Distributed system requirement modeling with Message Sequence Charts: the case of the RMTP2 protocol ⋆

Loïc Hélouët

IRISA/INRIA,
Campus de Beaulieu, 35042 Rennes Cedex, France

Abstract

This document describes a case study realized for the INTERVAL european project. The aim of this study was to test time features of MSC'2000 and the real time extensions proposed by the INTERVAL partners. The starting point for this study, was an IETF document describing requirements for a multicast protocol called RMTP2. Some significant requirements could not be modeled with MSC 2000, which lead to extension proposals for MSC. The main extension proposed is the introduction of multicast communications in MSC.

Key words: Message Sequence Charts, requirement modeling, Real time.

1 Introduction

The standardization of Message Sequence Charts by ITU can be divided in three historical periods. At their very beginning (before 1992), MSC were limited to very simple diagrams called basic MSCs, and mainly composed of instances messages and timer operations (this version of MSC is often referred as MSC'92). During the next study period (from 1992 to 1996), MSC have been extended with composition mechanisms (inline expressions and HMSC), allowing for the expression of more elaborated behaviors. This language is known as MSC'96 [1]. Recent extensions have introduced data, object orientation, time observation, and so on, leading to a standard called MSC'2000 [2].

⋆ The major part of this work has been realized at France Télécom R&D Lannion
Email address: `loic.helouet@irisa.fr` (Loïc Hélouët).
URL: `www.irisa.fr/triskell/perso_pro/helouet/LHengpage.html` (Loïc Hélouët).

During each study period, the language has been used on practical examples. [1] for example, uses MSC'96 to describe a telecommunication service called CCBS (Completion of Calls to Busy subscriber). In [3], MSC'96 are used as an input language for the specification of distributed reactive systems. [4] uses MSC'2000 to specify the connection establishment part of the INRES protocol. Putting MSC to practice allows to measure the expressiveness of the language, points out inconsistencies, therefore providing useful feedback for the next study period. [5], for example shows some inconsistencies in MSC'96 related to the use of gates in combination with loops. [6] shows that timers can become a problem when they are used within an iteration. [7] exhibits some cases in MSC'96 that are syntactically correct, but can lead to ambiguous interpretations (these cases are called pathological Message Sequence Charts, and are mainly due to a language misuse). So, an experiment with Message Sequence Charts often provides useful information on necessary extensions, but also gives guidelines on how to use the language.

This document presents a case study realized for the European project INTERVAL. The aim of INTERVAL is to provide languages such as MSC, SDL and TTCN with extensions for real time systems specification analysis, and testing. Within this framework, some extensions have been proposed to the standard MSC'2000 by ITM Lübek. The existing time constructs of MSC and the new extensions have been used to model requirements of a reliable multicast protocol called RMTP2. For this study, we took as a base document a draft written by the IETF describing the protocol in natural language [8]. Some parts of the requirements that could be expressed by Message Sequence Charts have been translated. From the beginning, it was obvious that the whole document could not be described with MSC. The first reason is that in some cases, MSC are not expressive enough to describe even very simple scenarios. For such cases, we propose extensions to the language, and apply them. The second reason for leaving untranslated some parts of the IETF draft is the abstraction level for requirement expression. When some requirement needed low-level description, we have considered that MSC were not really adapted for this, and left the concerned part.

This document is organized as follows: section 2 describes how time is represented in MSC'2000 and the extension that was proposed by the INTERVAL project. Section 3 proposes extensions to the language that were needed to model significant requirements of the RMTP2 protocol. Section 4 list and justifies the design decisions that were taken at the beginning of the study. Section 5 describes the RMTP2 protocol, and illustrates the use of MSC on some peculiar parts of RMTP2 requirements. Section 6 concludes this work.

2 MSC and Time

This section describes how times is handled by the new standard MSC'2000 [2], and the timed extension that was proposed by the INTERVAL project [9].

2.1 Timers

Message Sequence Charts allow for operations on timers (set reset, and time-out). Timers have the same semantics as in SDL, and should be set and reset (or expire) on the same instance. However, timer constructs suppose that a clock mechanism is used in an eventual implementation. The example of Figure 1 shows how timers may be used. This MSC could be translated in natural language as "after sending $m1$, if $m2$ is not received within 10 ms, then an error is detected".

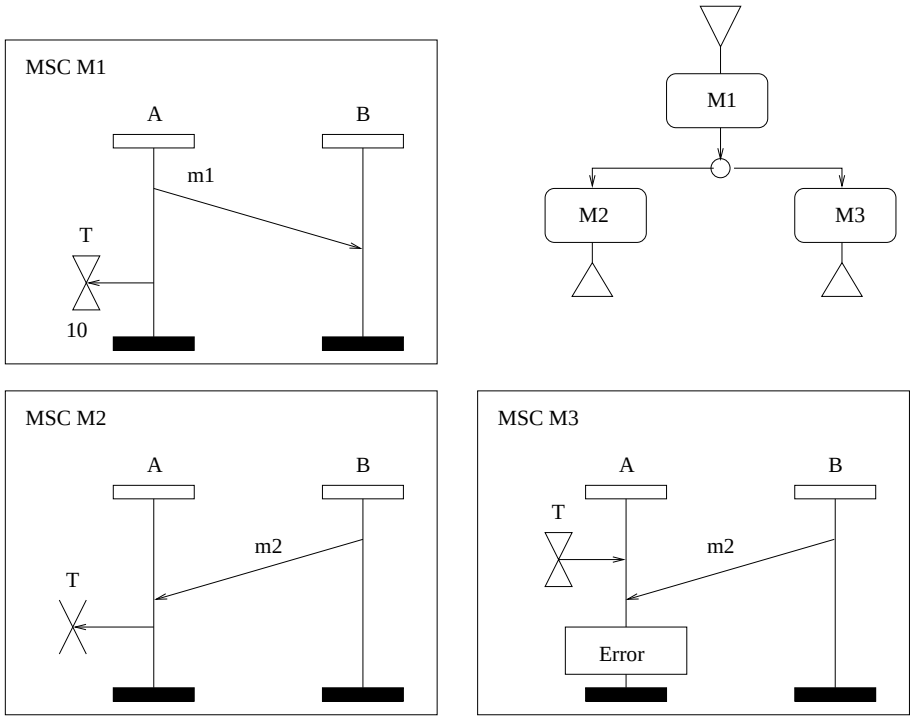

Fig. 1. Use of Timers

In some case, a designer does not wish to specify how time constraints are implemented, but only express requirements such as "the execution of a given scenario should be performed within at least 10 ms and at most 100 ms". Timers are not adapted for expressing such time constraints. Furthermore, a system under design may naturally fulfill a time constraint without imposing an interruptive timer definition.

2.2 Time in MSC'2000

The new ITU standard MSC'2000 [2] allows for the definition of very elaborated timed constraints, without defining how they are implemented. The new constructs that were added to the MSC'96 language [1] are occurrence dates and time interval definitions.

2.2.1 Occurrence dates

The new temporal extensions provided by MSC'2000 allow for the definition of a date of occurrence for each event in a MSC. MSC $Date1$ in Figure 2 shows a scenario in which emission of $m1$ occurs at date $date1$, and reception of $m2$ at $date2$, where $date1$ and $date2$ are time variables referring to event occurrence dates. MSC $Date2$ shows the same scenario with imposed dates. Note that in the first MSC, emission of $m1$ or reception of $m2$ can occur at any time and the occurrence date is recorded, while in scenario $Date2$, emission of $m1$ must occur at date 0, and reception of $m2$ at date 335 ms.

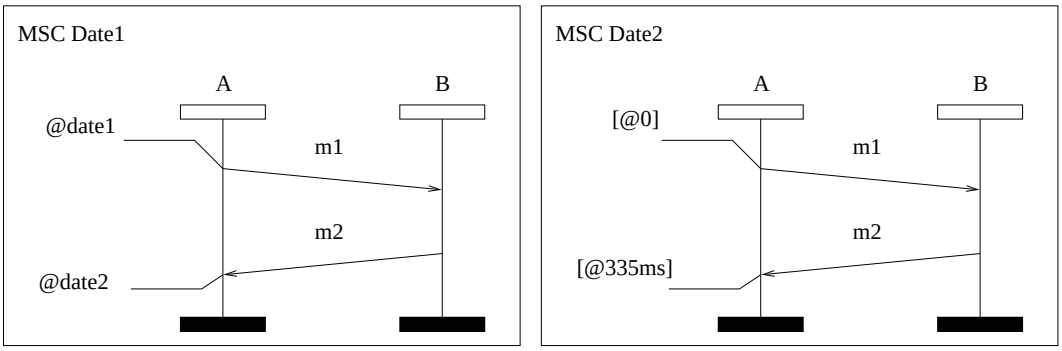

Fig. 2. Definition of dates

2.2.2 Time intervals

MSC'2000 also allow for the definition of time intervals elapsed between the occurrence of two events. The definition of a duration is done by a double arrow, one extremity pointing on the first even of the interval, the second on the last event of the interval. The example $Interval1$ of Figure 3 shows the definition of a time interval called $inter1$. As previously for dates, it is possible to give values instead of naming an interval. The example $Interval2$ of Figure 3 shows a scenario in which the time elapsed between the emission of $m1$ and the reception of $m2$ should be included in the interval $[10, 100\ ms]$. The example $Interval3$ of Figure 3 shows how a duration can be associated to a message transmission. It is also possible to reuse intervals defined to express more elaborated timed constraints. The example $Interval4$ of Figure 3 indicates that the time elapsed between the emission of $m1$ and the emission

of $m3$ should be greater than twice the time needed for transmitting $m1$, and lower than 3 times this duration.

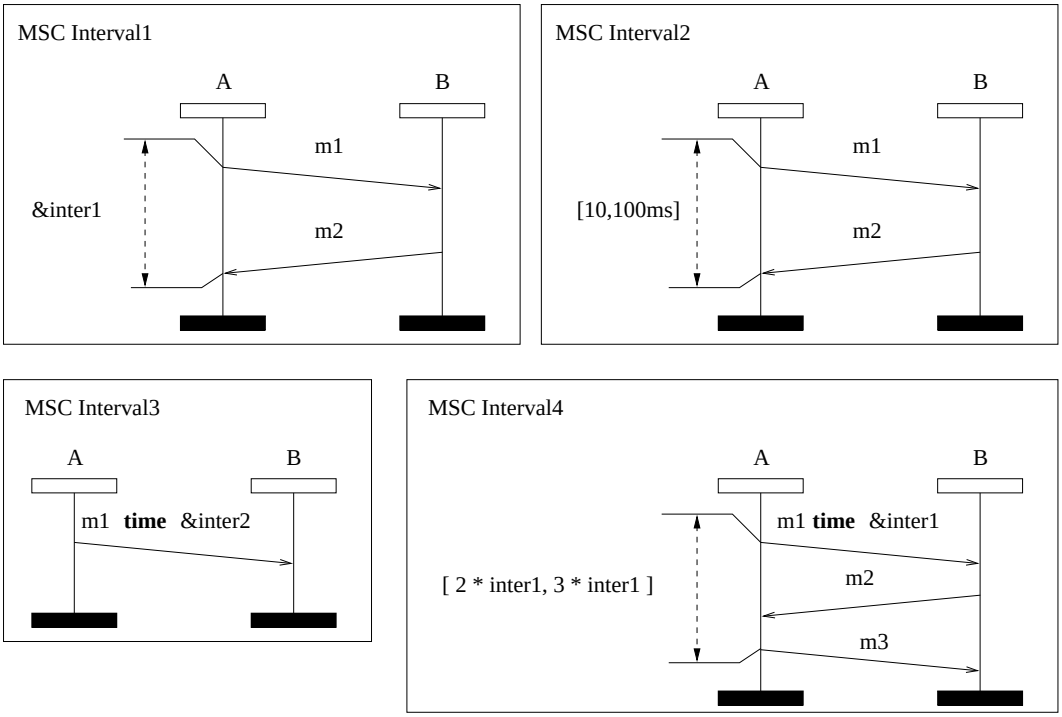

Fig. 3. Time intervals definition

In the standard, time intervals can be defined between any pair of events contained in a MSC document. This means that an intervals starting in a MSC and ending in another are considered as valid. However, an interval definition may lead to ambiguous interpretation due to iteration: an interval start event may be matched to more than one end event (a similar problem was stated in [6] for timers setting and timeout).

2.3 Periodicity proposed by INTERVAL

Periodicity has been introduced to allow the definition of time intervals between two consecutive occurrences of the same event. The example of Figure 4 shows a scenario where a message $m1$ must be sent regularly. The time elapsing between two consecutive emissions should be greater than 2 seconds and lower than 3 seconds.

3 Extensions needed

This section proposes some extensions to MSC. A better definition for data was not a clear necessity for this work: assuming that data were local and could

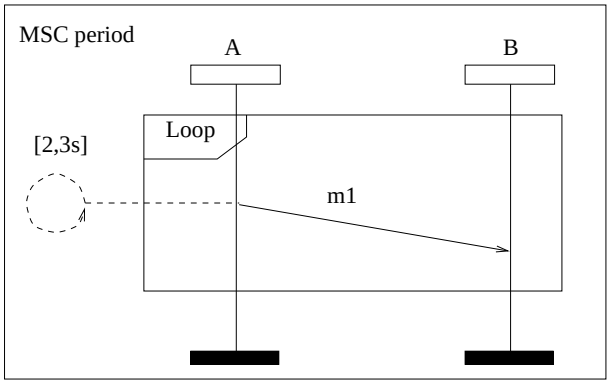

Fig. 4. Periodicity

be used in guards defined by conditions was sufficient. However, an extension of loop constructs and a definition of multicast were needed to complete the modeling of important parts of the requirements.

3.1 Data

Data have only been introduced in the MSC'2000 standard. They can be manipulated by actions, be carried by messages, or be used in predicates for conditions. Yet, some clarification are still missing:

- The scope of variables is still undefined. It has not been decided if variables should be local (in such case, their use in guards may be restricted to conditions on one single instance) or global (in which case it may not be possible to implement such a variable).
- Data of a MSC document are defined in a declaration part, which can contain a data definition zone expressed in an external language. However, the standard is not precise on how these externally defined types are used. For example, a message can carry information t of type T defined externally as a structure, and a guard can refer to a specific field of t. Hence, a data language (such as ASN.1 or a C-like language) should be chosen for MSC, allowing for the definition of new types such as structures, and defining precisely how these types must be used.

3.2 For loops

In the ITU standard, loops are defined by an expression of the kind $loop < x, y >$ where x and y are respectively the minimal and the maximal number of repetitions of a scenario. The RMTP2 requirements point out the need for more general concepts of loops: for example, a retransmission should be performed for each missed packet. In some other parts of the IETF document,

behaviors are repeated for a set of instances (for example for each child of a control node). We propose the following definition: $loop < varname\ in\ X >$, where $varname$ is a variable name that can be used in the body of the loop, and X is a set of elements. Note that this definition remains compatible with the previous notations, as expressions such as $loop\ < 1, n >$ will just be noted $loop\ <\ i\ in\ 1..n >$.

3.3 Arrays of timers

In order to avoid message loss, RMTP2 associates a timer to each missed packet. It seems more convenient for applications with many timers to define array of timers, as in SDL. Set, reset and timeout remain the same, the only difference is the possibility to add an index to a timer name, (for example $set(Timer\_Data[i], 10ms)$). Timers definition could be integrated to the data definition part mentioned previously.

3.4 Multicast

The following behavior is an extract from the IETF draft: "the sender application provides Data packets to the sender node. The sender node assigns consecutive sequence numbers to the Data packets and **multicasts** the packets on the data channel".

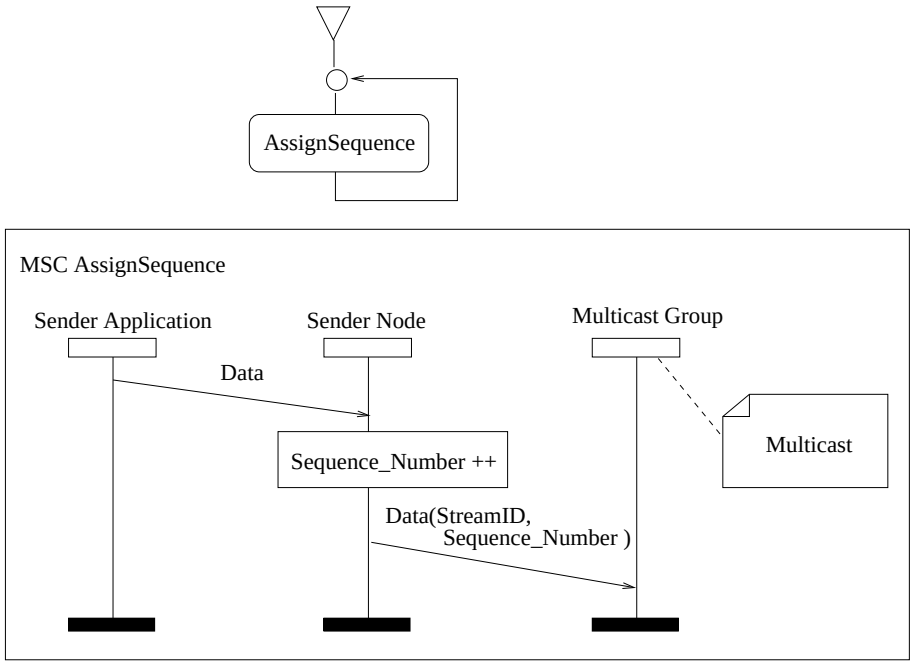

Fig. 5. Sequence number assignment

The only assumption about communications in Message Sequence Charts is that messages are asynchronous. There is no way of defining multicast, or synchronous communications. We propose to define multicast for MSC the following way :

- Instance can be multicast groups, which is indicated by a "multicast" comment attached to the instance axis (as shown in Figure 5). Of course, any message sent from an instance to a multicast group is a multicast message.
- Multicast sending and receiving are differentiated from the classical send and receive events.

In addition to these requirements, a semantics of multicast in MSC should preserve some rules:

- Map one single emitting event to multiple receptions,
- Allow the evolution of groups of receivers,
- Preserve the semantics of sequential composition in MSC (weak sequential composition).

Note that the semantics proposed below is just a tentative semantics defined to allow the use of multicast within the INTERVAL framework. As such, some of the choices made are open to discussion, and the semantics should be subject to major improvements.

Let us give a new definition of MSC extending the notations of [11], and integrating the concept of multicast. Following the definitions of [12,13], we will consider a MSC as a partial order between events. A MSC is a tuple $M =< E, \leq, A, I, \alpha, \phi, m, NG >$ where:

- E is a set of events
- $\leq$ is a partial order relation on events, due to sequential ordering on instance axis, and ordering imposed by messages,
- A is a set of labels,
- I is a set of instance names,
- $\alpha : E \longrightarrow A$ is a function associating an action name to each event,
- $\phi : E \longrightarrow I \cup NG$ is a function associating a location to each event of a MSC,
- $m : E \longrightarrow E$ is a mapping associating a sending event to its corresponding receiving event. Note that due to message crossing, non uniqueness of message labeling, and transitivity of the order relation, this mapping can not always be deduced from $\leq$ and ϕ.
- NG is a set of multicast group names,

In addition to usual actions allowed in a MSC (send, receive, timers set/reset/timeout), we define the following ones:

- Join(Instance_Name,Group_Name)
- Leave(Instance_Name, Group_Name)
- mcast_Send(Emitting_Instance, message, Receiving_Group)
- mcast_Receive(Emitting_Instance, message, Receiving_Group | receiving_instance)

We propose the following semantics rules. First, the behaviors defined by a multicast MSC will depend on the configuration of multicast groups. Therefore, the semantics rules will be of the form:

$$\frac{Preconditions}{M, ctxt \quad \underline{action}_{\rightarrow} \quad M', ctxt'}$$

Where $M =< E, \leq, A, I, \alpha, \phi, m, NG >$ and $M' =< E', \leq', A', I', \alpha', \phi', m', NG' >$ are MSC and $ctxt = (R, G)$ and $ctxt' = (R', G')$ are contexts. A context will contain two functions :

- $G : NG \longrightarrow \mathcal{P}(I)$ indicating the current composition of a group.
- $R : E \longrightarrow \mathcal{P}(I)$ indicating the instances that have to perform an event (for multicast receptions). For a multicast reception e, $R(e)$ is fixed by the corresponding emission. If $m^{-1}(e)$ has not been executed then $R(e) = \emptyset$.

The contents of a multicast group may change before a multicast message is received. For this reason, we impose that the group of receivers for a given message is set when the message is sent, which explains why $R(e)$ can be different from $G(\phi(e))$ for a multicast reception. For a given MSC M, we define as $Min(M)$ the set of minimal event with respect to the order relation:

$$Min(M) = \{e \in E | \nexists e' \in E, e \neq e' \wedge e' \leq e\}$$

Furthermore, for any set E, and any subset $E' \subseteq E$, the projection of the order relation $\leq$ on E' is $\leq_{|E'} = \leq \cap E' \times E'$ and the projection of a function f on E' is the restriction:

$$f_{|E'} : E' \longrightarrow E'$$
$$e \longrightarrow f(e)$$

The operational semantics rule for non-multicast events will be as usual:

$$\frac{\exists e \in Min(M), \alpha(e) = act}{M =< E, \leq, A, I, \alpha, \phi, m, NG >, ctxt \quad \underline{act}_{\rightarrow} \quad M' =< E', \leq_{|E'}, A, I, \alpha_{|E'}, \phi_{|E'}, m_{|E'}, NG >, ctxt}$$

with $E' = E - \{e\}$.

3.4.1 Joining and leaving a multicast group

When an instance i joins a multicast group ng, it is allowed to receive all messages sent to this group.

$$\frac{e \in Min(M), \alpha(e) = Join(i, ng)}{M =< E, \leq, A, I, \alpha, \phi, m, NG >, ctxt = (R, G) \xrightarrow{Join(i,ng)} M' =< E', \leq_{|E'}, A, I, \alpha_{|E'}, \phi_{|E'}, m, NG >, ctxt' = (R, G')}$$

Where $E' = E - \{e\}$, and

$$G' : NG \longrightarrow \mathcal{P}(I)$$
$$g \longrightarrow G(g) \cup \{i\} \text{ if } g = ng$$
$$g \longrightarrow G(g) \text{ otherwise}$$

After an instance i has left a multicast group ng, it should not receive messages sent to this group.

$$\frac{e \in Min(M), \alpha(e) = Leave(i, ng)}{M =< E, \leq, A, I, \alpha, \phi, m, NG >, ctxt = (R, G) \xrightarrow{Leave(i,ng)} M' =< E', \leq_{|E'}, A, I, \alpha_{|E'}, \phi_{|E'}, m, NG >, ctxt' = (R, G')}$$

With $E' = E - \{e\}$ and

$$G' : NG \longrightarrow \mathcal{P}(I)$$
$$g \longrightarrow G(g) - \{i\} \text{ if } g = ng$$
$$g \longrightarrow G(g) \text{ otherwise}$$

3.4.2 Multicast message emission

$$\frac{e \in Min(M), \alpha(e) = mcast\_send(i, msg, ng)}{M =< E, \leq, A, I, \alpha, \phi, m, NG >, ctxt = (R, G) \xrightarrow{mcast\_send(i,msg,ng)} M' =< E', \leq_{|E'}, A, I, \alpha_{|E'}, \phi_{|E'}, m_{|E'}, NG >, ctxt = (R', G)}$$

Where $E' = E - \{e\}$, and

$$R' : NG \longrightarrow \mathcal{P}(I)$$
$$f \longrightarrow R(f) \text{ if } f \in dom(R)$$
$$e' \longrightarrow G(ng) \text{ if } e' = m(e)$$

We suppose that multicast messages are still asynchronous, but that the list of recipients is set at message emission. When a message is sent in multicast,

the function R' associates the contents of the multicast group ng to the corresponding reception e'. So, a message m will be received by all instances that belonged to group ng when m was sent.

3.4.3 Multicast message reception

A reception of a multicast message can be performed by any member i of the destination group ng if the message has already been sent, and i has performed all its preceding receptions in ng.

$$\frac{\begin{array}{c}\exists e \in E, \alpha(e) = mcast\_receive(sender, msg, ng), i \in R(e), |R(e)| > 1, \\ \forall e' \leq e \text{ such that } \phi(e') = ng, i \notin R(e'), \\ \nexists e'' \leq e \text{ such that } m(e'') = e\end{array}}{\begin{array}{c}M =< E, \leq, A, I, \alpha, \phi, m, NG >, ctxt = (R, G) \xrightarrow{mcast\_receive(sender, msg, i)} \\ M, ctxt' = (R', G)\end{array}}$$

with
$$\begin{array}{l} R' : NG \longrightarrow \mathcal{P}(I) \\ \quad f \longrightarrow R(f) \text{ if } f \neq e \\ \quad e \longrightarrow R(e) - \{i\}\end{array}$$

$$\frac{\begin{array}{c}\exists e \in E, \alpha(e) = mcast\_receive(sender, msg, ng), i \in R(e), |R(e)| = 1, \\ \forall e' \leq e \text{ such that } \phi(e') = ng, i \notin R(e'), \\ \nexists e'' \leq e \text{ such that } m(e'') = e\end{array}}{\begin{array}{c}M =< E, \leq, A, I, \alpha, \phi, m, NG >, ctxt = (R, G) \xrightarrow{mcast\_receive(sender, msg, i)} \\ M' =< E', \leq_{|E'}, A, I, \alpha_{|E'}, \phi_{|E'}, m, NG >, ctxt' = (R', G)\end{array}}$$

where $E' = E - \{e\}$ and $R' = R_{|E'}$

The first rule describes a multicast reception when the set of instances that must receive message msg is composed of more than one instance, and the second rule depicts a situation where the last member of a group receives the message. Note that actions associated to multicast groups are only receptions. This restriction could be removed to allow the definition of generic behaviors associated to groups of communication, but this is beyond the scope of this article. An example of MSC containing a multicast group and the corresponding semantics are provided Figure 6. Figure 7 provides another example of multicast MSC. Messages $m1$, $m2$, and $m3$ are multicast to group G. Initially,

G is only composed of instance B, so message $m1$ is only sent to B. Then, instance C is added to G. So, message $m2$ is sent to B and C. Finally, B leaves the multicast group, so message $m3$ is only sent to C.

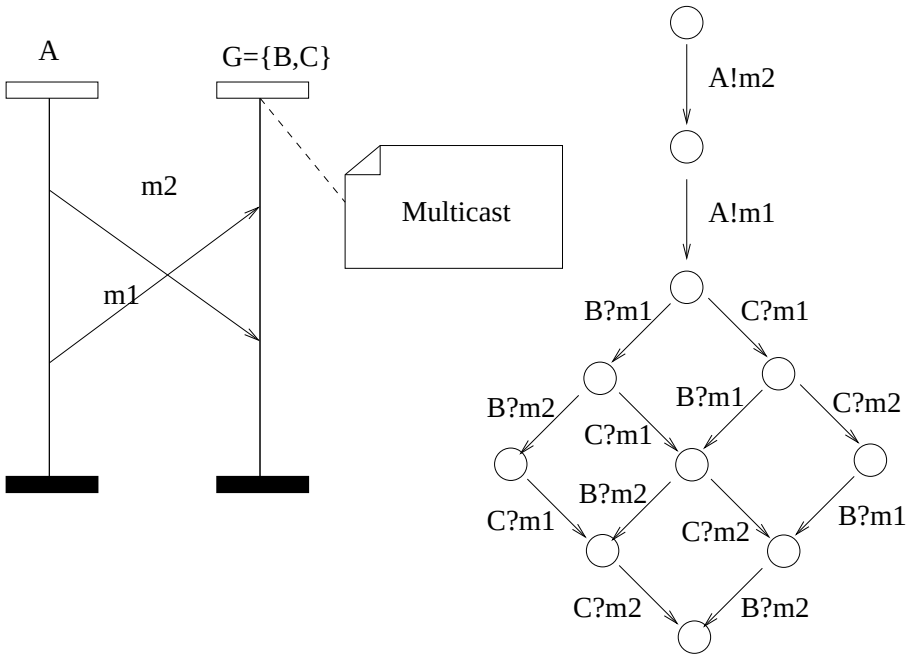

Fig. 6. Example of multicast communications and corresponding transition system

Some confusion may occur when a MSC contains a multicast emission to a group G, and an instance I that is also a member of G (see MSC M in Figure 8 for an example). The intuitive behavior defined in such a case is that multicast receptions on I for group G and separate events on I should be concurrent. However, a MSC similar to M can be obtained by composition of a MSC containing the multicast communication between $Sender$ and G, and another containing only action A. The semantics of both should be the same, but this makes the definition of sequential composition very uneasy. A safe restriction would be to limit the use of multicast to MSC where groups and set of instances remain disjoint.

3.4.4 Sequential composition of multicast MSC

MSC allows for the composition of elements, with operators such as choice, iteration, and sequential composition. Usually, the sequential composition of two MSC $M1$ and $M2$ is noted $M1 \circ M2$. According to the sequential composition semantics, an event e of $M2$ can not be executed in $M1 \circ M2$ if there is one event on the same instance $\phi(e)$ to be executed in $M1$. The sequential composition of MSC with multicast should remain consistent with the previous semantics of sequential composition [11]. However, the contents of multicast groups can only be known at execution time. Consider two MSC $M1$ and $M2$.

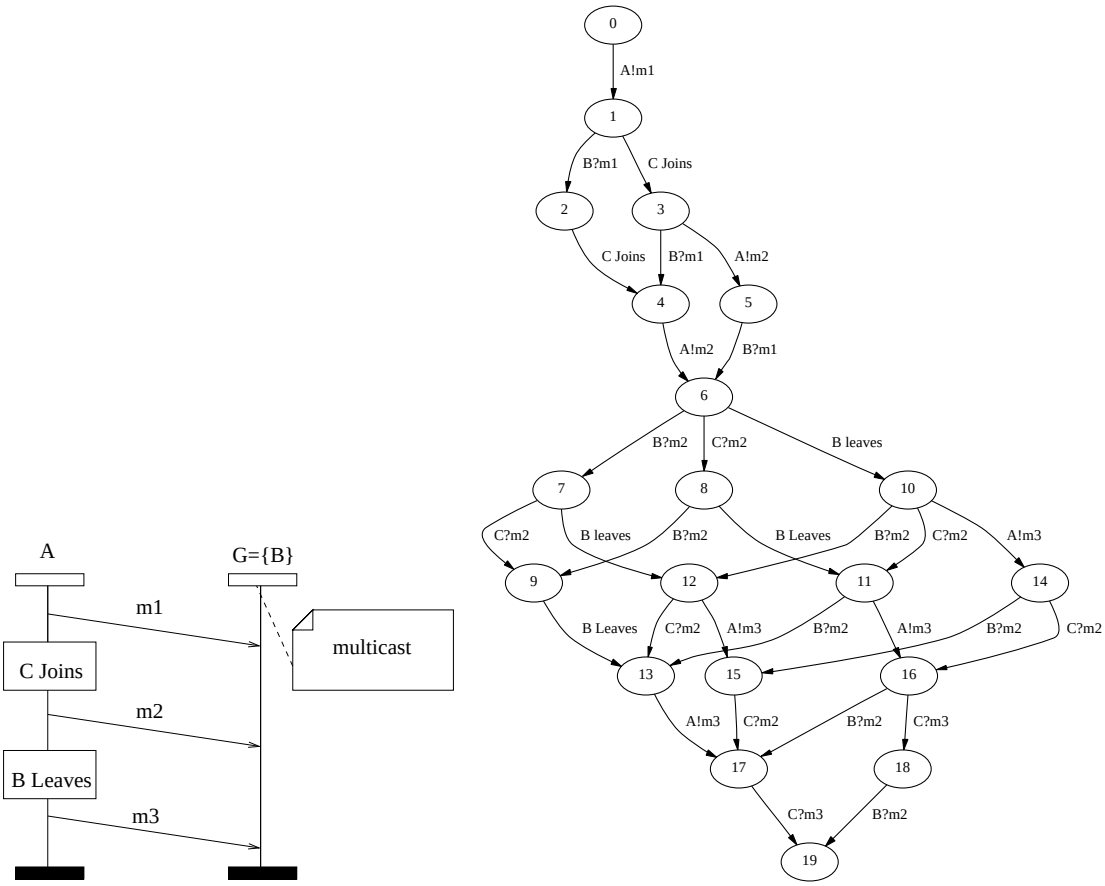

Fig. 7. Example of multicast with join and leave and corresponding transition system

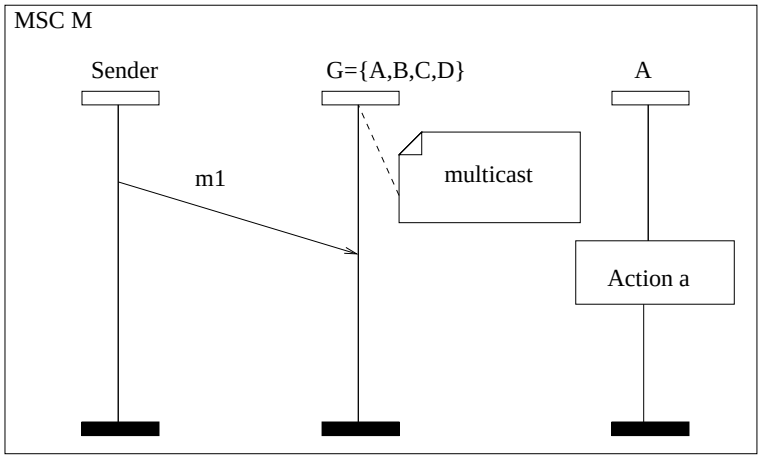

Fig. 8. Ambiguous MSC containing a multicast operation

Two behaviors can be defined. Either any event e of $M2$ is forbidden if a multicast message can be received by $\phi(e)$ in $M1$. Or any event e can be executed if it is minimal for $M1 \circ M2$, and that no multicast message reception has to be performed by $\phi(e)$ at the moment of the execution of e. We will adopt the second approach, which seems to be less restrictive. Then, the semantics of sequential composition when MSC can contain multicast messages becomes:

$$\frac{M1, ctxt \xrightarrow{x} M1', ctxt'}{M1 \circ M2, ctxt \xrightarrow{x} M1' \circ M2, ctxt'}$$

$$\frac{\begin{array}{c} M2, ctxt \xrightarrow{x} M2', ctxt' \text{ and} \\ \forall e \in \phi_1^{-1}(I_1), \phi_2(x) \neq \phi_1(e) \text{ and} \\ \forall e \in \phi_1^{-1}(NG_1), \phi_2(x) \not\subseteq R(e) \end{array}}{M1 \circ M2, ctxt \xrightarrow{x} M1 \circ M2', ctxt'}$$

All minimal events of $M1$ can be executed, but an event e of $M2$ can not be executed if there is a reception by a group G in $M1$, and G is known to contain $\phi(e)$. Figure 9 provides an example of sequential composition with multicast messages. At the beginning, G only contains instance C. The behavior described by the example will differ if B is added to the multicast group G before or after emission of message $m2$ The associated transition system calculated using the operational semantics is provided in Figure 10.

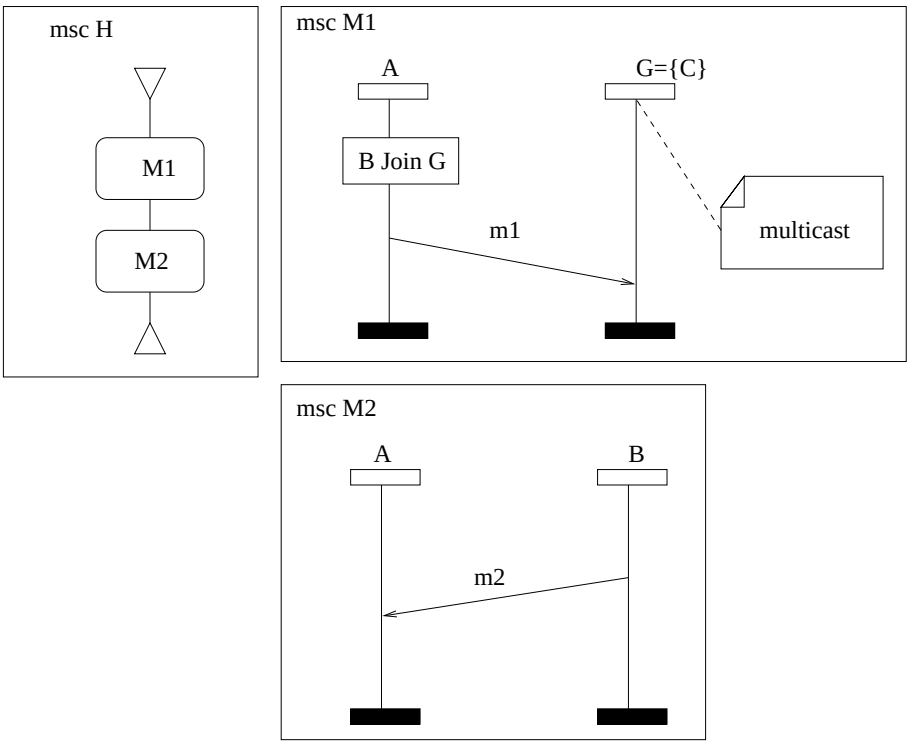

Fig. 9. Sequential composition and multicast

4 Deign decisions

MSC are supposed to provide clear understanding of typical executions. This intuitive understanding can be easily limited by inappropriate use of the lan-

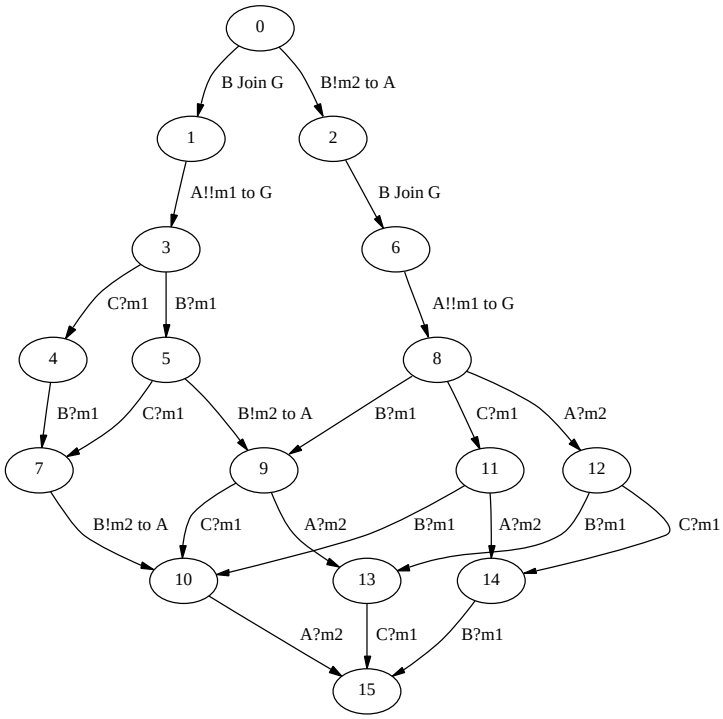

Fig. 10. Transition system associated to the Example Figure 9

guage. One may, for example, describe control structures similar to while loops in C programs. Clearly, MSC were not designed for such use, and the graphical syntax does not bring any help in understanding low level code-like descriptions. Our opinion is that MSC should remain clear, and rather abstract. As we are dealing with requirements, and that our intention is not to provide an exhaustive description of all possible runs of a system, this should not be seen as a limitation. Thus, we shall leave parts of the RMTP2 requirement when they are considered as too low level. Figure 11 shows how the simple loop below can be specified using MSC.

```
i:= 0
while (i <= 10) {
    Act(i);
    i++
}
```

With gates, MSC can also be used as communicating automata. As shown in [10], the uncontrolled use of gates limits the decidability results proved on closed MSC (ie compositions of patterns in which any message sent is received in the same pattern). Our opinion is that languages such as SDL are better adapted for describing communicating processes at a low level. From the beginning, MSC has been designed to define sequential compositions of communications, and using them to define parallel composition of communicating sequences seems to be a nonsense. Figure 12 shows how MSC can be used in

a SDL-like style. For this study, we have tried to limit the use of gates, to preserve clarity of requirement representation.

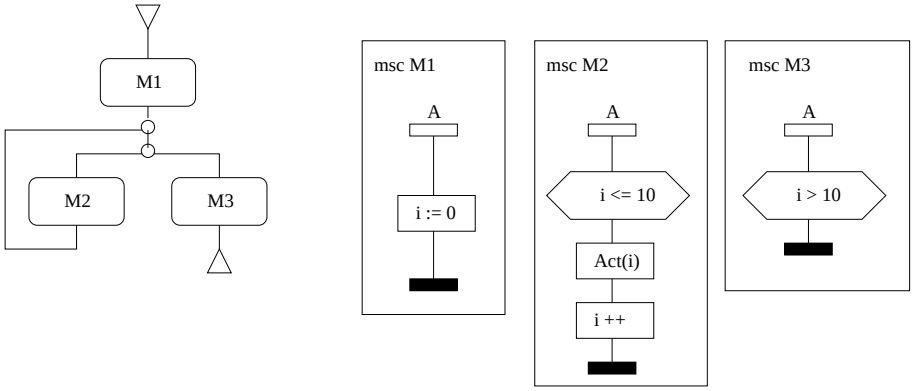

Fig. 11. While loop expressed by a MSC

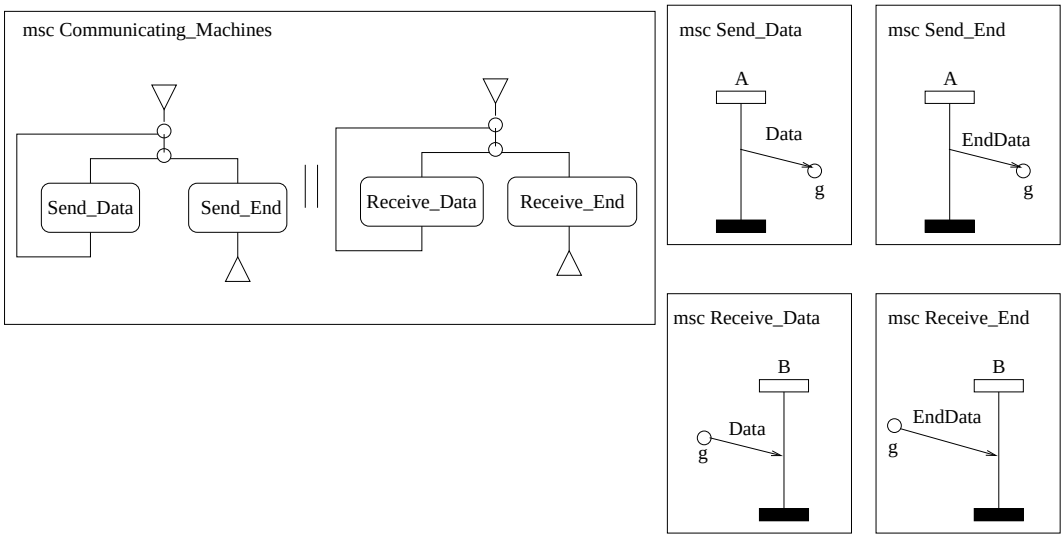

Fig. 12. Using MSC like communicating state machines

Another weak point for MSC is the lack of semantics for the new constructs that were added to the language in the 1996-2000 period. For example, data definition is not yet clear. For the scope of this study, we have considered that data were local, could be carried by messages, and used in predicates for guard definitions in conditions.

5 Modeling RMTP2 with MSC

5.1 Short description of RMTP2

This section briefly describes the main functionalities of RMTP2 (Reliable Multicast Transport Protocol). A more complete description can be found in [8]. RMTP2 provides a reliable transmission of data sequences for large groups of receivers. The objectives of RMTP2 are guaranteed reliability, high

throughput, and low end-to-end delay on any network topology, while providing the network manager with control over transmission traffic. In a lossy environment, reliability can only be obtained if data packets are acknowledged. However, due to the large number of receivers for each data channel, direct acknowledgment of packets from each receivers to the source would result in an immediate network congestion. For this reason, the network is organized as a tree. Receivers are grouped in local regions, and in each region a special control node is responsible for maintaining receiver membership and for aggregating the acknowledgments from the receivers in its region and forwarding them to the sender. The network comports a descending way, on which data are send using IP multicast to all members of the RMTP2 tree. Data are received from a sending application by a node called Sender Node, and are then multicast on the data channel. Packets for acknowledgment are aggregated by multiple levels of control nodes, which forward information about missed packets to the Sender Node. Control Nodes may be of two types : Aggregator Nodes, which only aggregate and forward information about missed packets to the upper level, and Designated Receiver Nodes, which can keep a copy of data packets, and retransmit missed data to the subtree situated below them. Acknowledgments are eventually aggregated at the top of the tree by a node called Top Node, which retransmits this information to the Sender Node. Each node in the tree provides multiple services such as data transmission, tree integrity control, quality of service maintenance. Figure 13 shows how a RMTP2 tree with 3 hierarchical levels is organized.

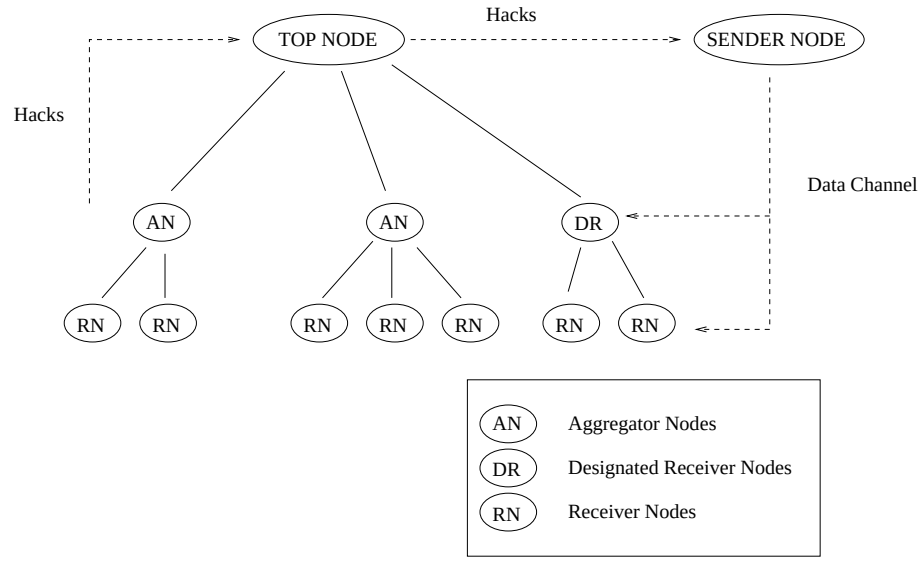

Fig. 13. A 3-level RMTP2 Tree

MSC allowed us to describe a large subset of the requirements written in the IETF Draft [8]. We have followed the document in a linear way, and tried to design a MSC for most of the behaviors described. The following sections provide a sample of the requirements. For each example, we provide the IETF requirement as it was expressed, and the MSC translation follows.

5.2 Heartbeat Packets

One of the main functions of nodes is to ensure tree integrity. A child node must be able to detect its parent failure, and join another parent. So, a parent node periodically sends heartbeat messages to its children, to notify it is alive. The following textual requirements appear in the IETF draft: " A control node periodically sends Heartbeat packets to notify its child nodes that it is alive. *Thb*: the number Thb is the time interval at which control nodes multicast Heartbeat packets."

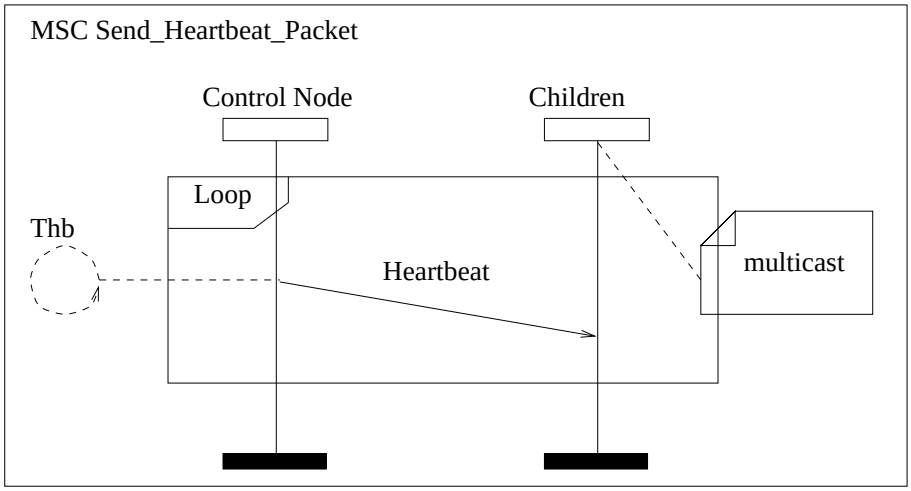

Fig. 14. Periodical Heartbeat packet emission

This part of RMTP2 requirements emphasizes the need for periodicity. Note that Heartbeat messages are sent on a multicast local channel (the definition of multicast will be discussed later). This behavior applies to any part of the RMTP2 tree composed of a control node and its children. Furthermore, nothing is specified about how such a time constraint should be implemented. Another possibility for illustrating this requirement is to use timers, as in Figure 15. Note that the meaning of Figures 14 and 15 are slightly different. Figure 14 requires that a heartbeat message is sent every Thb time units (this can be considered as a property to be checked on an implementation), while Figure 15 describes a timer mechanism for heartbeat emission.

5.3 Parent Failure detection

As indicated before, a child node waits for heartbeats packets from its parent. If Heartbeat packets have not been received for a too long time, the parent is considered to have failed. The following requirements are extracted from the IETF draft: "If the child does not receive any Heartbeat from the parent in an interval $F * Thb$, it declares the parent failed. F: the failure threshold constant, F, determines the threshold time for failure detection." Figure 16 describes a timer mechanism that allows the parent failure detection.

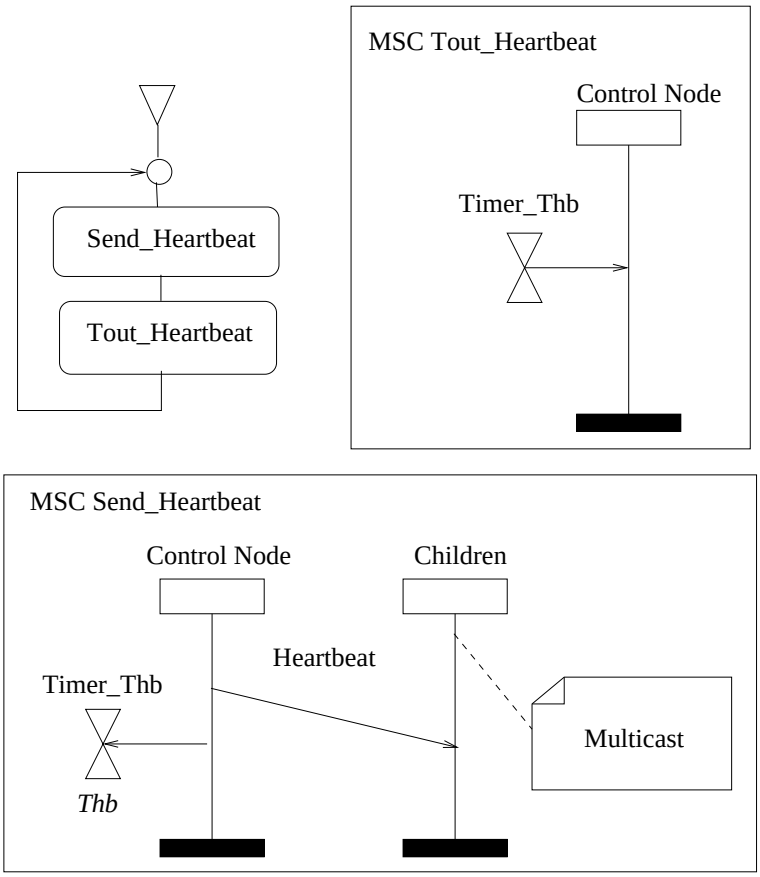

Fig. 15. Timers associated to Heartbeat packets

5.4 Join Algorithm

Data transmission is considered as a continuous flow of information that can be joined at any time. Connection to a data channel is immediate, but in order to be able to acknowledge data packets, or ask for retransmission, a receiver must be connected to the RMTP2 tree. A receiver has to contact its parent to be authorized to join the RMTP2 tree. The following requirements appear in the IETF draft:

"When a receiver node is created, it sends a separate JoinStream packet for each of the streams that it intends to receive. When a receiver node rejoins a stream after a parent failure, it may send a single JoinStream packet to join multiple streams.

Aggregator and designated receiver nodes send a JoinStream packet with StreamID value zero to join the RMTP tree, without joining any stream. An aggregator or designated receiver later joins the streams that their children join.

A node sends a JoinStream request to its parent node. It waits a time interval,

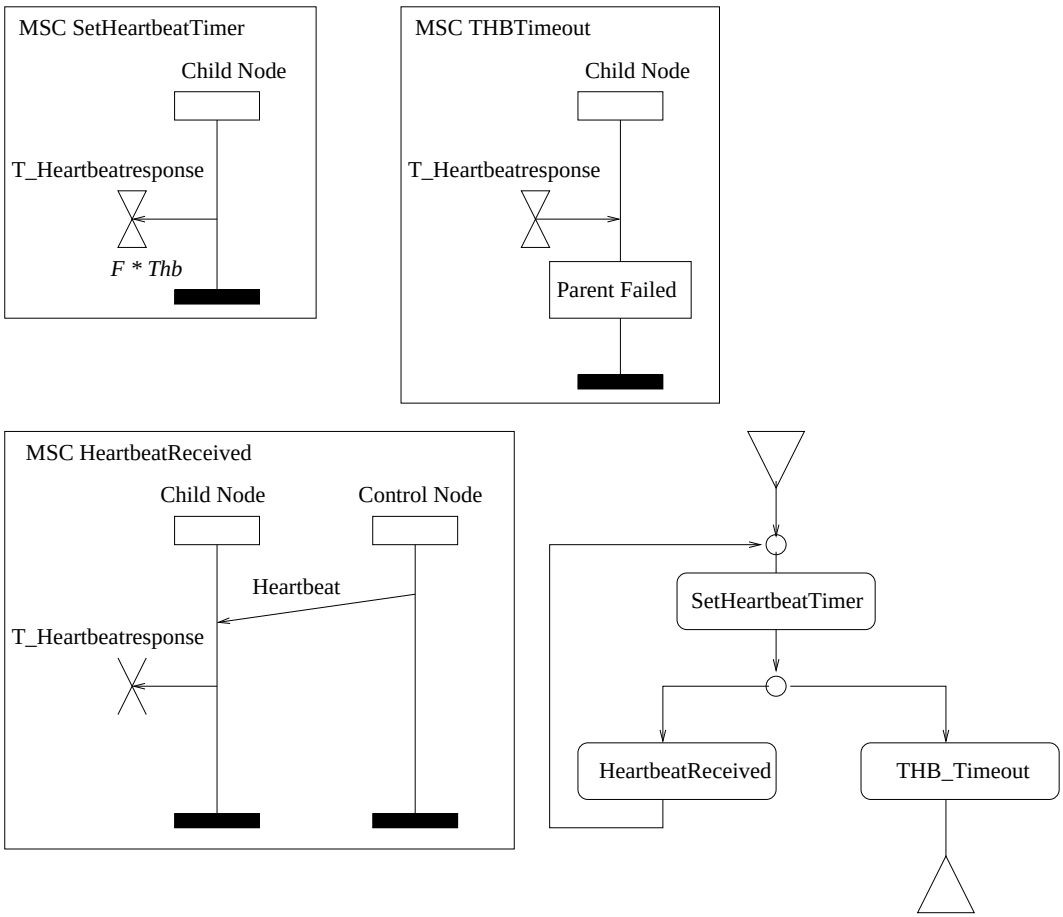

Fig. 16. Parent Failure detection

$Tjoin\_response$, for a response from the parent node. A parent node responds to a JoinStream request with a JoinAck packet or a JoinConfirm packet. If the parent node cannot process the request immediately, it responds with a JoinAck packet. Otherwise, it sends a JoinConfirm packet.

If no response is received, the node retransmits the JoinStream request. The node retransmits the JoinStream request a maximum number of times, Rjoin. If a node does not receive a response for the JoinStream requests after Rjoin tries, it reports a Parent-Unreachable failure after Rjoin retries.

If a node receives a JoinAck packet, it continues transmitting JoinStream requests at exponentially longer times, until it receives a JoinConfirm rather than a JoinAck. Every time that a node does not receive a response from its JoinStream request, it increments a local variable NumberFailures. Every time that it does not receive a JoinAck in response to its JoinStream request, it increments NumberFailures. If NumberFailures exceeds Rjoin, it reports a Parent-Unreachable failure. A node receives the tree's constant parameters and the parent's multicast control channel address from the JoinConfirm."

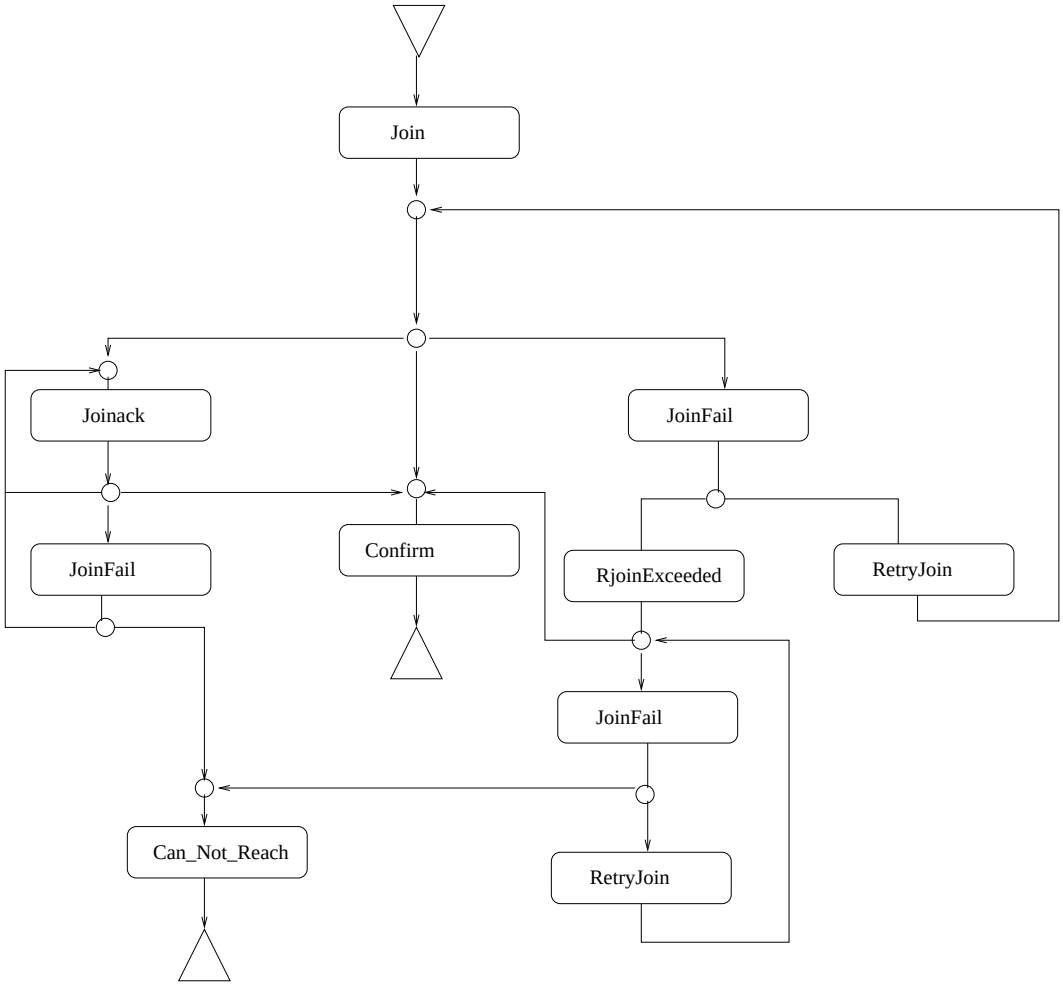

Fig. 17. Tree Connection algorithm

Figure 17 and 18 describe the connection algorithm for a node joining a stream already owned by its parent. On this example, it is easy to detect that the behavior described by MSC *Joinack* can be iterated forever. We assume that this is an error in the protocol, and that a maximal number of acknowledgments should limit the number of iteration of the behavior described in Joinack. This shows that a light formalization of textual requirements can help finding inconsistencies during the early stages of design.

5.5 Hack

The following requirements are extracted from the IETF draft: "a HACK or NACK indicates which packets have not been received. A sender retransmits the missed packets on the data channel. The time interval between successive retransmissions for a data packet is doubled for each retransmission, with an upper limit of 64 seconds (exponential backoff)." Figure 19 shows how loop generalization can be used for retransmitting lost data packets.

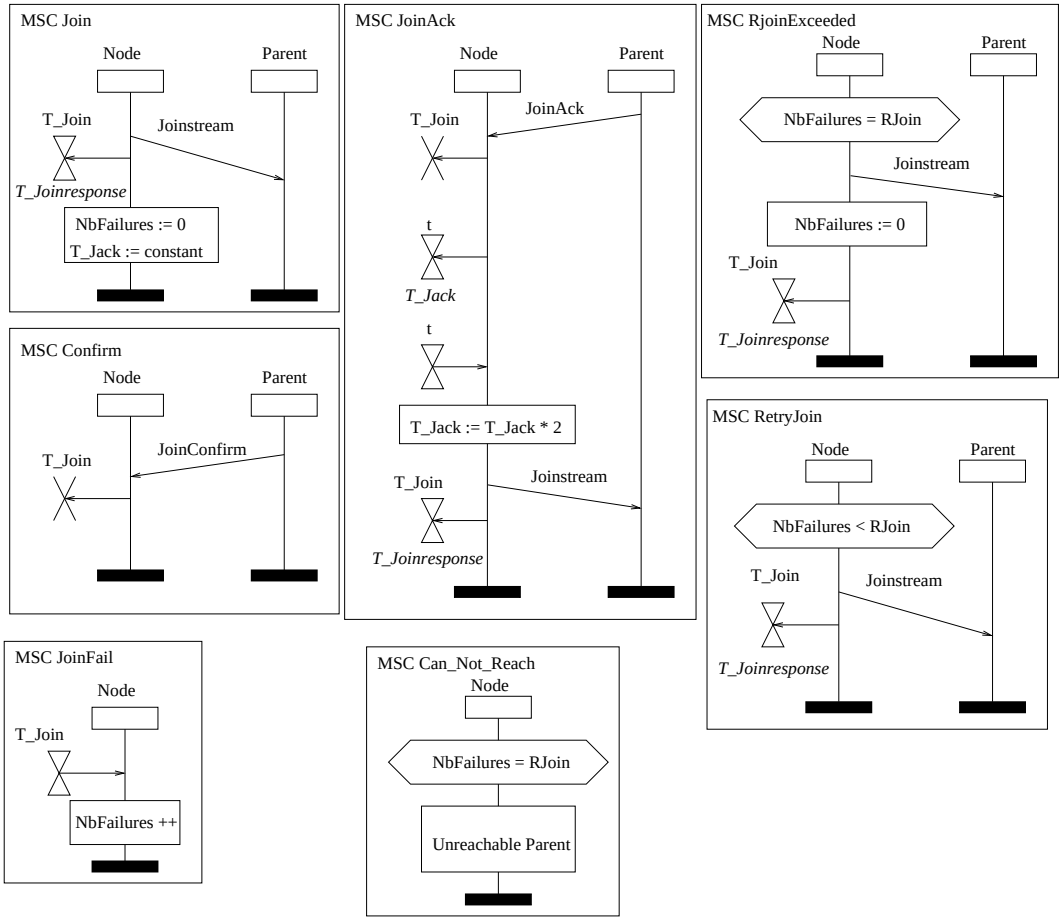

Fig. 18. MSC for Tree Connection

5.6 Expression of a QoS constraint

Data sent by the application are admitted at a variable rate. The following sentences were found in the IETF document: "Fixed, rate-based flow control limits the transmission speed to a predefined value. $Radmit\_rate\_min$: This is the minimum admission rate that the sender can use for Data packets. $Radmit\_rate\_max$: This is the maximum admission rate that the sender can use for Data packets."

The MSC of Figure 20 shows how periodicity can help expressing admission rate for one single message. Constraint on data transmission rate for more than one kind of message can be more difficult to implement, as it may involve more than one communication pattern.

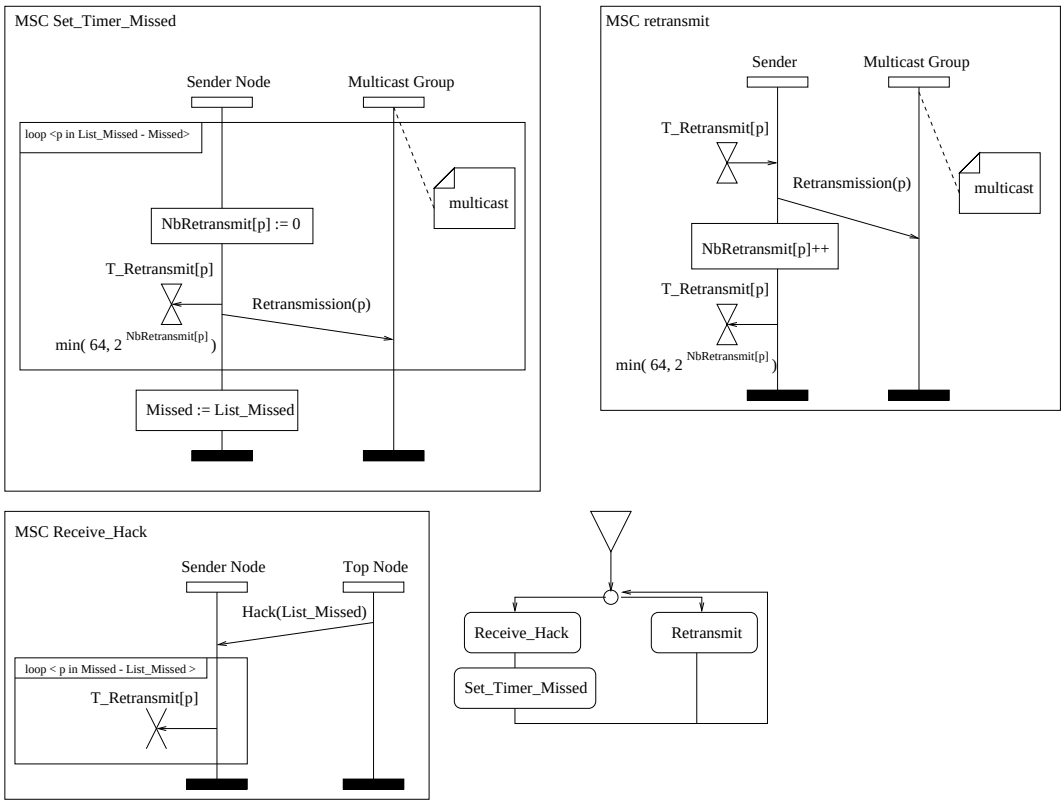

Fig. 19. Data Retransmission

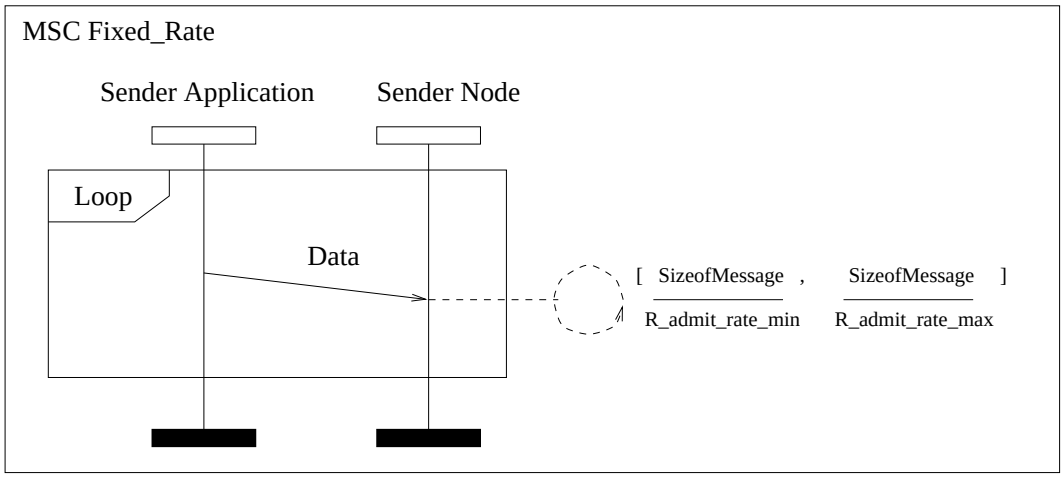

Fig. 20. Fixed Rate based Control

6 Conclusions

This study has proposed some extensions to MSC in order to model requirements of distributed systems. Improvements such as loop generalization or timer arrays are mainly syntactic extensions. On the other hand, the multicast extension is more complex, and new semantics rules have been proposed. Further studies on extensions of communications in MSC is probably needed to make sure that the rules proposed are consistent with all the other MSC constructs, but also for the integration of new communication modes such as synchronous communications.

The expression of a part of RMTP2 requirements using MSC has lead to the definition of 38 scenarios. This study has shown that early formalization of requirements can help finding inconsistencies. Of course, the error detection of section 5.4 is more due to the formalization effort than to MSC, and another language would probably have helped pointing out the same mistake. Yet, MSC appear as a good candidate for illustrating the requirements of distributed systems while allowing some formal manipulations.

Each scenario designed could be considered as a peculiar "view" of the complete RMTP2 tree behavior. From this set of scenarios, a question immediately arises : what is the set of behaviors defined by these views? A first interpretation is to consider that the behaviors defined by a set of views can be expressed as sequential and parallel compositions of the views. We think that this approach is too strict and does not really correspond to our understanding of views. The set of RMTP2 requirements defined by means of MSC may contain a lot of redundancy. The reason for this is that the initial requirements were designed according to functionalities, and also according to a level in the RMTP2 tree. An acknowledgment message from a child node to its parent may play a role for data transmission, but also for the control of the tree integrity. Hence, the combination of sets of views is far more complex than simple interleavings and serializations of behaviors. Furthermore, some views describe implementation details, such as timers use, while some other can be considered as global constraints (such as QoS requirements). An ideal composition operator should consider executions of a product of all behavioral views restricted by the constraint views.

Acknowledgments: The major part of this word has been realized within the INTERVAL european project, and during a post doctoral period at France Telecom R&D Lannion. I would also like to thank the reviewers for numerous and helpful comments on this article.

References

[1] E. Rudolph, P. Graubmann, J. Grabowski, Tutorial on Message Sequence Charts, Computer Networks and ISDN Systems 28 (12) (1996) 1629–1641.

[2] ITU-T, Z.120 : Message sequence charts (MSC) (november 1999).

[3] I. Kruger, Distributed system design with message sequence charts, Ph.D. thesis, Technische Universität München (2000).

[4] E. Rudolph, Putting extended MSC-2000 to practice: Specification of the inres connection establishment using extended MSC-2000, Draft - Technical Report.

[5] S. Loidl, E. Rudolph, U. Hinkel, MSC'96 and beyond – a critical look, in: A. Cavalli, A. Sarma (Eds.), SDL'97: Time for Testing – SDL, MSC and Trends, Proceedings of the Eighth SDL Forum, elsevier, Evry, France, 1997, pp. 213–227.

[6] H. Ben-Abdallah, S. Leue, Expressing and analyzing timing constraints in message sequence charts, Tech. Rep. 97-04, Dept. of Electrical and Computer Engineering, University of Waterloo (April 1997).

[7] L. Hélouët, Some pathological message sequence charts and how to detect them, in: R. Reed, J. Reed (Eds.), Proc. of SDL-Forum 2001: Meeting UML, no. 2078 in LNCS, 2001, pp. 348–364.

[8] T. Montgomery, B. Whetten, M. Basavaiah, S. Paul, N. Rastogi, J. Conlan, T. Yeh, The RMTP2 protocol, Ietf draft, IETF (Internet Engineering Task Force) (april 1998).

[9] H. Neukirchen, Real-time extensions to MSC-2000 and TTCN-3, Tech. rep., INTERVAL Project, http://www-interval.imag.fr/News/Slides_workshop_june_01/interestgroup_msc_ttcn.pdf (June 2001).

[10] E. Gunter, A. Muscholl, Compositionnal message sequence charts, in: Proc. of TACAS'01, no. 2031 in LNCS, 2001, pp. 496–511.

[11] S. Mauw, M. Reniers, High-level message sequence charts, in: A. Cavalli, A. Sarma (Eds.), SDL'97: Time for Testing – SDL, MSC and Trends, Proceedings of the Eighth SDL Forum, Elsevier, 1997, pp. 291–306.

[12] J. Katoen, L. Lambert, Pomsets for message sequence charts, in: Proceedings of SAM98:1st conference on SDL and MSC, Berlin, 1998, pp. 281–290.

[13] R. Alur, G. Holzmann, D. Peled, An analyzer for message sequence charts, in: TACAS'96, Tools and Algorithms for the Construction and Analysis of Systems, no. 1055 in LNCS, Springer-Verlag, 1996, pp. 35–48.