Implementability of Message Sequence Charts*

F. Khendek[1], G. Robert [2], G. Butler[2] and P. Grogono[2]
[1]Department of Electrical and Computer Engineering
khendek@ece.concordia.ca
Tel. (514) 848 – 3081, Fax. (514) 848-2802
[2]Department of Computer Science
{gabriel | gregb | grogono}@cs.concordia.ca
Concordia University
1455, De Maisonneuve W., Montréal, Canada H3G 1M8

Abstract

In [1], we have introduced an approach for the generation of SDL process specifications from a basic MSC. The translation is constrained by the given architecture for the system. We extended our approach to handle inline constructs such as *alt, opt, seq, etc*. We came across the problem of non-implementability of MSCs for given architectures. This paper focuses on the implementability issue. We illustrate this problem with examples and discuss the notion of implementability of a MSC under a given communication architecture. We will also point out problems with the communication hierarchy introduced in [2] when full MSC'96 is taken into consideration.

Keywords

MSC, SDL, communication architecture, translation, implementability, compatibility

1 INTRODUCTION

MSC (Message Sequence Charts) [3, 4] are often used to capture user requirements and specify use cases. SDL (Specification and Description Language) [5] is used to describe abstract or detailed design with an explicit architecture of the system. In the software engineering lifecycle, the validation of the design against the requirements is mandatory. The verification of the design against the requirements can be greatly simplified if the SDL processes, or at least skeletons of these processes that represent the communications, have been generated automatically from the MSCs.

On the other hand, systems evolve with the addition of new capabilities that may also be described with MSCs. Typical examples of such evolving systems are the telecommunications systems that are very often extended with new services. In general, the existing system has a fixed architecture and the new service described with the MSCs is added without any modification to the system architecture.

In [1], we presented an algorithm for generating a skeleton SDL specification for a given bMSC (basic MSC). A bMSC consists of instances, messages and conditions. In our approach, the architecture of the SDL specification is given. Our approach can be used for the generation of an initial design from user requirements or for the incremental enrichment of an existing system design. We are extending our work to handle full MSC'96. We recently came across the problem of implementability of MSCs. Indeed, for a MSC, it may be impossible to

*This research was partly supported by the National Sciences and Engineering Research Council of Canada (NSERC) and Fonds pour la Formation des Chercheurs et l'Aide à la Recherche du Québec (FCAR).

translate it, under a given communication architecture, into its equivalent SDL specification. In other words, the MSC cannot be implemented with the given communication architecture.

In this paper, we discuss the notion of implementability of a MSC with a given communication architecture, and we point out problems with the communications hierarchy described in [2] when extended to full MSC'96. We use the terms "SDL architecture", "communication architecture" or "architecture" as synonyms.

In the next section, we review briefly our translation approach and we illustrate the non-implementability of MSCs with examples. In Section 3, we discuss the implementability concept and the communication hierarchy introduced in [2]. In Section 4, we discuss briefly an extension to our MSC to SDL translation algorithm, in order to handle the implementability issue. We conclude in Section 5.

2 FROM MSC TO SDL

A MSC specifies the required order of sending and consuming of messages, but not the actual arrival order into the input queues of the processes. The actual arrival order depends on the communication architecture of the system and the interleaving of the processes. MSCs do not specify the communication architecture. To translate a MSC into an SDL specification, the architecture of the target SDL system is required. The algorithm we have introduced in [1], takes as input a MSC and a communication architecture, and generates the behavior specification for the SDL processes.

For the description of the communication architecture, we use an explicit SDL specification. Processes may be grouped into blocks. Processes communicate through signalroutes or channels. The data part is given with the SDL specification. Prior to the translation, the algorithm checks for the "architectural consistency" between the MSC and the given communication architecture, i.e. there is a one to one correspondence between the MSC processes and a subset of the processes in the SDL architecture and a one to one correspondence between the MSC messages and a subset of the SDL signals [1]. The given architecture may have more processes and signals, which are introduced by other MSCs.

A MSC describes a "global" view of the system, whereas an SDL specification is distributed by nature. With the SDL system, signals travel through different channels or signalroutes and processes have different speeds. A signal "*a*" that is supposed to be received before a signal "*b*" may arrive into the destination process input queue later than "b". During the translation of a MSC into an SDL specification with a given architecture, we have to take into account the actual arrival order of signals into the input queue of each process. The algorithm avoids discarding from the input queue signals that are expected in later stages. These signals are saved using the SDL *save* concept [5].

Our translation approach ensures, by construction, behavioral consistency between the SDL specification and the MSC specification. The SDL specification is also free from deadlocks and unspecified receptions; no further validation is required. When multiple MSCs are taken as input, the consistency relationship between the whole set of MSCs and the SDL specification is the trace equivalence. In other words, the set of traces defined by the set of MSCs is equal to the set of traces of the SDL specification. Therefore, the set of traces of each MSC is a subset of the set of traces of the SDL specification.

We have extended our approach to handle MSC'96 inline constructs, such as *alt, seq, opt,* etc. An example of MSC with inline construct is given in Figure 1.a. MSC Ex1 describes two alternate behaviors, either *Sender* sends message "*a*", *Receiver* receives "*a*" and replies with message "*c*"; or *Sender* sends message "*b*" and *Receiver* receives "*b*" and replies in this case with message "*d*". In other words, *Receiver* will reply with "*c*" if it receives "*a*" or with "*d*" if it receives "*b*". For the generation of an equivalent SDL specification, the target architecture is given in Figure 1.b. In this architecture, messages "*a*" and "*b*" are sent from *Sender* to *Receiver* through one channel and messages "*c*" and "*d*" are sent from *Receiver* to *Sender* through a second channel.

The generated SDL processes are shown in Figure 2. As specified by the MSC in Figure 1, the sending of messages "*a*" or "*b*" are two exclusive events for *Sender*, therefore only "*a*" or "*b*" will be send and queued in *Receiver* input queue. For the generation of the skeleton process *Receiver*, we expect only one of these two signals; therefore we do not save the other signal.

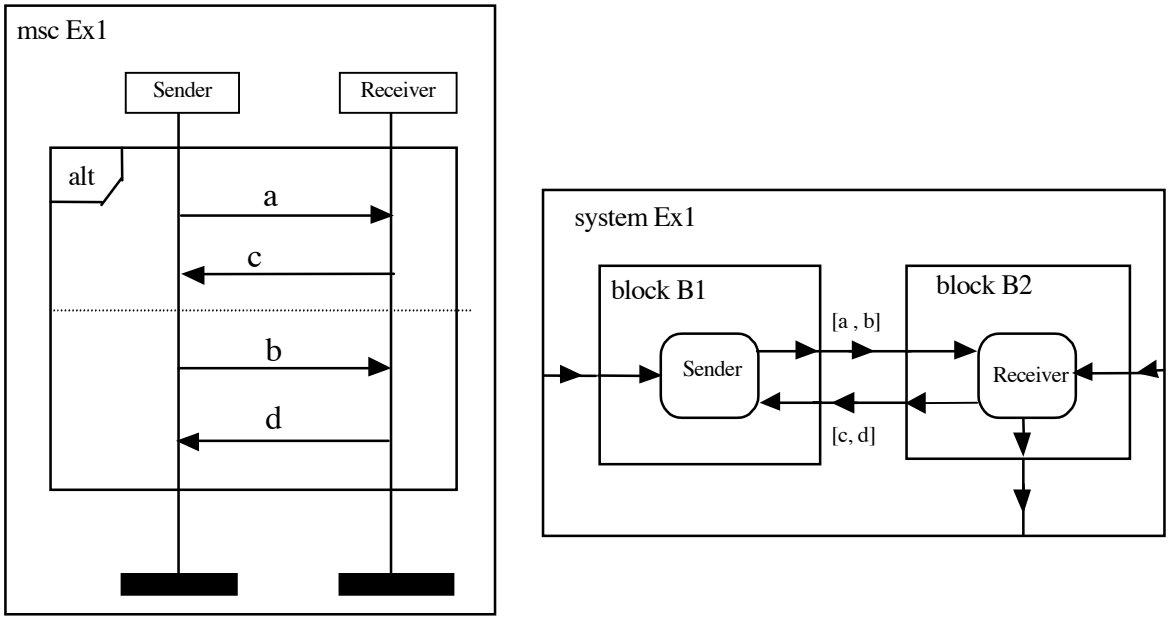

Figure 1. **(a)** MSC with alt construct. **(b)** SDL Architecture

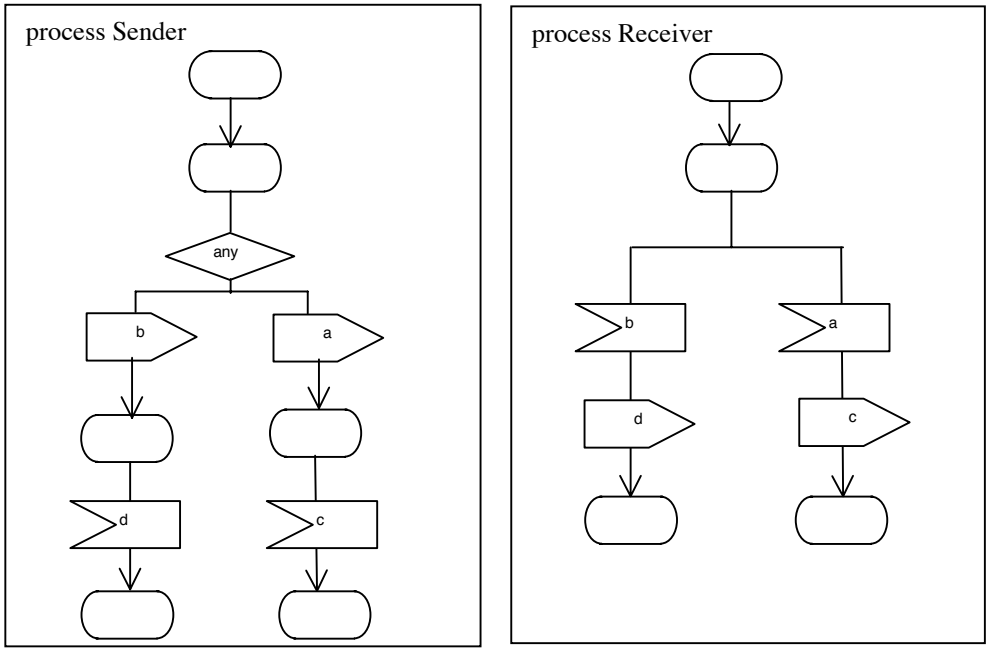

Figure 2. Generated SDL processes for the example in Figure 1.

The *MSC Ex2* in Figure 3.a is a second illustration for the *alt* construct. Here again, we have two processes, *Sender* and *Receiver*. The MSC describes two alternate behaviors, either *Sender* sends message "*a*" then message "*b*", and *Receiver* receives "*a*" then "*b*", and sends message "*c*" or *Sender* sends message "*b*" then

message "*a*" and *Receiver* receives "*b*" then "*a*", but in this case the receiver sends message "*d*". In other words, *Receiver* will reply with "*c*" if it receives sequence "*a.b*" or with "*d*" if it receives sequence "*b.a*". The labeled transition system in Figure 4 represents the set of traces specified by this MSC. The notation "*-x*" stands for sending of message "*x*", whereas "*+y*" stands for the consumption of message "*y*".

Our initial set of MSCs consists of the unique MSC given in Figure 3.a. The SDL architecture given in Figure 3.b, specifies that signals "*a*" and "*b*" are conveyed from *Sender* to *Receiver* through two different channels, whereas signals "*c*" and "*d*" are conveyed from *Receiver* to *Sender* through the same channel. Both channels are delaying channels. The translation of MSC Ex2 into an equivalent SDL specification with the architecture given in Figure 3.b is impossible. In other words, we cannot, for the given architecture, find an SDL specification that has exactly the same set of traces as MSC Ex2. Because of this architecture, the message "*a*" and "*b*" may be delivered to *Receiver* in the reverse order of their transmission. These two messages being conveyed through two different delaying channels, both arrival orders are possible independently of the transmission order. Process *Receiver* cannot distinguish which branch has been taken by *Sender* and how it should reply. Since both arrival orders are possible and exclusive, we cannot solve this problem with use of *save*. The same problem can be described using local or global conditions instead of process *Receiver* replying with "*c*" or "*d*".

Notice that MSC Ex2 is implementable with the architecture in Figure 1.b. SDL processes, very similar to the processes in Figure 2; will be generated by the translation algorithm. MSC Ex1 is implementable under both architectures, Figure 1.b and Figure 3.b, and the algorithm generates exactly the same SDL processes in both cases.

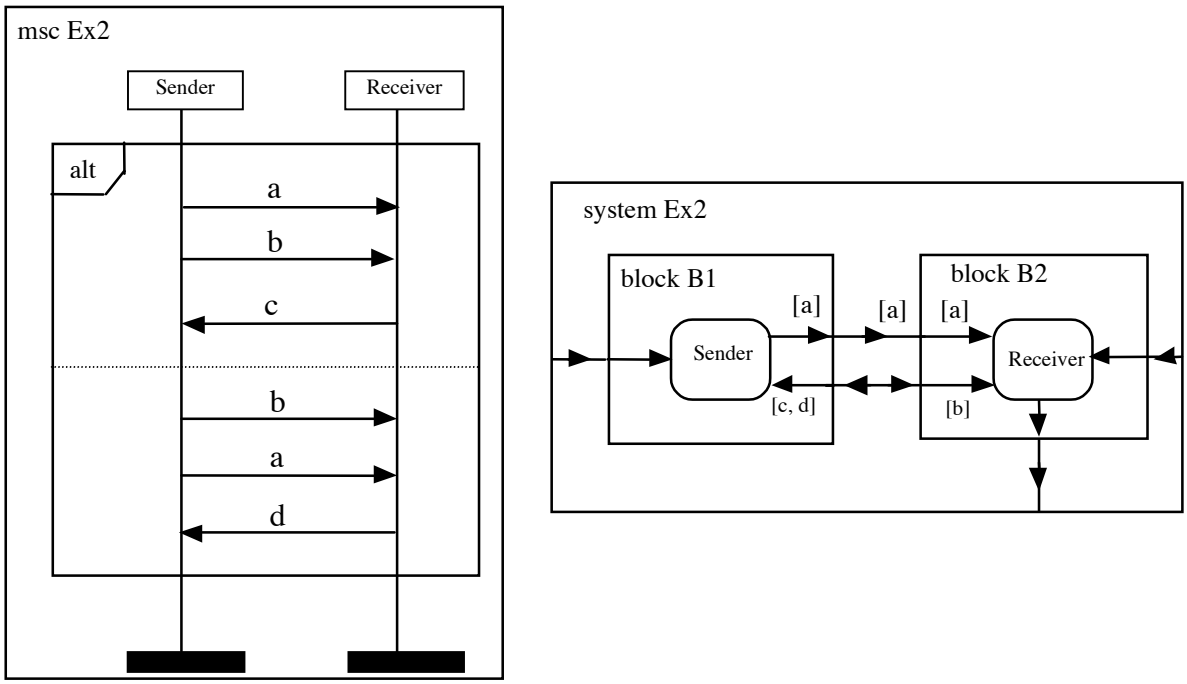

Figure 3. (a) MSC non-implementable with (b). **(b)** SDL Architecture

This problem with the *alt* construct can be characterized as follows: for at least one process, at least two alternative traces with the same messages in different orders are allowed, and the communication architecture does allow for this process to distinguish between these two alternatives.

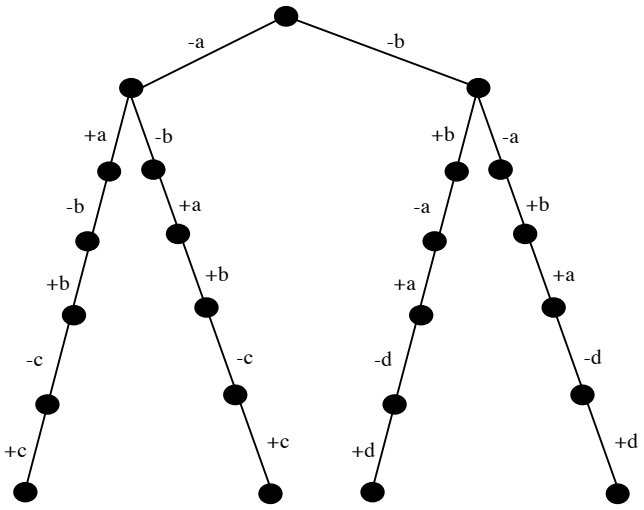

Figure 4. Traces specified by the MSC in Figure 3.a.

Now, consider the MSC Ex3 given in Figure 5 with the architecture given in Figure 3.b. Even if the messages sent and received in both alternatives of the *alt* expression are different, *Receiver* cannot figure out which alternative has been taken by *Sender* when signal "*a*" and "*b*" are both in its input queue. Here again, MSC Ex3 cannot be implemented with the given architecture. Similar problems may arise with *opt* expressions.

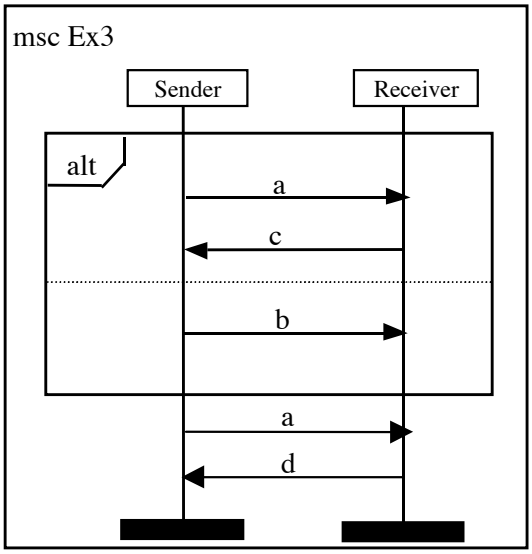

Figure 5. A second case of non-implementability.

Notice that this is not a "distributed choice" as described in [6], the choice of sending "*a*" then "*b*" or "*b*" then "*a*" is made locally by *Sender*. However, the same problem arises with the distributed choice in the translation from MSC to SDL. A MSC that contains a distributed choice can be implemented with only full synchronization architectures, where the sending and the reception of a message are simultaneous. This is due to the natures of MSC and SDL specifications: the first one gives a "global" view of the system, whereas the second one is distributed in general and processes communicate through channels and signalroutes. The same problem of "distributed choice" is encountered in the derivation of communication protocol specifications from service specification [7].

3 IMPLEMENTABILITY OF MSCs

As shown in previous section, a MSC is not always implementable. In [2], the authors developed a hierarchy of communication models under which a bMSC can be implemented. This communication model hierarchy goes from a model ("msg") where each message travels through its own buffer (i.e. channel), to a model ("global") where all messages for all instances travel through the same channel. The highest model in the hierarchy "nobuf" has no buffer and implements synchronous communications. Between the "global" and "msg" models, there exist different models: the "pair" model refers to the communication model where all the signals between each pair of processes goes through two channels, one for each direction. According to this hierarchy, any bMSC that is implementable should be "msg"-implementable.

Our example in Figure 3.a shows clearly that the hierarchy in [2] does not hold for full MSC. The MSC in Figure 3.a is "generally implementable". However, it is not "msg"-implementable. As explained previously, when each message is conveyed through its own channel, *Sender* can send sequence "*a.b*", and expect to receive "*c*" or can send sequence "*b.a*", and expect to receive "*d*". This behavior can be implemented only if the sending order of the messages is preserved, which is not the case with the "msg"model. On the other hand, the MSC is "pair" implementable. As mentioned previously, the MSC Ex2 can be implemented with the architecture in Figure 1.b. Therefore, the hierarchy in [2] (any "pair"-implementable message should be "msg"--implementable) cannot be extended to full MSCs.

The problem is that the hierarchy developed considers only the fact that the architecture should allow all MSC allowed traces (strong implementability) or at least one (weak implementability) to be implemented, but does not consider that fact that the architecture should also preserve "enough" of the ordering of messages to ensure that various branches of the MSC remain distinguishable.

We suggest that the hierarchy could be transformed such that a MSC would be implementable in a "band" of architectures. For example, a MSC could be implementable in any architecture lower than "xxx" and higher than "zzz":

xxx: Architecture does not allow the traces
yyy: The MSC is (weakly/strongly) implementable under the architecture
zzz: Architecture allows too many traces that will make branches of the MSC indistinguishable

The MSC in Figure 6.a shows clearly that it cannot be implemented with neither the "nobuf" model nor with the "msg" model. However, there are many architectures in between in which the MSC can be implemented. Messages "*x*" and "*y*" have to be conveyed through different channels to allow for overtaking and the communication model for the messages in the *alt* construct should allow for distinguishing between the two alternatives, like the architecture given in Figure 1.b for instance. However, the MSC in Figure 6.b cannot be implemented at all. The first part requires an "msg" model whereas the second part cannot be implemented with an "msg" model.

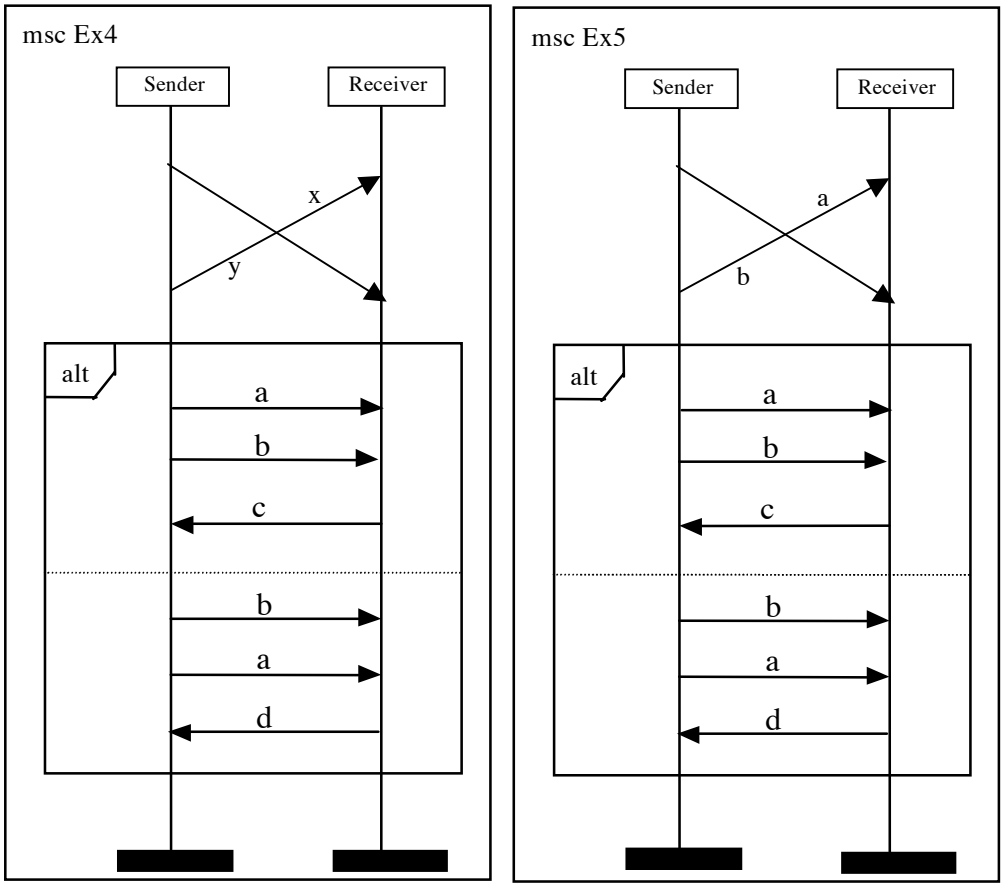

Figure 6. (a) An MSC implementable in a restricted "band". **(b)** Non-implementable MSC.

4 EXTENSION OF THE MSC TO SDL TRANSLATION ALGORITHM

In this section, we discuss extensions to our translation algorithm to handle the implementability issue. The translation task is further complicated by the fact that various messages may be conveyed through different channels, i.e.: the SDL architecture of the processes may be a composite of many of the basic architectures described in [2].

The algorithm starts by mapping the given SDL architecture to a set of communication models. Then, each message can be mapped into a communication model (we assume that each message travels through a unique channel type). The algorithm then converts the MSC into one tree per instance. Then, it determines the message sequences that distinguish each branch of the tree (in the above case: "*a.b*" and "*b.a*"). Finally, the algorithm determines, from the set of communication models used for the messages composing the sequence, whether a branch can contain a sequence that is in another sibling branch. If so, the MSC is non-implementable under the given architecture and the error is reported back to the system designer.

Other algorithms can also used to determine whether the MSC is not implementable because the architecture is insufficient (similar to what is described in [2]. A simple example is a MSC with message overtaking which can not be implemented through a "global" model (all channels are assumed FIFO).

The problem of non-implementability is driving us toward the problem of compatibility between MSCs. Indeed, if you decompose MSC Ex5 in Figure 6.b into the component MSCs as shown in Figure 7. We know from Section 3 that MSC Ex6 can be implemented with "msg" model only while the MSC Ex7 can be implemented with the "pair" model only. There is no common architecture in which both MSCs can be implemented. Therefore, we can conclude they are incompatible. This is an example of "absolute" incompatibility. The

"relative" compatibility is defined relatively to a given architecture or a set of architectures. Notice that these definitions of compatibility are related to the implementability, other definitions of compatibility may be given for other purposes.

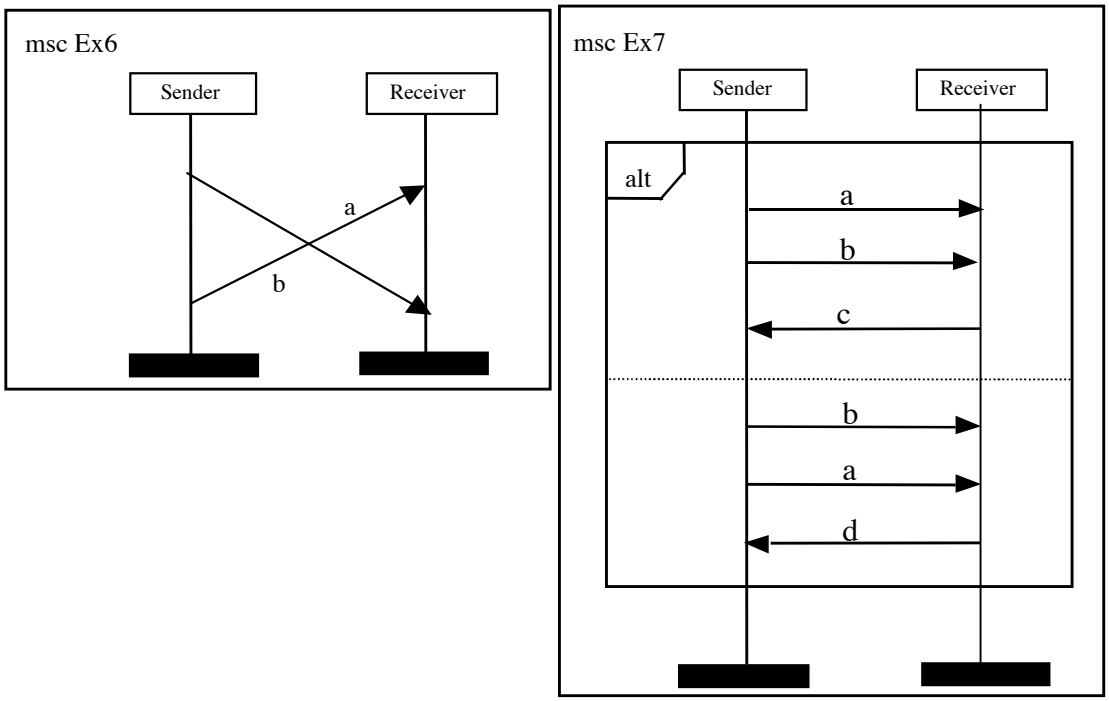

Figure 7. Incompatibility between MSCs.

5 CONCLUSION

The objective of our work is the automatic generation of correct SDL specification, from MSC specifications and incremental enrichment of SDL specifications with behaviors specified with MSCs. We assume the target architecture is fixed. This differs significantly from [8], which reports on race conditions.

We are extending our basic approach to handle MSC'96. Our algorithm generates automatically the SDL process specifications when the MSC is implementable in the given architecture. We came across this problem of non-implementabilty, which is due to the distributed nature of SDL specifications compared to the "global view" described by a MSC. We know that some MSCs cannot be implemented with some architecture and other MSCs are not implementable at all. These non-implementable MSCs are generally composed of incompatible bMSCs.

Engels et al. [2] have also considered the implementability of bMSCs under different communication models and came out with a hierarchy for a bMSC. In our work, we have shown that this hierarchy does not hold for full MSC'96 as discussed in sections 2 and 3. Our work on translating MSCs to SDL, implementability of MSCs, and compatibility between MSCs is still in progress.

Acknowledgment
The authors thank Mohamed Ashour and Mohamed M. Abdalla for many insightful discussions.

REFERENCES

[1] G. Robert, F. Khendek and P. Grogono, “Deriving an SDL Specification with a Given Architecture from a Set of MSCs”, in A. Cavalli and A. Sarma (eds.), SDL'97: Time for Testing - SDL, MSC and Trends, Proceedings of the eight SDL Forum, Evry, France, Sept. 22 - 26, 1997.

[2] A. Engels, S. Mauw and M. A. Reniers, “A Hierarchy of Communication Models for Message Sequence Charts”, Proceedings of the IFIP International Symposium on Protocol Specification Testing and Verification (PSTV), Osaka, Japan, Nov. 1997.

[3] ITU-T, “Recommendation Z.120 - Message Sequence Chart (MSC)”, 1996.

[4] E. Rudolf, J. Grabowski and P. Graubmann, “Tutorial on Message Sequence Charts (MSC’96)”, Tutorial of the FORTE/PSTV’96 conference in Kaiserslautern, Germany, Oct. 1996.

[5] ITU-T, “Recommendation Z.100-Specification and Description Language (SDL)”, 1993.

[6] H. Ben-Abdallah and S. Leue, “Syntactic Analysis of Message Sequence Chart Specifications”, Technical Report 96-12, University of Waterloo, Electrical and Computer Engineering, Nov. 1996.

[7] F. Khendek, G. v. Bochmann and C. Kant, “New Results on deriving protocol specifications from service specifications”, in Communication Architectures and Protocols: ACM SIGCOMM’89, pp. 136-145, Austin, Texas, Sept. 1989.

[8] R. Alur and G. J. Holzmann and D. Peled, “An analyzer for Message Sequence Charts”, Proceedings of Tools and Algorithms for the Construction and Analysis of Systems: second international workshop; proceedings (TACAS '96), pp. 35-48, Springer-Verlag, 1996.