I n t e r n a t i o n a l   T e l e c o m m u n i c a t i o n   U n i o n

ITU-T

TELECOMMUNICATION
STANDARDIZATION SECTOR
OF ITU

Z.120

Amendment 2
(09/2009)

SERIES Z: LANGUAGES AND GENERAL SOFTWARE ASPECTS FOR TELECOMMUNICATION SYSTEMS

Formal description techniques (FDT) – Message Sequence Chart (MSC)

Message sequence chart (MSC)

Amendment 2: Revised Appendix I – Application of MSC

Recommendation ITU-T Z.120 (2004) – Amendment 2

ITU-T Z-SERIES RECOMMENDATIONS

LANGUAGES AND GENERAL SOFTWARE ASPECTS FOR TELECOMMUNICATION SYSTEMS

FORMAL DESCRIPTION TECHNIQUES (FDT)	
Specification and Description Language (SDL)	Z.100–Z.109
Application of formal description techniques	Z.110–Z.119
Message Sequence Chart (MSC)	**Z.120–Z.129**
User Requirements Notation (URN)	Z.150–Z.159
Testing and Test Control Notation (TTCN)	Z.160–Z.179
PROGRAMMING LANGUAGES	
CHILL: The ITU-T high level language	Z.200–Z.209
MAN-MACHINE LANGUAGE	
General principles	Z.300–Z.309
Basic syntax and dialogue procedures	Z.310–Z.319
Extended MML for visual display terminals	Z.320–Z.329
Specification of the man-machine interface	Z.330–Z.349
Data-oriented human-machine interfaces	Z.350–Z.359
Human-machine interfaces for the management of telecommunications networks	Z.360–Z.379
QUALITY	
Quality of telecommunication software	Z.400–Z.409
Quality aspects of protocol-related Recommendations	Z.450–Z.459
METHODS	
Methods for validation and testing	Z.500–Z.519
MIDDLEWARE	
Processing environment architectures	Z.600–Z.609

For further details, please refer to the list of ITU-T Recommendations.

Recommendation ITU-T Z.120

Message sequence chart (MSC)

Amendment 2

Revised Appendix I – Application of MSC

Summary

Amendment 2 to Recommendation ITU-T Z.120 addresses the problem of possible applications of message sequence charts (MSCs). These application domains are not explicitly defined in the main part of this Recommendation, which leaves ground for the following interpretation that MSCs can be used in almost any context, without restriction. This is not the case, and this appendix clarifies several interpretation issues related to the verification and implementability of MSCs, and shows some syntactic requirements needed for each of these applications.

Amendment 2 cancels and replaces Amendment 1 (2008) to Recommendation ITU-T Z.120 (2004).

Source

Amendment 2 to Recommendation ITU-T Z.120 (2004) was agreed on 25 September 2009 by ITU-T Study Group 17 (2009-2012).

Keywords

Implementability, message sequence charts, model-checking, requirements.

FOREWORD

The International Telecommunication Union (ITU) is the United Nations specialized agency in the field of telecommunications, information and communication technologies (ICTs). The ITU Telecommunication Standardization Sector (ITU-T) is a permanent organ of ITU. ITU-T is responsible for studying technical, operating and tariff questions and issuing Recommendations on them with a view to standardizing telecommunications on a worldwide basis.

The World Telecommunication Standardization Assembly (WTSA), which meets every four years, establishes the topics for study by the ITU-T study groups which, in turn, produce Recommendations on these topics.

The approval of ITU-T Recommendations is covered by the procedure laid down in WTSA Resolution 1.

In some areas of information technology which fall within ITU-T's purview, the necessary standards are prepared on a collaborative basis with ISO and IEC.

NOTE

In this Recommendation, the expression "Administration" is used for conciseness to indicate both a telecommunication administration and a recognized operating agency.

Compliance with this Recommendation is voluntary. However, the Recommendation may contain certain mandatory provisions (to ensure e.g., interoperability or applicability) and compliance with the Recommendation is achieved when all of these mandatory provisions are met. The words "shall" or some other obligatory language such as "must" and the negative equivalents are used to express requirements. The use of such words does not suggest that compliance with the Recommendation is required of any party.

INTELLECTUAL PROPERTY RIGHTS

ITU draws attention to the possibility that the practice or implementation of this Recommendation may involve the use of a claimed Intellectual Property Right. ITU takes no position concerning the evidence, validity or applicability of claimed Intellectual Property Rights, whether asserted by ITU members or others outside of the Recommendation development process.

As of the date of approval of this Recommendation, ITU had not received notice of intellectual property, protected by patents, which may be required to implement this Recommendation. However, implementers are cautioned that this may not represent the latest information and are therefore strongly urged to consult the TSB patent database at http://www.itu.int/ITU-T/ipr/.

CONTENTS

Recommendation ITU-T Z.120

Message sequence chart (MSC)

Amendment 2

Revised Appendix I – Application of MSC

(This appendix does not form an integral part of this Recommendation)

I.1 Introduction

This appendix addresses the problem of possible applications of message sequence charts (MSCs). The main part of this Recommendation states that MSCs are meant to "describe interactions between a number of independent message passing Instances". In addition to this, MSCs are "a scenario language", "a graphical language", "a formal language", and are "widely applicable", that is not "tailored for one single application domain". These application domains are not explicitly defined in the main part of this Recommendation, which leaves ground for the following interpretation that MSCs can be used in almost any context, without restriction. This is not the case, and recent literature has shown that with their whole expressive power, several applications of MSCs were impracticable.

Among the applications of MSC, the following are frequently addressed:

- model checking,
- comparison of specifications,
- specification and implementation.

This list is not exhaustive, but has been well covered by the literature in the last decade. In particular, several publications have shown that these applications can be undecidable problems for MSCs, that is there exists no algorithm that takes as an input any MSC and that terminates with as output a correct solution. The objective of this appendix is to provide a list of known decision problems that are impracticable in general for MSCs, and a list of syntactic criteria that ensure decidability of some of the problems listed.

This appendix is organized as follows. Clause I.2 gives a more precise definition of the model checking or comparison problems that can be considered for MSCs, and of the notion of implementation of MSCs. Clause I.4 identifies syntactic subclasses of MSCs called regular MSCs, local choice MSCs and globally cooperative MSCs for which model checking, comparison, or implementation problems have a solution.

I.2 Problems

I.2.1 Model checking

The usual definition of model checking is verifying whether a logic formula ϕ, described with a specific syntax, is satisfied by a model M. This is written $M \models \phi$. Several popular logics exist. We can cite linear temporal logic (LTL), computational tree logic (CTL), CTL*, alternating-time temporal logic (ATL), and the modal μ-calculus. For an introduction to model checking and logics, interested readers may consult [b-Clarke99], and [b-Holzmann99].

Temporal logics are frequently used to ensure that modelled systems satisfy some safety or liveness properties. Logics can address properties of global states, or question the structure of the model itself, and the interpretation of a formula ϕ depends on the semantics of the logic. Similarly, the usual interpretation of model checking is that the formulae address properties of runs and global states of the model.

An example of linear temporal logic (LTL) formula is:

$$\phi_1 = \mathbf{G}(a \Rightarrow \mathbf{F}\ b)$$

With the LTL semantics, if *a* and *b* are action names, this property means that it is always true that when action *a* is played, then action *b* is eventually played. A whole description of LTL and other logics is beyond the scope of this appendix.

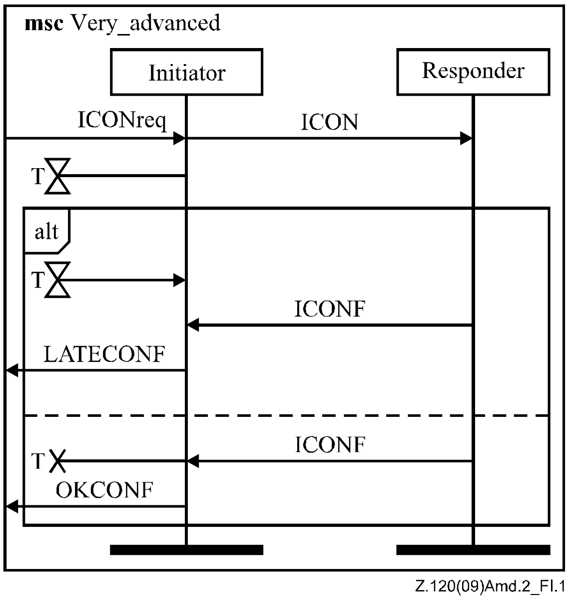


Figure I.1 – An MSC description with an alternative

For message sequence charts, the runs of a description are defined by the semantics provided by [b-ITU-T Z.120 Annex B]. For a given MSC description *M*, we will denote by *L(M)* the set of all runs defined by *M*. Note also that an MSC description does not only represent a set of runs, but also a set of basic MSCs, which can be obtained by unfolding loops, replacing alternatives by a single choice, etc. Consider, for instance, the MSC description of Figure I.1. This description defines two possible MSCs that are depicted in Figure I.2.

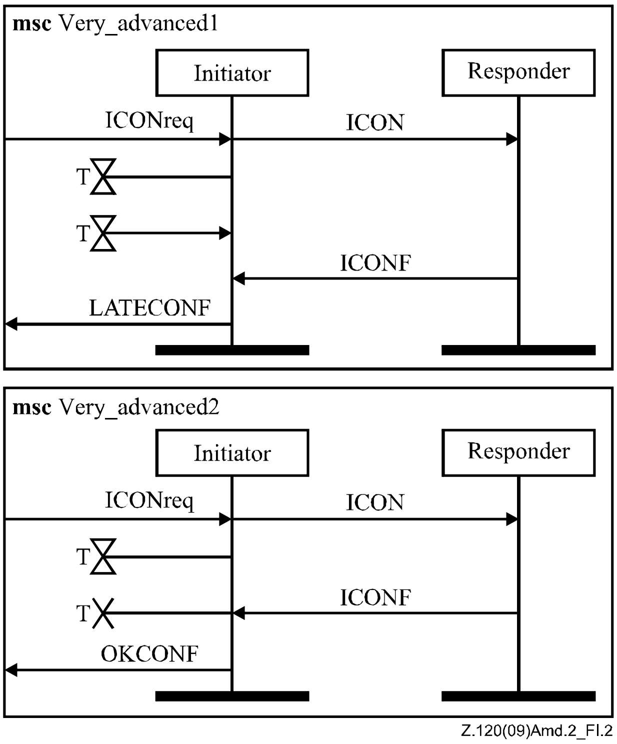


Figure I.2 – Basic MSC interpretation of the MSC in Figure I.1

From now on, we will denote by *F(M)* the set of basic MSCs (bMSCs) described by an MSC description *M*.

Frequently, checking a logical formula over runs of a model *M* is equivalent to verifying joint properties of runs of the model and of a finite state automaton R_ϕ computed from the formula. For instance, an automaton $R_{\neg\phi 1}$ associated to the negation of formula ϕ_1 above is depicted in Figure I.3. This automaton describes all runs that do not satisfy ϕ_1. Checking whether a MSC *M* satisfies ϕ_1 consists in verifying that the set of runs described by *M* and by the automaton $R_{\neg\phi 1}$ are disjoint.

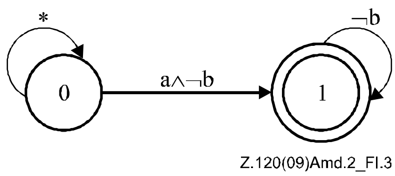


Figure I.3 – An automaton $R_{\neg\phi 1}$ collecting runs that satisfy $\neg\phi_1$

To model-check logical properties on message sequence charts specifications, tools have to provide an answer to a question of the kind:

(Mc1)	$L(M) \subseteq R_\phi$?
(Mc2)	$R_\phi \subseteq L(M)$?
(Mc3)	$L(M) \cap R_\phi = \phi$?

Where *M* is the MSC description, ϕ a logical formula, R_ϕ a finite state automaton that describes sets of runs that satisfy (or do not satisfy) ϕ. Here, *Mc*1 occurs when R_ϕ models all acceptable behaviours: the behaviours of the MSC specification should be contained in the behaviours of R_ϕ and the user wants a positive answer. *Mc*2 occurs when R_ϕ models the bad properties of all behaviours that should not occur, and the user expects a negative answer. *Mc*3 occurs when R_ϕ

models behaviours that have some undesired property, and the user expects a negative answer. Within this setting, comparison of the runs of an MSC with a finite state automaton should be a tractable problem.

I.2.2 Comparison of MSC descriptions

Comparison of MSC descriptions is another problem close to model checking. As there might be several ways to describe similar behaviours with message sequence charts, the questions of the equivalence of two descriptions might be interesting. For two MSC descriptions $M1$ and $M2$, one may also want to verify that $M2$ is an extension of $M1$, i.e., that all behaviours described by $M1$ can be found in $M2$. This comparison can occur at the level of runs, or at the level of basic MSCs generated by two MSC specifications. Hence, comparing two MSC specifications $M1$ and $M2$ resumes to answering any of the six following questions:

(EquivL) $L(M1) = L(M2)$?

(RefL) $L(M1) \subseteq L(M2)$?

(IntL) $L(M1) \cap L(M2)= \phi$?

(EquivF) $F(M1) = F(M2)$?

(RefF) $F(M1) \subseteq F(M2)$?

(IntF) $F(M1) \cap F(M2) = \phi$?

I.2.3 Specification and implementation

Message sequence charts allow for the description of interactions. It is then tempting to consider them as a specification or even as a development and programming language. However, not all MSC descriptions can be implemented. Consider, for instance, the example of Figure I.9. In this MSC, two systems are performing actions, namely *count* for instance *System*1, and *recount* for instance *System*2. Implicitly, in all bMSCs depicted by this description, the number of occurrences of actions *count* and *recount* executed by both instances should be the same. However, the two systems never communicate. Hence, without providing additional mechanisms to synchronize *System*1 and *System*2, the description of Figure I.9 cannot be implemented.

If a user considers that message sequence charts present behaviours at a certain abstraction level, this kind of description is not a real problem, as using additional messages in implementations of this description is allowed. Now, if the MSC description is considered as complete, i.e., all message actions, timers and so on that will be used by any implementation appear in the description, then some MSC descriptions cannot be implemented.

A general approach to implement a MSC description is to implement the behaviour of each instance separately (for instance with SDL) [b-Khendek99]. However, it has been shown that not all MSC descriptions can be implemented this way [b-Alur05], as some additional unspecified behaviours appear in the generated implementation. When a MSC description can be implemented by separating all instances behaviours, it will be called a *realizable* MSC.

Hence, a natural question that arises for a given MSC description *M* is:

(Rez) is M realizable ?

In general, there is no procedure to answer the realizability question [b-Alur05]. However, recent results [b-Genest02], [b-Genest04], and [b-Helouet00] have shown that a slight modification of the message contents can allow for the implementation of some subclasses of MSCs.

I.3 General undecidable results

A problem whose answer can be yes or no is called *decidable* when there exists an algorithm that can compute a correct answer for any instance of this problem. If such algorithm does not exist, the problem is said to be *undecidable*.

(Mc1)	$L(M) \subseteq R_\phi$?
(Mc2)	$R_\phi \subseteq L(M)$?
(Mc3)	$L(M) \cap R_\phi = \phi$?
(EquivL)	$L(M1) = L(M2)$?
(RefL)	$L(M1) \subseteq L(M2)$?
(IntL)	$L(M1) \cap L(M2) = \phi$?
(EquivF)	$F(M1) = F(M2)$?
(RefF)	$F(M1) \subseteq F(M2)$?
(IntF)	$F(M1) \cap F(M2) = \phi$?
(Rez)	*is M realizable ?*

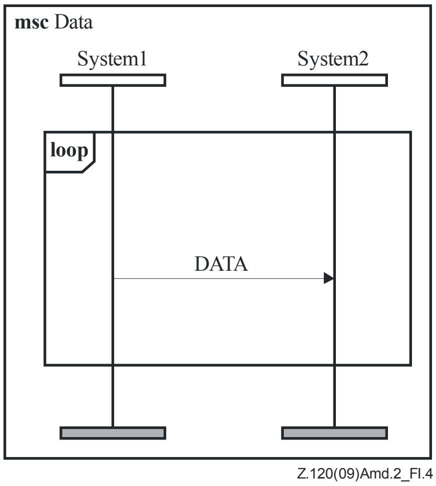
are undecidable problems. This does not mean that model checking, comparison or implementability of MSCs are always untractable problems for MSCs (i.e., they have no algorithmic solution), but rather that when considering a target application for an MSC description, users have to make sure that their specification meets some syntactical requirements. Some simple syntactic criteria allow for the characterization of several kinds of MSC descriptions (or MSC subclasses) that enable the decidability of some of the problems listed above.

I.4 Syntactical description of MSC subclasses

Due to the undecidability results cited in clause I.3, several applications could be considered as impossible for message sequence charts. Several restrictions to the use of MSC constructs have been defined. This clause lists three of them, and for each syntactical subclass lists the possible applications.

I.4.1 Regular MSCs (RMSCs)

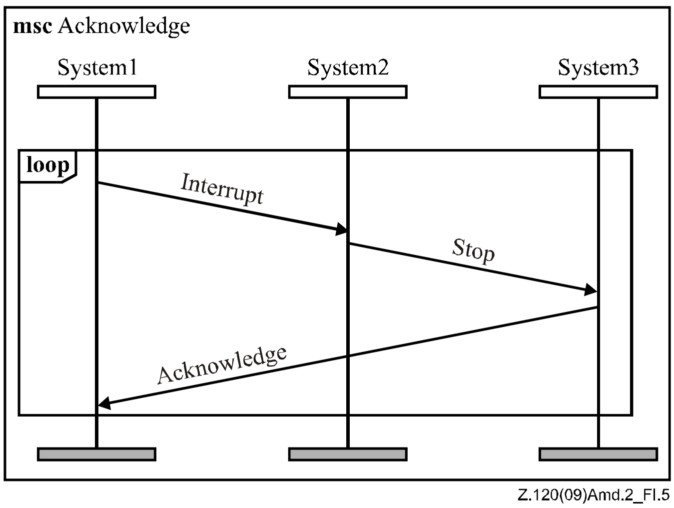
The set of runs of a regular MSC forms a regular language. This means that this set can be represented by a finite state machine, but also that usual techniques of model checking can be applied to regular MSC. An MSC description forms a regular MSC description if, in all loops that can appear in the description, all messages that are sent are acknowledged, either directly or indirectly, and if the body of the loop does not form disconnected parts of behaviours.




```
msc Data;
inst System1, System2;
  instance System1;
    loop begin simpleloop shared all;
      out DATA to System2;
    loop end;
  endinstance;
  instance System2;
    loop begin simpleloop shared all;
      in DATA from System1;
    loop end;
  endinstance;
endmsc;
```

Figure I.4 – A non-regular MSC

Consider, for instance, the MSC description of Figure I.4. In message sequence charts, messages are considered asynchronous. This specification then describes a protocol where instance *System*1 does not have to wait for an acknowledgement of *DATA* messages before sending the next message. Runs of this MSC cannot be depicted by a finite state automaton. The second condition is illustrated by the MSC *Count* of Figure I.9. In this MSC description, all MSCs in *F(Count)* contain the same number of occurrences of atomic actions *count* and *recount*. The runs of this MSC cannot be described with a finite state automaton. The MSC Acknowledge of Figure I.5 fulfils the conditions to be regular.




```
msc Acknowledge;
inst System1, System2, System3;
  instance System1;
    loop begin simpleloop shared all;
      out Interrupt to System2;
      in Ack from System3;
    loop end;
  endinstance;
  instance System2;
    loop begin simpleloop shared all;
      in Interrupt from System1;
      out Stop to System3;
    loop end;
  endinstance;
  instance System3;
    loop begin simpleloop shared all;
      in Stop from System2;
      out Ack to System1;
    loop end;
  endinstance;
endmsc;
```

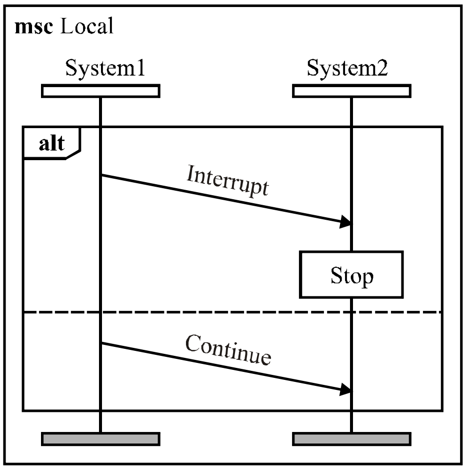
Figure I.5 – A regular MSC

To summarize, the following questions might be solved by appropriate algorithms for the class of regular MSCs: *Mc*1*, Mc*2*, Mc*3*, EquivL, RefL, IntL.*

I.4.2 Local choice MSCs (LMSCs)

An MSC description is called a local choice MSC when, for all alternatives, there is one single instance which can take the decision of how to continue interactions with other instances.

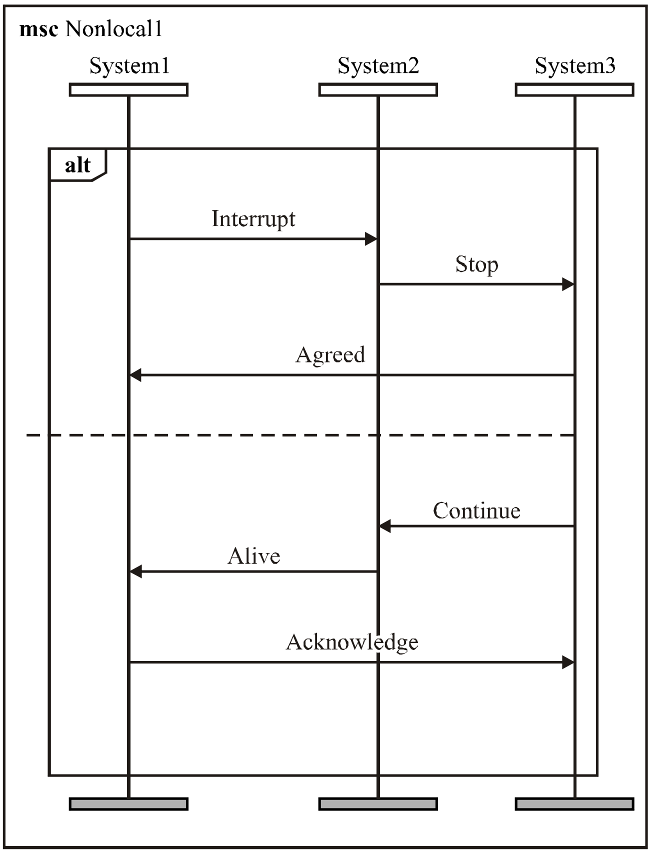
Consider the example of Figure I.6. MSC *Local* is a local choice MSC, as for both behaviours in the alternative, instance *System*1 chooses how the interaction will continue, by sending a message to *System*2. MSC *Nonlocal* in Figure I.7 is not a local choice MSC, as the decision to perform one of the alternative scenarios can be taken either by *System*1 or by *System*3. There is a chance that an implementation of such a scenario leads to a deadlock. A deadlock is a situation where two or more processes are waiting for each other to continue their execution. For the MSC description of Figure I.7, if the program implementing *System*1 behaves as in the first part of the alternative, and the program implementing *System*3 behaves as in the second part of the alternative, then a deadlock can occur.



Z.120(09)Amd.2_FI.6

```
msc Local;
inst System1, System2;
  instance System1;
    alt begin simplealt shared all;
      out Interrupt to System2;
    alt;
      out Continue to System2;
    alt end;
  endinstance;
  instance System2;
    alt begin simplealt shared all;
      in Interrupt from System1;
      action Stop;
    alt;
      in Continue from System2;
    alt end;
  endinstance;
endmsc;
```

Figure I.6 – A local choice MSC

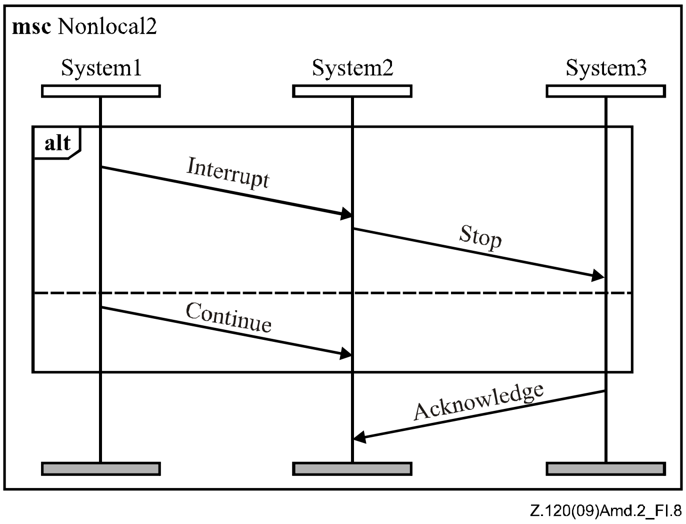


Z.120(09)Amd.2_FI.7

```
msc Nonlocal1;
inst System1, System2, System3;
  instance System1;
    alt begin simplealt shared all;
      out Interrupt to System2;
      in Agreed from System3;
    alt;
      in Alive from System2;
      out Acknowledge to System3;
    alt end;
  endinstance;
  instance System2;
    alt begin simplealt shared all;
      in Interrupt from System1;
      out Stop to System3;
    alt;
      in Continue from System3;
      out Alive to System1;
    alt end;
  endinstance;
  instance System3;
    alt begin simplealt shared all;
      in Stop from System2;
      out Agreed  to System1;
    alt;
      out Continue to System2;
      in Acknowledge from System1;
    alt end;
  endinstance;
endmsc;
```

Figure I.7 – A non-local choice MSC

Note that local choice property is not purely local to an alternative frame. Consider, for instance, the example of Figure I.8. According to the semantics of MSCs [b-Reniers99], and [b-ITU-T Z.120 Annex B], instance *System*3 can decide to send message *Acknowledge* without waiting for the decision of *System*1. However, if message *Acknowledge* is sent, this means that nothing occurs in the alt frame on instance *System*3, and then that the first behaviour of the alternative is ruled out.




```
msc Nonlocal2;
inst System1, System2, System3;
  instance System1;
    alt begin simplealt shared all;
      out Interrupt to System2;
    alt;
      out Continue to System2;
    alt end;
  endinstance;
  instance System2;
    alt begin simplealt shared all;
      in Interrupt from System1;
      out Stop to System3;
    alt;
      in Continue from System3;
    alt end;
    in Acknowledge from System3
  endinstance;
  instance System3;
    alt begin simplealt shared all;
      in Stop from System2;
    alt;
    alt end;
    out Acknowledge to System2;
  endinstance;
endmsc;
```

Figure I.8 – A non-local MSC

An important property of local choice MSCs is that they can be implemented, provided some additional control information is added to the contents of messages that are exchanged between instances. For more information, read [b-Helouet00], and [b-Genest02]. Local choice MSCs are a subclass of globally cooperative MSCs described hereafter.

I.4.3 Globally cooperative MSCs (GCMSCs)

An MSC description is globally cooperative if, in all loops that can appear in the description, the body of the loop does not form disconnected parts of behaviours running on distinct groups of instances. MSC *Counting* in Figure I.9 is not a globally cooperative MSC: the part of the MSC enclosed in the loop frame contains two atomic actions located on different instances. MSC *Data* of Figure I.5 is globally cooperative.

For two globally cooperative MSCs $M1$ and $M2$, the following properties are decidable:

(EquivF) $F(M1) = F(M2)$?

(RefF) $F(M1) \subseteq F(M2)$?

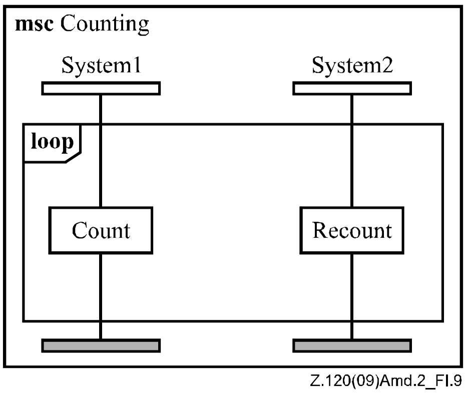
(IntF) $F(M1) \cap F(M2) = \phi$?

When considering two MSC descriptions $M1$ and $M2$, whenever $M2$ is globally cooperative, the following problems have an algorithmic solution.

(*EquivF*) $F(M1) = F(M2)$?

(*RefF*) $F(M1) \subseteq F(M2)$?

There are also generic implementation procedures for globally cooperative MSCs [b-Genest04], but the drawback is that the obtained implementations can contain deadlocks, which is in general an undesirable property of a system.




```
msc Counting;
inst System1, System2;
  instance System1;
    loop begin simpleloop shared all;
      action count;
    loop end;
  endinstance;
  instance System2;
    loop begin simpleloop shared all;
      action recount;
    loop end;
  endinstance;
endmsc;
```

Figure I.9 – A non-globally cooperative MSC

I.5 Summary of results

We recall here the relationship between different classes of MSCs. Local choice MSCs and regular MSCs are necessarily globally cooperative MSCs. Table I.1 should be read line by line, i.e., for inclusion of subclasses ($\subseteq$), the class mentioned by each line is contained in the class mentioned by the column.

Table I.1 – Comparison of syntactical subclasses of MSCs

	RMSC	**LMSC**	**GCMSC**	**MSC**
RMSC	=		$\subseteq$	$\subseteq$
LMSC		=	$\subseteq$	$\subseteq$
GCMSC			=	$\subseteq$
MSC				=

Table I.2 below recalls the decidability of the problems listed in clause I.2. "Yes" means that the considered problem is decidable for the class of MSC. "No" means that the considered problem is undecidable for the class of MSC. Note that for $Mc1$, $Mc2$ and $Mc3$, there is no immediate answer, as the existence of a decision procedure for local choice and globally cooperative depends on the nature of the properties considered.

Table I.2 – Decidable problems for MSC subclasses

	Mc1	Mc2	Mc3	EquivL	EquivF	RefL	RefF	IntL	IntF	Rez
MSC	No	No	No	No	No	No	No	No	No	No
RMSC	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No
LMSC	?	?	?	Yes	Yes	Yes	Yes	Yes	Yes	No
GCMSC	?	?	?	Yes	Yes	Yes	Yes	Yes	Yes	No

Table I.3 below recalls the different classes of MSC that can be implemented. Implementation mechanisms can use additional information on message to ensure correctness of implementation. The notion of correctness is also subject to several interpretations. Some implementation approaches (see for instance [b-Genest04]) consider that an MSC description and an implementation are only compared according to their correct runs, and do not consider deadlocked executions of the implementation. This way, an implementation that can deadlock can be considered as correct. For each subclass of MSC in Table I.3, we indicate whether an implementation mechanism has been proposed, and the restrictions (presence of deadlocks).

Table I.3 – Implementation of MSCs

	Implementation
MSC	?
RMSC	?
LMSC	With additional control information on messages
GCMSC	With deadlocks

I.6 Recommendations

We give here a list of recommendations according to the targeted application for a MSC specification.

I.6.1 Model checking

If the targeted application is model-checking of message sequence charts, the MSC description should remain regular, that is, for every loop:

- All messages sent from an instance I1 to an instance I2 should be acknowledged, either directly or indirectly.
- If the loop comports two atomic actions located on different instances, then there must be a direct or indirect message exchange between the instances where these actions are located.

I.6.2 Comparison of MSC specifications

If the targeted application is a comparison of specifications, then the message sequence charts used should remain globally cooperative; that is, for every loop, each instance or group of instance must either send or receive a message from the rest of the instances participating to the loop.

I.6.3 Implementation

If the targeted application is implementation of specifications, then the message sequence charts used should remain local choice; that is, for every alternative, the parts of the MSC in the scope of each part of this alternative should start with events located on a single instance.

Bibliography

[b-ITU-T Z.120 Annex B]	Recommendation ITU-T Z.120 Annex B (1998), *Formal semantics of message sequence charts*.
[b-Alur05]	Alur Rajeev, Etessami Kousha, and Yannakakis Mihalis (2005), *Realizability and verification of MSC graphs*, Theoretical Computer Science. 331(1): 97-114.
[b-Clarke99]	Clarke Edmund M., Grumberg Orna, and Peled Doron (1999), *Model Checking*, MIT Press, December.
[b-Khendek99]	Abdalla Miguel, Khendek Ferhat, and Butler Greg (1999), *New Results on Deriving SDL Specifications from MSCs*, in the Proceedings of SDL Forum'99, Elsevier Science B. V., R. Dssouli, G.v. Bochmann and Y. Lahav (eds.), Montreal, Canada, June 21-25.
[b-Genest02]	Genest Blaise, Muscholl Anca, Seidl Helmut, and Zeitoun Marc (2002), *Infinite-State High-Level MSCs: Model-Checking and Realizability,* Proceedings of ICALP, pp. 657-668.
[b-Genest04]	Genest Blaise, Kuske Dietrich, and Muscholl Anca (2004), *A Kleene Theorem for a Class of Communicating Automata with Effective Algorithms,* Proceedings of DLT 2004, pp. 30-48, LNCS 3340.
[b-Helouet00]	Hélouët Loïc, and Jard Claude (2000), *Conditions for synthesis of communicating automata from HMSCs*, 5th International Workshop on Formal Methods for Industrial Critical Systems (FMICS), Berlin, 3-4 April.
[b-Holzmann99]	Holzmann Gerard J., and Smith Margaret H. (1999), *Software Model Checking*, Proceedings of FORTE 1999, pp. 481-497.
[b-Reniers99]	Mauw Sjouke, and Reniers Michel A. (1999), *Operational Semantics for MSC'96*, Computer Networks and ISDN Systems 31(17):1785-1799, Elsevier Science B.V.

SERIES OF ITU-T RECOMMENDATIONS

Series A	Organization of the work of ITU-T
Series D	General tariff principles
Series E	Overall network operation, telephone service, service operation and human factors
Series F	Non-telephone telecommunication services
Series G	Transmission systems and media, digital systems and networks
Series H	Audiovisual and multimedia systems
Series I	Integrated services digital network
Series J	Cable networks and transmission of television, sound programme and other multimedia signals
Series K	Protection against interference
Series L	Construction, installation and protection of cables and other elements of outside plant
Series M	Telecommunication management, including TMN and network maintenance
Series N	Maintenance: international sound programme and television transmission circuits
Series O	Specifications of measuring equipment
Series P	Terminals and subjective and objective assessment methods
Series Q	Switching and signalling
Series R	Telegraph transmission
Series S	Telegraph services terminal equipment
Series T	Terminals for telematic services
Series U	Telegraph switching
Series V	Data communication over the telephone network
Series X	Data networks, open system communications and security
Series Y	Global information infrastructure, Internet protocol aspects and next-generation networks
Series Z	**Languages and general software aspects for telecommunication systems**

Printed in Switzerland
Geneva, 2010