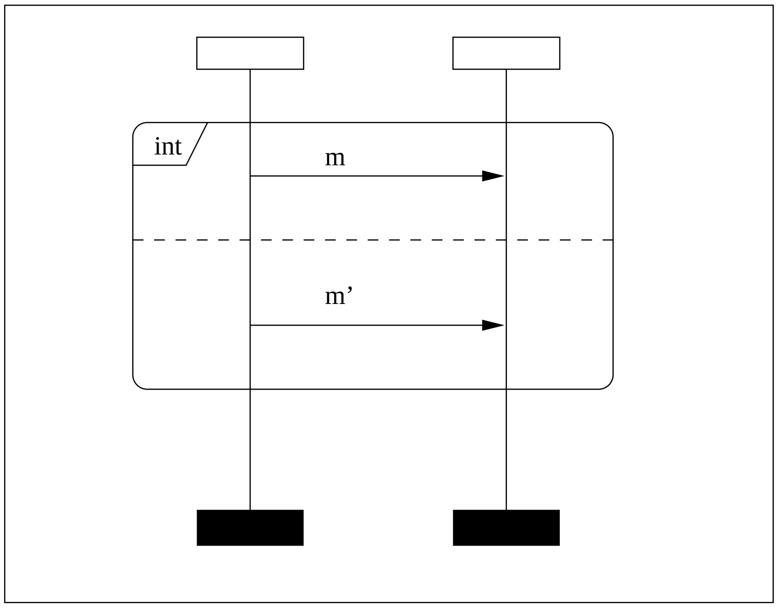


Figure 4: Example of unexpected message overtaking

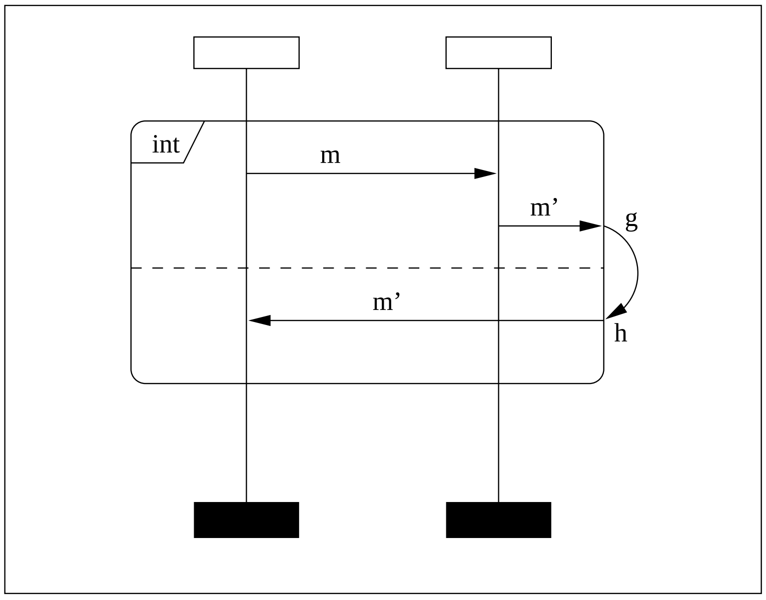


Figure 5: MSC with Interrupt and gates