UNION INTERNATIONALE DES TÉLÉCOMMUNICATIONS

UIT-T Z.120

(11/1999)

SECTEUR DE LA NORMALISATION
DES TÉLÉCOMMUNICATIONS
DE L'UIT

SÉRIE Z: LANGAGES ET ASPECTS GÉNÉRAUX LOGICIELS DES SYSTÈMES DE TÉLÉCOMMUNICATION

Techniques de description formelle – Diagrammes des séquences de messages

Diagrammes des séquences de messages

Recommandation UIT-T Z.120

(Antérieurement Recommandation du CCITT)

RECOMMANDATIONS UIT-T DE LA SÉRIE Z

LANGAGES ET ASPECTS GÉNÉRAUX LOGICIELS DES SYSTÈMES DE TÉLÉCOMMUNICATION

TECHNIQUES DE DESCRIPTION FORMELLE	
Langage de description et de spécification (SDL)	Z.100–Z.109
Application des techniques de description formelle	Z.110–Z.119
Diagrammes des séquences de messages	**Z.120–Z.129**
LANGAGES DE PROGRAMMATION	
CHILL: le langage de programmation de l'UIT-T	Z.200–Z.209
LANGAGE HOMME-MACHINE	
Principes généraux	Z.300–Z.309
Syntaxe de base et procédures de dialogue	Z.310–Z.319
LHM étendu pour terminaux à écrans de visualisation	Z.320–Z.329
Spécification de l'interface homme-machine	Z.330–Z.399
QUALITÉ DES LOGICIELS DE TÉLÉCOMMUNICATION	Z.400–Z.499
MÉTHODES DE VALIDATION ET D'ESSAI	Z.500–Z.599

Pour plus de détails, voir la Liste des Recommandations de l'UIT-T.

Recommandation UIT-T Z.120

Diagrammes des séquences de messages

Résumé

Domaine d'application/objectif

La recommandation d'un diagramme des séquences de messages (MSC, *message sequence chart*) a pour objet de fournir un langage d'analyse pour la spécification et la description du comportement communicationnel des composantes d'un système et de leur environnement, au moyen d'un échange de messages. Etant donné que, dans un diagramme MSC, le comportement communicationnel est présenté de manière très intuitive et transparente, en particulier dans sa représentation graphique, le langage MSC est facile à apprendre, à utiliser et à interpréter. En association avec d'autres langages, on peut l'utiliser pour exprimer des méthodes de spécification, de conception, de simulation, d'essais et de documentation de système.

Contenu

La présente Recommandation présente une définition syntaxique des diagrammes des séquences de messages en mode texte et en mode graphique, complétée d'une description sémantique non formelle.

Application

Les diagrammes MSC ont un domaine d'application étendu. Ils ne sont pas conçus pour un seul domaine d'application. Un mode d'application important des diagrammes MSC consiste à spécifier un aperçu général du comportement communicationnel pour des systèmes en temps réel, plus particulièrement des systèmes de commutation en télécommunication. Grâce aux diagrammes MSC, l'on peut spécifier des traces d'un système, choisies principalement sous forme de cas de figure types, à partir desquels l'on peut construire des cas atypiques décrivant des comportements exceptionnels. Les diagrammes MSC peuvent donc être utilisés pour spécifier des prescriptions, des interfaces, des simulations, des validations, des cas de test et des documentations pour systèmes en temps réel. Ils peuvent être employés en association avec d'autres langages de spécification, en particulier le langage SDL. Dans ce contexte, les diagrammes MSC offrent également une base pour la conception de systèmes SDL.

Statut et stabilité

Le diagramme MSC est stable. La présente Recommandation est une suite logique de l'édition de 1996, à laquelle elle ajoute des concepts pour:

- les données;
- les durées;
- les fluences;
- l'orientation objet pour les diagrammes MSC.

Travaux associés

– Recommandation UIT-T Q.65 (1997), *Méthode fonctionnelle unifiée de caractérisation des services et des capacités des réseaux*.

– Recommandation UIT-T X.210 (1993) | ISO/CEI 10731:1994, *Technologies de l'information – Interconnexion de systèmes ouverts – Modèle de référence de base: conventions pour la définition des services de l'interconnexion de systèmes ouverts*.

– Recommandation UIT-T X.292 (1998), *Cadre et méthodologie des tests de conformité OSI pour les Recommandations sur les protocoles pour les applications de l'UIT-T – Notation combinée arborescente et tabulaire (TTCN)*.

– Recommandation UIT-T Z.100 (1999), *Langage de description et de spécification*.

– UML 1.2, OMG 1999.

Source

La Recommandation Z.120 de l'UIT-T, révisée par la Commission d'études 10 (1997-2000) de l'UIT-T, a été approuvée le 19 novembre 1999 selon la procédure définie dans la Résolution 1 de la CMNT.

AVANT-PROPOS

L'UIT (Union internationale des télécommunications) est une institution spécialisée des Nations Unies dans le domaine des télécommunications. L'UIT-T (Secteur de la normalisation des télécommunications) est un organe permanent de l'UIT. Il est chargé de l'étude des questions techniques, d'exploitation et de tarification, et émet à ce sujet des Recommandations en vue de la normalisation des télécommunications à l'échelle mondiale.

La Conférence mondiale de normalisation des télécommunications (CMNT), qui se réunit tous les quatre ans, détermine les thèmes d'étude à traiter par les Commissions d'études de l'UIT-T, lesquelles élaborent en retour des Recommandations sur ces thèmes.

L'approbation des Recommandations par les Membres de l'UIT-T s'effectue selon la procédure définie dans la Résolution 1 de la CMNT.

Dans certains secteurs des technologies de l'information qui correspondent à la sphère de compétence de l'UIT-T, les normes nécessaires se préparent en collaboration avec l'ISO et la CEI.

NOTE

Dans la présente Recommandation, l'expression "Administration" est utilisée pour désigner de façon abrégée aussi bien une administration de télécommunications qu'une exploitation reconnue.

DROITS DE PROPRIÉTÉ INTELLECTUELLE

L'UIT attire l'attention sur la possibilité que l'application ou la mise en œuvre de la présente Recommandation puisse donner lieu à l'utilisation d'un droit de propriété intellectuelle. L'UIT ne prend pas position en ce qui concerne l'existence, la validité ou l'applicabilité des droits de propriété intellectuelle, qu'ils soient revendiqués par un Membre de l'UIT ou par une tierce partie étrangère à la procédure d'élaboration des Recommandations.

A la date d'approbation de la présente Recommandation, l'UIT [avait/n'avait pas] été avisée de l'existence d'une propriété intellectuelle protégée par des brevets à acquérir pour mettre en œuvre la présente Recommandation. Toutefois, comme il ne s'agit peut-être pas de renseignements les plus récents, il est vivement recommandé aux responsables de la mise en œuvre de consulter la base de données des brevets du TSB.

© UIT 2001

Droits de reproduction réservés. Aucune partie de cette publication ne peut être reproduite ni utilisée sous quelque forme que ce soit et par aucun procédé, électronique ou mécanique, y compris la photocopie et les microfilms, sans l'accord écrit de l'UIT.

TABLE DES MATIÈRES

		Page
1	Introduction	1
1.1	Objectifs des diagrammes MSC	1
1.2	Organisation de la présente Recommandation	1
1.3	Métalangage pour la grammaire textuelle	2
1.4	Métalangage pour la grammaire graphique	3
2	Règles générales	6
2.1	Règles lexicales	6
2.2	Règles de visibilité et de nommage	11
2.3	Commentaire	12
2.4	Règles de dessin	13
2.5	Mise en page des diagrammes MSC	14
3	Document de diagramme des séquences de messages	15
4	Diagramme MSC de base	17
4.1	Diagramme des séquences de messages	17
4.2	Instance	23
4.3	Message	24
4.4	Fluences	28
4.5	Environnement et portes	33
4.6	Relation d'ordre général	40
4.7	Condition	42
4.8	Temporisateur	44
4.9	Action	47
4.10	Création d'instance	48
4.11	Terminaison d'instance	49
5	Concepts relatifs aux données	49
5.1	Introduction	49
5.2	Interface syntaxique avec des langages externes de données	49
5.3	Interface sémantique avec des langages externes de données	51
5.4	Déclaration de données	53
5.5	Données statiques	55
5.6	Données dynamiques	56
5.7	Associations	57
5.8	Données dans les paramètres de message et de temporisation	59
5.9	Données dans les paramètres de création d'instance	61

		Page
5.10	Données dans les cadres d'action	61
5.11	Types de données supposés	62
6	Concepts temporels	62
6.1	Sémantique temporisée	63
6.2	Temporisation relative	64
6.3	Temporisation absolue	64
6.4	Domaine temporel	64
6.5	Variables temporelles statiques et dynamiques	64
6.6	Décalage temporel	64
6.7	Points, mesures et intervalles temporels	65
6.8	Points temporels	65
6.9	Mesures	65
6.10	Intervalles temporels	66
7	Concepts structurels	69
7.1	Corégion	69
7.2	Expression en ligne	70
7.3	Référence MSC	74
7.4	Décomposition d'instance	79
7.5	Diagramme MSC de haut niveau (HMSC)	85
8	Document de diagramme des séquences de messages	89
8.1	Documents MSC	89
8.2	Décomposition d'instance	90
8.3	Héritage d'instance	92
9	Diagrammes MSC simples	93
9.1	Diagramme MSC de base	93
9.2	Dépassement de message	94
9.3	Concepts de base de diagramme MSC	95
9.4	Composition de diagramme MSC par étiquetage de conditions	96
9.5	Diagramme MSC avec supervision temporelle	98
9.6	Diagramme MSC avec perte de message	99
9.7	Conditions locales	100
10	Données	101
11	Temps	104
12	Processus de création et de terminaison	108

Page

13 Corégion 109

14 Relation d'ordre général 110

14.1 Relation d'ordre généralisée dans une corégion 110

14.2 Relation d'ordre généralisé entre différentes instances 110

15 Expressions en ligne 111

15.1 Expression en ligne avec composition en alternative 111

15.2 Expression en ligne avec portes 114

15.3 Expression en ligne avec composition en parallèle 115

16 Références de diagramme MSC 116

16.1 Référence MSC 116

16.2 Référence MSC avec porte 118

17 Diagramme MSC de haut niveau 118

17.1 MSC de haut niveau avec boucle libre 118

17.2 MSC de haut niveau avec boucle 119

17.3 MSC de haut niveau avec composition en alternative 120

17.4 MSC de haut niveau avec composition en parallèle 121

Annexe A – Index 123

Recommandation UIT-T Z.120

DIAGRAMMES DES SÉQUENCES DE MESSAGES

(révisée en 1999)

1 Introduction

1.1 Objectifs des diagrammes MSC

Les diagrammes des séquences de messages (MSC) constituent un langage décrivant l'interaction entre un certain nombre d'instances indépendantes échangeant des messages. Les principales caractéristiques du langage MSC sont les suivantes:

- MSC est un langage par scénario qui décrit l'ordre dans lequel interviennent les communications et d'autres événements. Il permet en outre d'exprimer des restrictions applicables aux valeurs des données transmises et à la temporisation des événements.
- MSC prend en charge les spécifications complètes ou incomplètes. Il permet de décrire des comportements incomplets qui servent à une préanalyse ou à des fins documentaires.
- MSC est un langage graphique. Les diagrammes bidimensionnels donnent un aperçu général du comportement d'instances communicantes. La forme textuelle des diagrammes MSC vise principalement aux échanges entre outils. Elle forme la base de l'analyse formelle automatique.
- MSC est un langage formel dont la définition est donnée en langage naturel ainsi qu'en notation formelle.
- MSC est un langage pratique qui est utilisé d'un bout à l'autre du processus d'ingénierie. Son usage va de l'analyse de domaine et de l'idéation jusqu'aux essais en passant par les phases de saisie des exigences et de conception. Le langage MSC est utilisé de façon légèrement différente au cours de ces diverses phases et il importe qu'il ait une puissance d'expression formelle ainsi qu'une allure intuitive.
- MSC est largement applicable. Il n'est pas défini pour un seul domaine d'application.
- MSC prend en charge la conception structurée. L'on peut combiner des scénarios simples (décrits par les diagrammes des séquences de messages de base) pour former des spécifications plus complètes au moyen de diagrammes des séquences de messages de haut niveau. Les diagrammes MSC sont réunis dans un document MSC. Une conception modulaire des scénarios est assurée par les mécanismes de décomposition et de réutilisation.
- MSC est souvent utilisé en association avec d'autres méthodes et d'autres langages. Sa définition formelle permet une validation formelle et automatisée d'un diagramme MSC par rapport à un modèle décrit dans un autre langage. Le langage MSC peut par exemple être utilisé en combinaison avec le langage SDL et la notation TTCN.
- L'interprétation habituelle d'un scénario spécifié dans un diagramme MSC est que la réalisation concrète doit au moins manifester le comportement exprimé dans ce scénario. Des interprétations différentes sont également possibles. Un diagramme MSC peut, par exemple, servir à spécifier des scénarios déconseillés.

1.2 Organisation de la présente Recommandation

La présente Recommandation est structurée comme suit. Le paragraphe 2 décrit les règles générales concernant la syntaxe, le dessin et la mise en page. Le paragraphe 3 donne une définition du document MSC. Le paragraphe 4 contient la définition des diagrammes des séquences de messages

et leurs constituants de base: instance, message, relation d'ordre général, condition, temporisateur, action, création et terminaison d'instance. Le paragraphe 5 contient les concepts de données et le paragraphe 6 définit les concepts de temps. Le paragraphe 7 présente les concepts de haut niveau concernant la structuration et la modularisation. Ces concepts permettent une spécification en hiérarchie inverse ainsi qu'un affinement d'instances individuelles au moyen de la création d'une corégion (voir 7.1) et de la décomposition d'instance (voir 7.4). Les expressions en ligne sont définies au 7.2 et les références MSC au 7.3. Le langage MSC de haut niveau (voir 7.5) permet de composer et de réutiliser tout (ou partie) des diagrammes MSC à différents niveaux. La Partie II donne des exemples de toutes les constructions de diagrammes MSC. L'Annexe A contient un index renvoyant aux mots clés <keyword> et aux non terminaux.

1.3 Métalangage pour la grammaire textuelle

En formalisme de Backus-Naur (BNF, *Backus-Naur form*), on indique un symbole terminal soit en évitant de l'encadrer par des chevrons (c'est-à-dire le signe < et le signe >) soit en le représentant sous l'une des deux formes suivantes: <name> ou <character string>.

Les chevrons et le ou les mots qui y sont contenus représentent soit un symbole non terminal soit l'un des deux terminaux <character string> ou <name>. Les catégories syntaxiques sont les non terminaux indiqués par un ou plusieurs mots entre crochets. Pour chaque symbole non terminal, une règle de production est donnée soit en grammaire textuelle concrète soit en grammaire graphique concrète. Par exemple:

```
<instance parameter decl> ::=
                    inst <instance parm decl list> <end>
```

Une règle de production pour les symboles non terminaux est composée d'un symbole non terminal en membre gauche du symbole ::=, et d'une ou plusieurs constructions, composée(s) de symboles non terminaux ou terminaux en membre droit. Par exemple, <instance parameter decl> et <instance parm decl list> et <end> dans l'exemple ci-dessus sont des non terminaux; **inst** est un symbole terminal.

Parfois une partie du symbole est soulignée. Cette partie soulignée met en évidence l'aspect sémantique de ce symbole: par exemple, le symbole <msc name> est syntaxiquement identique à <name>, mais sémantiquement il faut que le nom soit un nom de diagramme des séquences de messages.

Le membre droit du symbole ::= peut être composé de plusieurs variantes de production, séparées par une barre verticale (|). Par exemple:

```
<incomplete message area> ::=
                    <lost message area>
                    |       <found message area>
```

exprime qu'une région de message incomplète <incomplete message area> est soit une région de message perdu <lost message area> soit une région de message trouvé <found message area>.

Les éléments syntaxiques peuvent être regroupés au moyen d'accolades ({ et }). Un groupe entre accolades peut contenir une ou plusieurs barres verticales indiquant un choix entre éléments syntaxiques. Par exemple:

```
<msc document body> ::=
                    { <message sequence chart> | <msc diagram> }*
```

La répétition de groupes entre accolades est indiquée par un astérisque (*) ou par le signe d'addition (+). Un astérisque indique que le groupe est facultatif et peut être répété un nombre quelconque de fois; un signe + indique que le groupe doit être présent et qu'il peut être répété un nombre quelconque de fois. L'exemple ci-dessus signifie qu'un corps de document <msc document body> peut être vide mais peut aussi contenir un nombre quelconque de diagrammes <message sequence chart> et/ou de diagrammes <msc diagram>.

Si des éléments syntaxiques sont groupés au moyen de crochets ([et]), ce groupe est facultatif. Par exemple:

```
<identifier> ::=
                    [ <qualifier> ] <name>
```

indique qu'un identificateur peut contenir un qualificatif mais qu'il n'y est pas contraint.

Les opérateurs de métalangage présentent l'ordre d'importance suivant. L'opérateur qui établit la liaison la plus faible est l'opérateur de choix "|". Vient ensuite l'opérateur de séquencement de gauche à droite. Les opérateurs "+" et "*" établissent une liaison plus forte que l'opérateur de séquencement et enfin les crochets "[...]" et les accolades {...} établissent la liaison la plus forte.

1.4 Métalangage pour la grammaire graphique

Peu de constructions constituent le métalangage de la grammaire graphique. Elles sont décrites de manière informelle ci-dessous. La syntaxe graphique n'est pas assez précise pour décrire les éléments graphiques. C'est pour cela qu'il n'y a pas de variations graphiques. De petites variations dans les formes réelles des symboles pour terminaux graphiques sont autorisées. Il s'agira, par exemple, du hachurage des symboles pleins, de la forme d'une pointe de flèche et de la taille relative d'éléments graphiques. Chaque fois que cela sera nécessaire, la syntaxe graphique sera complétée d'une explication informelle sur les représentations graphiques des constructions.

Le métalangage consiste en une sorte de notation BNF composée de métaconstructions spéciales: ***contains***, ***is followed by***, ***is associated with***, ***is attached to***, ***above*** et ***set***. Ces constructions se comportent comme les règles de production BNF normales, mais impliquent en plus des relations logiques ou géométriques entre les arguments. Le comportement de la construction ***is attached to***, expliqué ci-dessous, est un peu différent. Le membre gauche de toutes les constructions doit être un symbole sauf ***above***. Un symbole est un non terminal produisant dans chaque séquence de production exactement un terminal graphique et un seul. Nous considérerons un symbole qui est attaché à (***is attached to***) d'autres régions ou qui est associé à (***is associated with***) une chaîne de texte comme étant également un symbole. L'explication est informelle et le métalangage ne décrit pas avec précision les dépendances géométriques.

contains:

"<area1> ***contains*** <area2>" signifie que <area2> est contenu dans <area1>, géométriquement en principe, mais aussi logiquement.

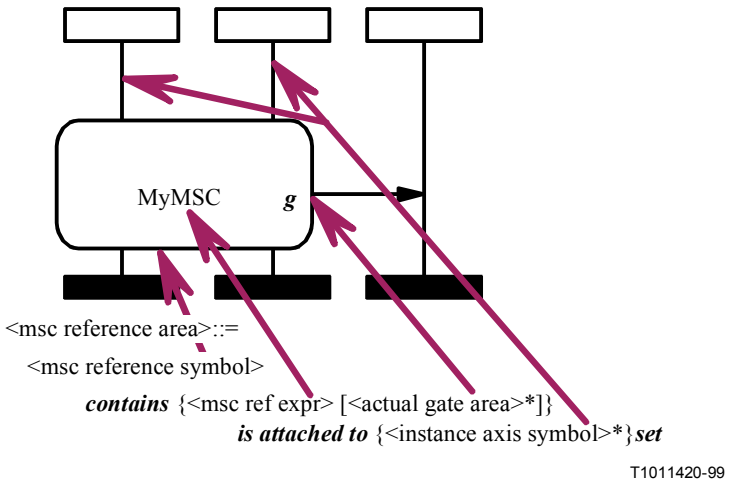


Figure 1/Z.120 – Exemple pour 'contains'

is followed by:

"<area1> ***is followed by*** <area2>" signifie que <area2> est logiquement et géométriquement rattaché à <area1>.

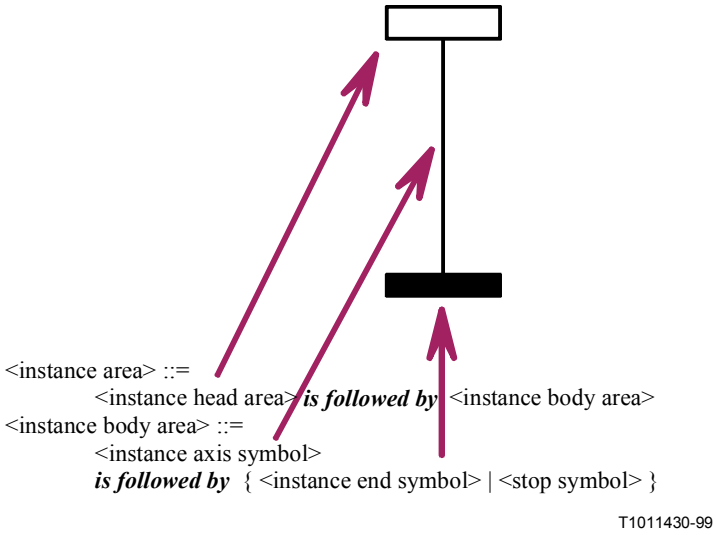


Figure 2/Z.120 – Exemple pour 'is followed by'

is associated with:

"<area1> ***is associated with*** <area2>" signifie que <area2> est un texte affilié à <area1>. Il n'y a pas d'association géométrique plus étroite entre les régions que lorsqu'elles sont associées l'une avec l'autre.

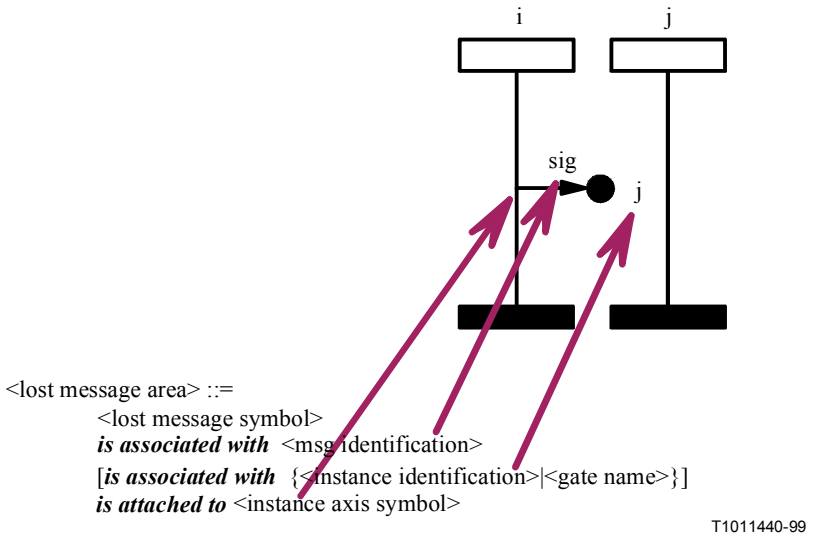


Figure 3/Z.120 – Exemple pour 'is associated with'

is attached to:

"<area1> ***is attached to*** <area2>" n'est pas une règle de production BNF normale et son utilisation est restreinte. <area2> doit être un symbole ou un ensemble de symboles. La règle de production "<P>::=<area1> ***is attached to*** <area2>" signifie que, si le symbole non terminal <P> est dérivé au

moyen de cette règle, seul le symbole <area1> est produit. Plutôt que de produire également <area2>, une occurrence de <area2> est identifiée comme ayant été produite par une autre règle. Le résultat de la construction ***is attached to*** est une relation logique et géométrique entre <area1> et <area2>. Si le membre droit est un ensemble de symboles, il y a une relation entre <area1> et tous les éléments de l'ensemble.

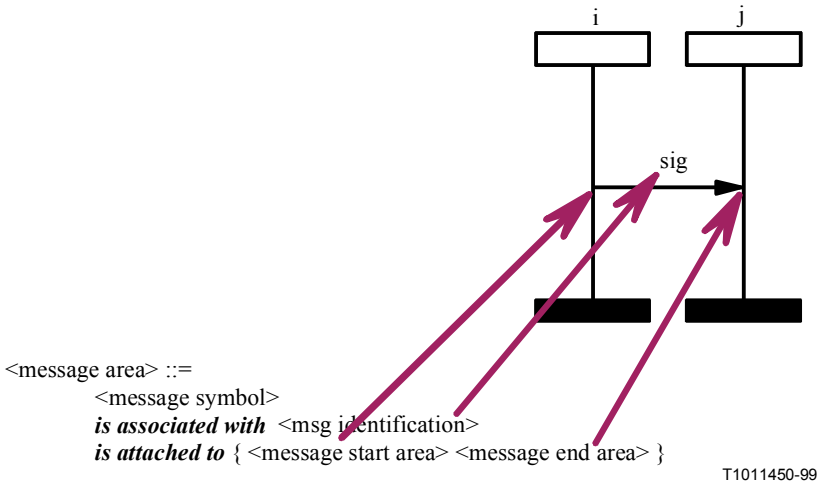


Figure 4/Z.120 – Exemple pour 'is attached to'

Notons que si le symbole A ***is attached to*** symbole B, alors le symbole B est aussi ***attached to*** symbole A. Dorénavant, un attachement entre deux symboles est toujours défini deux fois dans la grammaire. Nous suivons l'un des non terminaux ci-dessus pour obtenir ceci:

<message end area> ::=
<message in area> | <gate in area> | <gate out def area> | <inline expr>

<message in area> ::=
<message in symbol>
is attached to <instance axis symbol>
is attached to <message symbol>

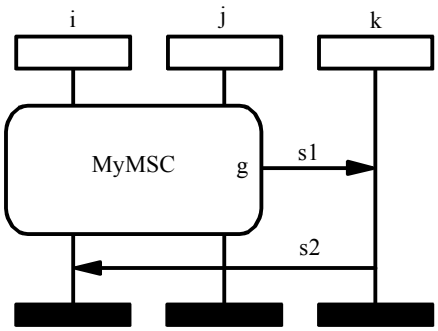
<message in symbol> ::=
<void symbol>

Ainsi un symbole <message symbol> ***is attached to*** un symbole <message in symbol> et vice versa.

La construction "***is attached to***" peut être spécialisée par insertion en préfixe ou en appendice d'indications sur l'emplacement du rattachement de la construction. Les modificateurs peuvent être ***top, bottom, left, right*** ou une combinaison d'entre eux. C'est-à-dire que nous pouvons avoir des constructions telles que "<area1> ***bottom is attached to top or right*** <area2>", ce qui signifie que la partie inférieure du symbole <area1> est attachée logiquement et géométriquement au sommet ou à la partie droite du symbole <area2>.

above:

"<area1> ***above*** <area2>" signifie que <area1> et <area2> sont logiquement ordonnés. Géométriquement, on exprime cette idée en ordonnant verticalement <area1> et <area2>. Si deux symboles <area> ont les mêmes coordonnées verticales, alors aucun ordre entre eux n'est défini, mis à part le fait qu'un envoi de message doit avoir lieu avant une réception.



<event layer> ::=
<event area> | <event area> ***above*** <event layer>

T1011460-99

Figure 5/Z.120 – Exemple pour 'above'

La Figure 5 décrit une séquence d'événements. Lorsque ces événements sont sur des instances différentes, la coordonnée géométrique verticale n'a aucune signification sémantique. La Figure 5 décrit la séquence suivante:

i,j: ***reference*** mr1: MyMSC ***gate*** g ***output*** s1 ***to*** k;

k: ***input*** s1 ***from*** mr1 ***via*** g;

k: ***output*** s2 ***to*** i;

i: ***input*** s2 ***from*** k.

set

Le métasymbole ***set*** est un opérateur postfixé portant sur les éléments syntaxiques entre accolades se trouvant immédiatement avant et indiquant un ensemble (non ordonné) d'éléments. Chaque élément peut être un groupe d'éléments syntaxiques, auquel cas il doit être développé avant l'application du métasymbole ***set.***

<text layer> ::=
{<text area>*} ***set***

est un ensemble de 0 ou plusieurs symboles <text area>.

<msc body area> ::=
{ <instance layer> <text layer> <gate def layer>
<event layer> <connector layer> } ***set***

est un ensemble non ordonné des éléments entre accolades.

2 Règles générales

2.1 Règles lexicales

Les règles lexicales définissent des unités lexicales. Les unités lexicales sont des symboles terminaux de la syntaxe textuelle concrète (*concrete textual syntax).*

<lexical unit> ::=
<word>
| <character string>
| <special>
| <composite special>
| <note>
| <keyword>

<word> ::=
{<alphanumeric> | <full stop>}*
<alphanumeric>
{<alphanumeric> | <full stop>}*

<alphanumeric> ::=
<letter>
| <decimal digit>
| <national>

<letter> ::=
A | B | C | D | E | F | G | H | I | J | K | L | M
| N | O | P | Q | R | S | T | U | V | W | X | Y | Z
| a | b | c | d | e | f | g | h | i | j | k | l | m
| n | o | p | q | r | s | t | u | v | w | x | y | z

<decimal digit> ::=
0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<national> ::=
| ` | ¤ | @ | \
| <left square bracket>
| <right square bracket>
| <left curly bracket>
| <vertical line>
| <right curly bracket>
| <overline>
| <upward arrow head>

<left square bracket> ::=
[

<right square bracket> ::=
]

<left curly bracket> ::=
{

<vertical line> ::=
|

<right curly bracket> ::=
}

<left open> ::=
(

<left closed> ::=
<left square bracket>

<right open> ::=
)

<right closed> ::=
<right square bracket>

<abs time mark> ::=
@

<rel time mark> ::=
&

<overline> ::=
~

<upward arrow head> ::=
^

<full stop> ::=
.

<underline> ::=
_

<left angular bracket> ::=
<

<right angular bracket> ::=
>

<character string> ::=
<apostrophe>{<alphanumeric>
| <other character>
| <special>
| <full stop>
| <underline>
| <space>
| <apostrophe><apostrophe>}* <apostrophe>

<text> ::=
{ <alphanumeric>
| <other character>
| <special>
| <full stop>
| <underline>
| <space>
| <apostrophe> }*

<apostrophe> ::=
'

<other character> ::=
? | & | % | + | - | ! | / | > | * | " | < | =

<special> ::=
(|) | , | ; | :

<composite special> ::=
<qualifier left> | <qualifier right>

<qualifier left> ::=
<<

<qualifier right> ::=
>>

<note> ::=
/* <text> */

<name> ::=
{<letter> | <decimal digit> | <underline> | <full stop>}+

<keyword> ::=
action
| **after**
| **all**
| **alt**
| **as**
| **before**
| **begin**
| **block**
| **by**
| **call**
| **comment**
| **concurrent**
| **condition**
| **connect**
| **create**
| **data**
| **decomposed**

| **def**
| **empty**
| **end**

endafter
endbefore
endconcurrent
endexpr
endinstance
endmethod
endmsc
endsuspension
env
equalpar
escape
exc
expr
external
finalized
found
from
gate
in
inf
inherits
inline
inst
instance
int_boundary
label
language
loop
lost
method
msc
mscdocument
msg
nestable
nonnestable
offset
opt
order
otherwise
out
par
parenthesis
process
receive
redefined
reference
related
replyin
replyout
seq
service
shared
startafter
startbefore
starttimer
stop
stoptimer
suspension
system
text
time
timeout

| **timer**
| **to**
| **undef**
| **using**
| **utilities**
| **variables**
| **via**
| **virtual**
| **when**
| **wildcards**

<space> ::=
caractère "espace" de l'alphabet n° 5 de l'UIT-T

Les caractères nationaux <national> sont représentés ci-dessus comme dans la version internationale de référence de l'alphabet n° 5 du CCITT (Recommandation T.50). La responsabilité de la définition des représentations nationales de ces caractères relève des organismes nationaux de normalisation.

Les caractères de commande sont définis conformément à la Recommandation T.50. Une séquence de caractères de commande peut apparaître là où un espace <space> peut apparaître, et a la même signification qu'un espace <space>. L'espace <space> représente le caractère "espace" de l'alphabet n° 5 du CCITT.

L'occurrence d'un caractère de commande n'est pas significative dans une <note>. Un caractère de commande ne peut pas apparaître dans une chaîne de caractères si sa présence est significative.

Dans toutes les unités lexicales <lexical unit>, les lettres majuscules et minuscules sont distinctes, c'est-à-dire que AB, ab, Ab et aB sont quatre unités lexicales différentes.

Une unité lexicale <lexical unit> se termine par le premier caractère qui ne peut pas faire partie de l'unité <lexical unit> conformément à la syntaxe spécifiée ci-dessous. Lorsqu'un caractère souligné <underline> est suivi par un ou plusieurs caractères <space>, tous ces caractères (y compris le <underline>) sont ignorés, par exemple A_B définit le même nom <name> que AB. Cette utilisation de <underline> permet de répartir les unités <lexical unit> sur plus d'une ligne.

Le caractère / immédiatement suivi par le caractère * en dehors d'une <note> marque le début d'une <note>. Le caractère * immédiatement suivi par le caractère / dans une <note> termine toujours la <note>. Une <note> peut être insérée avant ou après toute unité <lexical unit>.

Une chaîne de données est une chaîne de caractères qui sera analysée par un analyseur externe au langage MSC. L'analyseur lexical MSC doit renvoyer la chaîne de données sous la forme d'un jeton unique qui sera à son tour transmis à l'analyseur externe.

Etant donné qu'il serait préférable que la chaîne de données puisse être distinguée du langage MSC et du langage de données sans un certain nombre de caractères délimiteurs supplémentaires, la grammaire MSC définit des concepts permettant de traduire directement la plupart des langages de données.

Trois concepts de parenthèses sont définis: les parenthèses normales qui peuvent être imbriquées, c'est-à-dire que des parenthèses peuvent contenir d'autres parenthèses; les parenthèses non imbricables qui ne peuvent être contenues dans d'autres parenthèses sans que leur contenu soit considéré comme nul; et les parenthèses symétriques, dont les deux éléments – d'ouverture et de fermeture – sont égaux. Les délimiteurs de parenthèses sont eux-mêmes déclarés dans l'en-tête du document MSC.

Pour rendre cette méthode applicable dans toutes les circonstances, un échappement est également défini afin de décrire les cas où les délimiteurs de parenthèses ne doivent pas être considérés comme tels.

<string> ::=
<pure data string>

<pure data string> ::=
<non-parenthesis> [<parenthesis>] [<pure data string>]

<parenthesis> ::=
<nestable par> | <equal par> | <non-nestable par>

<nestable par> ::=
<open par> <pure data string> <close par>

Les symboles <open par> et <close par> doivent être les délimiteurs des parenthèses correspondantes, définis sous forme d'une paire de parenthèses dans l'en-tête du document MSC défini au 5.2.

<non-parenthesis> ::=
{<non-par-non-escape> |
<escapechar> {<escapechar> | <open par> | <close par>} }*

<non-par-non-escape> ::=
character string containing no escape char and no paranthesis delimiters and
not the possible terminators","")";" ";" ";=" "=:" or **comment**

<equal par> ::=
<delim> <unmatched string> <delim>

<unmatched string> ::=
<non-parenthesis> [<nestable par>] [<unmatched string>]

<non-nestable par> ::=
<open par> <text> <close par>

Les symboles <open par> et <close par> doivent être les délimiteurs des parenthèses correspondantes, définis sous forme d'une paire de parenthèses dans l'en-tête du document MSC.

Si la chaîne <string> doit contenir les caractères par lesquels est normalement censée finir une chaîne <non-par-non-escape>, ces caractères doivent figurer entre parenthèses ou donner lieu à une séquence d'échappement.

2.2 Règles de visibilité et de nommage

Les entités sont identifiées et référencées au moyen des noms correspondants. Les entités sont groupées par classes d'entités pour permettre une plus grande souplesse des règles de nommage. Les classes d'entités sont les suivantes:

a) document MSC;

b) diagramme MSC;

c) instance;

d) condition;

e) temporisateur;

f) message;

g) porte;

h) étiquette d'intervalle temporel divisé;

i) variable;

j) nom général.

Les unités de portée désignent l'ensemble du système de documents MSC: un seul document MSC ou un seul diagramme MSC.

L'ensemble du système de documents MSC est le domaine de définition des noms de document MSC (qui sont équivalents à des noms de sorte d'instance).

Le document MSC est le domaine de définition des diagrammes MSC, des instances, des conditions, des temporisateurs, des messages, des étiquettes d'intervalle temporel divisé, des variables et des noms généraux.

Le diagramme MSC est le domaine de définition des portes et des paramètres MSC formels. Les paramètres formels ont priorité sur les entités définies dans le document MSC.

Une instance sans sorte explicite prendra son nom comme sorte implicite.

Les messages à destination ou en provenance d'instances décomposées doivent être redéclarés avec le même nom et la même signature à l'intérieur du document MSC définissant la décomposition. Une signature de message se compose des types et de l'ordre des paramètres de ce message.

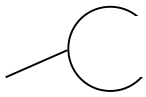
Une porte est référencée depuis l'extérieur de son diagramme MSC définisseur et seulement dans une référence à ce diagramme MSC.

En notation textuelle, il est parfois nécessaire d'introduire des identificateurs spécifiques pour des objets individuels qui n'ont pas besoin d'un nommage explicite dans la notation graphique. Il s'agira par exemple de références de diagrammes MSC dans lesquelles la notation graphique pourra facilement désigner deux ou plus de deux occurrences de la même référence MSC, alors que la notation textuelle doit les distinguer au moyen de leurs propres identificateurs particuliers.

Bien que l'on puisse utiliser des lignes et la proximité spatiale pour associer des constructions dans une description graphique, une notation textuelle nécessitera des identificateurs à cette même fin. Dans certains cas, la notation graphique nécessitera aussi des identificateurs d'entités afin de décrire une relation dont la répartition spatiale est étendue. De tels identificateurs peuvent assortir un certain nombre de constructions. Si la référence de diagramme MSC contient un identificateur, les portes peuvent être désignées par des noms, qui sont soit contenus dans un symbole (comme la référence MSC) adjacent dans l'espace et décrit dans la grammaire graphique sous la forme ***is associated with*** (comme les portes de message dans un diagramme), soit contenus dans un symbole de nommage spécial qui est associé à la construction (comme pour des régions d'événement d'instance). Lorsque des portes n'ont pas de nom explicite, un nom implicite est construit.

<general name area> ::=
<general name symbol> ***contains*** <name>
is attached to {<instance event area> | <inline expression area> |
<operand area>}

<general name symbol> ::=



Le symbole de nom général <general name symbol> peut également être copié en miroir. Le nom contenu dans ce symbole est inscrit dans l'arc de cercle mais peut s'étendre au-delà du symbole en passant par l'ouverture. La zone de nom général <general name area> est associée à d'autres constructions par la ligne droite.

2.3 Commentaire

Il existe trois sortes de commentaires.

Tout d'abord, il y a la note (*note*), qui n'apparaît que dans la syntaxe textuelle (voir les règles lexicales).

En deuxième lieu, il y a le commentaire (*comment*), qui est une notation représentant des explications non formelles, associées aux symboles ou au texte.

La syntaxe textuelle concrète (*concrete textual syntax*) des commentaires est la suivante:

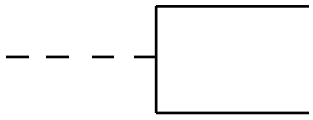
<end> ::=
[<comment>]**;**

<comment> ::=
comment <character string>

La syntaxe suivante est utilisée dans la grammaire graphique concrète (*concrete graphical grammar*):

<comment area> ::=
<comment symbol> ***contains*** <text>

<comment symbol> ::=



Un symbole de commentaire <comment symbol> peut être associé aux symboles graphiques.

En troisième lieu, il y a les commentaires de type texte (*text*), qui peuvent être utilisés afin de formuler des commentaires généraux.

Le texte contenu dans la grammaire textuelle (*textual grammar*) est défini comme suit:

<text definition> ::=
text <character string> <end>

Le texte contenu dans la grammaire graphique (*graphical grammar*) est défini comme suit:

<text area> ::=
<text symbol> ***contains*** <text>

<text symbol> ::=

2.4 Règles de dessin

La taille des symboles graphiques peut être choisie par l'utilisateur. Les frontières des symboles ne doivent pas se chevaucher ni se croiser. Les exceptions à cette règle sont les suivantes:

a) un symbole de message, un symbole de réponse et un symbole de relation d'ordre général peuvent croiser un symbole de message, un symbole de réponse, un symbole de relation d'ordre général, les lignes des symboles de temporisateur, un symbole de ligne de création, un symbole d'axe d'instance, un symbole de méthode, un symbole de suspension, un symbole de corégion, une ligne pointillée dans un symbole de commentaire et un symbole de condition;

b) les lignes des symboles de temporisateur peuvent croiser un symbole de message, un symbole de réponse, un symbole de relation d'ordre général, les lignes des symboles de temporisateur, un symbole de ligne de création, une ligne pointillée d'un symbole de commentaire et un symbole de condition;

c) un symbole de message, un symbole de réponse et un symbole de relation d'ordre général peuvent croiser le symbole d'expression en ligne ou le symbole d'expression en ligne d'exception lorsque les événements d'entrée et de sortie d'un symbole de message, d'un message de réponse et d'un symbole de relation d'ordre général ne sont pas connectés au même axe d'instance;

d) un symbole de ligne de création peut croiser un symbole de message, un symbole de réponse, un symbole de relation d'ordre général, les lignes des symboles de temporisateur,

un symbole d'axe d'instance, un symbole de méthode, un symbole de suspension, un symbole de corégion et une ligne pointillée d'un symbole de commentaire;

e) un symbole de condition peut croiser un symbole d'axe d'instance, un symbole de méthode, un symbole de corégion, un symbole de message, un symbole de réponse, un symbole de relation d'ordre général, les lignes des symboles de temporisateur;

f) un symbole d'expression en ligne peut croiser un symbole d'axe d'instance;

g) un symbole de référence peut croiser un symbole d'axe d'instance et un symbole de méthode;

h) un symbole d'action peut croiser un symbole d'axe d'instance dans sa forme linéaire et chevaucher un symbole d'axe d'instance dans sa forme colonne;

i) un symbole de méthode, un symbole de suspension et un symbole de corégion peuvent chevaucher un symbole d'axe d'instance;

j) un symbole de méthode peut chevaucher un symbole de méthode et un symbole de suspension;

k) deux lignes de diagramme hmsc peuvent se croiser.

Il existe deux formats pour représenter le symbole d'axe d'instance et le symbole de corégion: le format linéaire et le format vertical (en colonne). Il est interdit d'utiliser les deux formats au sein d'une même instance, excepté l'axe linéaire avec la forme verticale des corégions.

Si une condition partagée (voir 4.7) croise une instance non concernée par cette condition, le symbole d'axe d'instance la traverse.

Si une référence partagée (voir 7.3) croise une instance non concernée par cette référence, l'axe de l'instance la traverse.

Si une expression en ligne partagée (voir 7.2) croise une instance non concernée par cette expression en ligne, l'axe de l'instance ne doit pas la traverser.

Si le symbole d'axe d'instance est de forme verticale, les frontières verticales du symbole d'action doivent coïncider avec les lignes de la colonne.

Les lignes représentant les messages peuvent être horizontales ou obliques dirigées vers le bas (conformément à la direction de la flèche), et doivent être des segments de droite connectés en séquence.

Si un événement entrant et un événement sortant se trouvent sur le même point d'un axe d'instance, cela est interprété comme si l'événement entrant était dessiné au-dessus de l'événement sortant. Un symbole de relation d'ordre général ne peut pas être associé à ce point. Deux ou plus de deux événements sortants ne doivent pas être dessinés sur le même point. Deux ou plusieurs événements entrants ne doivent pas être dessinés sur le même point. Les événements suivants sont des événements entrants: entrée de message, détection de message, réception de message, réception de message trouvé, réponse entrante, détection de réponse entrante et expiration de temporisateur. Les événements suivants sont des événements sortants: sortie de message, perte de message, appel, perte d'appel, réponse sortante, perte de réponse sortante, armement de temporisateur, désarmement de temporisateur et création d'instance.

2.5 Mise en page des diagrammes MSC

Les diagrammes MSC peuvent être répartis sur plusieurs pages. La répartition peut être à la fois horizontale et verticale. En variante, la répartition peut être évitée au moyen d'une composition de diagramme MSC ou d'une décomposition d'instance (voir 7.4).

Si un diagramme MSC est réparti verticalement sur plusieurs pages, son en-tête <msc heading> est répété sur chaque page, mais les symboles de fin d'instance ne doivent apparaître que sur une page (sur la "dernière page" pour l'instance en question). Pour chaque instance, la zone d'en-tête d'instance

<instance head area> doit apparaître sur la première page où l'instance en question commence et doit être répétée en forme pointillée sur toutes les pages suivantes où elle continue.

Si des messages, des temporisateurs, des déclarations de création ou des conditions doivent être représentés d'une page à la suivante, le nom du message, du temporisateur, de la création ou de la condition doit apparaître entièrement sur la première page et tout ou partie du texte peut être répété sur la page suivante.

La numérotation des pages doit être indiquée sur les pages d'un diagramme MSC afin de mentionner la séquence correcte des pages. Les pages doivent être numérotées par paire: "v-h" où "v" est le numéro de la page verticale et "h" le numéro de la page horizontale. Les nombres arabes doivent être utilisés pour la numérotation verticale et les lettres majuscules ('A' à 'Z') pour la numérotation horizontale. Si le rang 'A'-'Z' n'est pas suffisant, il peut être étendu avec 'AA' à 'AZ', 'BA' à 'BZ', etc.

3 Document de diagramme des séquences de messages

Le document de diagramme des séquences de messages définit une sorte d'instance et l'ensemble associé de diagramme des séquences de messages qui définit à son tour un ensemble de traces. Les documents MSC peuvent être décrits textuellement ou graphiquement. Comme les documents MSC définissent les instances utilisées dans les diagrammes des séquences de messages, ces documents définissent les structures de décomposition d'instance.

L'en-tête d'un document de diagramme des séquences de messages contient le nom du document éventuellement suivi du mot clé **related to**, puis de l'identificateur (nom du chemin d'accès) du document auquel les diagrammes MSC se réfèrent. Ce document peut être décrit en SDL, UML, TTCN, etc.

Grammaire textuelle concrète

<textual msc document> ::=
<document head>
<textual defining part> <textual utility part>

<document head> ::=
mscdocument <instance kind> [**related to** <sdl reference>]
[<inheritance>] <end>
<using clause>
<containing-clause>
<message decl clause>
<timer decl clause>
[<data definition>]
[<parenthesis declaration>]

<textual defining part> ::=
<defining msc reference>*

<textual utility part> ::=
utilities [<containing-clause>] <defining msc reference>*

<defining msc reference> ::=
reference [<virtuality>] <msc name>

<virtuality> ::=
virtual | **redefined** | **finalized**

<using clause> ::=
{**using** <instance kind> <end> }*

<containing-clause> ::=
{**inst** <instance item>}+

<instance item> ::=
<instance name> [**:** <instance kind>] [<inheritance>]
[<decomposition>]
{ <dynamic decl list> | <end>}

<inheritance> ::=
inherits <instance kind>

<message decl clause> ::=
{**msg** <message decl> <end>}*

<timer decl clause> ::=
{**timer** <timer decl> <end>}*

<sdl reference> ::=
<sdl document identifier>

<identifier> ::=
[<qualifier>] <name>

<qualifier> ::=
<qualifier left> <text> <qualifier right>

Le texte contenu dans un qualificatif ne doit pas contenir de délimiteur '<<' ou '>>'.

La clause de décomposition dans le symbole <instance item> ne s'applique pas aux listes utilisées dans l'en-tête du document mais à la décompositions d'instances dans les diagrammes MSC de haut niveau.

Grammaire graphique concrète

<msc document area> ::=
<frame symbol> ***contains*** {<document head>
is followed by <defining part area> ***is followed by*** <utility part area>}

<defining part area> ::=
{{<defining msc reference area>*} ***set***} ***is followed by*** <separator area>

<utility part area> ::=
[<containing area> ***is followed by***] {<defining msc reference area>*} ***set***

<containing area> ::=
<containing-clause>

<defining msc reference area> ::=
<msc reference symbol>
contains [<virtuality>] <msc name>

Exigences statiques

Les instances contenues dans les diagrammes MSC citées en référence à partir de la partie définition et de la partie utilitaire ne doivent être choisies que dans la liste d'instances de la clause englobante <containing-clause> figurant dans l'en-tête du document <document head> et au début de la partie utilitaire.

La clause englobante facultative de la partie utilitaire comprendra les instances qui sont contenues dans les diagrammes MSC de la partie utilitaire mais non dans ceux de la partie définition.

Les instances contenues dans la clause englobante <containing_clause> peuvent être décomposées (**decomposed**) en quelques-uns des diagrammes MSC. Soit l'instance *x* dans le MSC *m*, qui est décomposée en (**decomposed as**) *xm*. La partie définition du document MSC pour *x* doit contenir le MSC *xm*. Toutes les instances qui sont décomposées en diagrammes MSC doivent être déclarées comme décomposées dans les clauses englobantes.

Lorsque la clause englobante contient une clause d'héritage (**inherits**) et que la sorte d'instance héritière est elle-même définie par un document MSC, il doit toujours y avoir une clause d'héritage correspondante dans l'en-tête de ce document.

La clause de virtualité de la référence définissant un diagramme MSC doit toujours correspondre à l'en-tête <msc heading> de la définition de ce diagramme MSC.

Un diagramme MSC redéfini (**redefined**) ou finalisé (**finalized**) doit toujours avoir dans la sorte d'instance héritée un diagramme MSC correspondant qui n'est pas finalisé (**finalized**) dans cette sorte d'instance héritée.

Sémantique

Un document MSC est un ensemble de diagrammes des séquences de messages, faisant éventuellement référence à un document SDL correspondant.

Le document MSC définit une instance ou une sorte d'instance. Une instance possède un ensemble de diagrammes des séquences de messages qui lui est associé. Les diagrammes MSC sont des scénarios d'interaction entre les instances contenues dans l'instance définie. La partie définissante (diagrammes MSC définisseurs) définit l'ensemble des traces possibles d'un diagramme MSC tandis que la partie utilitaire ne contient que les structures (diagrammes MSC utilitaires) réutilisées par la partie définissante. Il est néanmoins permis que les diagrammes MSC de la partie utilitaire fassent référence aux diagrammes MSC de la partie définissante.

La signification de la clause d'héritage facultative (**inherits**) est équivalente aux opérations suivantes:

1) inclusion, dans les clauses englobantes de la sorte héritière, de toutes les instances de la sorte héritée;
2) inclusion, dans la partie définissante de la sorte héritière, de tous les diagrammes MSC de la sorte héritée;
3) inclusion, dans la partie utilitaire de la sorte héritière, de tous les diagrammes MSC utilitaires de la sorte héritée;
4) remplacement de chaque diagramme MSC virtuel (**virtual**) ou redéfini (**redefined**) de la sorte héritée par le diagramme redéfini (**redefined**) ou finalisé (**finalized**) de la sorte héritière.

La clause facultative d'utilisation (**using**) définit l'utilisation des documents MSC en tant que recueils de diagrammes MSC. Le rôle de la clause d'utilisation est d'inclure la partie définissante des diagrammes MSC du recueil dans la partie utilitaire du document MSC englobant.

Les noms d'instance peuvent être qualifiés par les noms des instances englobantes.

Le mot clé **decomposed** dans la clause englobante indique que l'instance est décomposée dans au moins un diagramme MSC.

4 Diagramme MSC de base

4.1 Diagramme des séquences de messages

Un diagramme des séquences de messages, dont l'abréviation est MSC, décrit le flux des messages entre instances. Un diagramme des séquences de messages décrit un comportement partiel d'un système. Bien que l'expression *diagramme des séquences de messages (message sequence chart)* ait pour origine, de manière évidente, sa représentation graphique, ce terme est utilisé pour désigner à la fois les représentations textuelles et graphiques.

Un diagramme MSC est décrit soit par des instances et des événements, soit par une expression se rapportant à des références MSC ne nommant pas explicitement les instances (voir 7.5).

Grammaire concrète textuelle

<message sequence chart> ::=
[<virtuality>] **msc** <msc head> { <msc> | <hmsc> } **endmsc** <end>

<msc> ::=
<msc body>

<msc head> ::=
<msc name>[<msc parameter decl>] [<time offset>]<end>
[<msc inst interface>] <msc gate interface>

<msc parameter decl> ::=
([<data parameter decl>][<instance parameter decl>]
[<message parameter decl>][<timer parameter decl>])

<instance parameter decl> ::=
inst <instance parm decl list> <end>

<instance parm decl list> ::=
<instance parameter name> [: <instance kind>] [,<instance parm decl list>]

<instance parameter name> ::=
<instance name>

<message parameter decl> ::=
msg <message parm decl list>

<message parm decl list> ::=
<message decl list>

<timer parameter decl> ::=
timer <timer parm decl list>

<timer parm decl list> ::=
<timer decl list>

<msc inst interface> ::=
<containing-clause>

<instance kind> ::=
[<kind denominator>] <identifier>

<kind denominator> ::=
system | **block** | **process** | **service** | <name>

<msc gate interface> ::=
<msc gate def>*

<msc gate def> ::=
gate { <msg gate> | <method call gate> | <reply gate>|
<create gate> | <order gate> } <end>

<msg gate> ::=
<def in gate> | <def out gate>

<method call gate> ::=
<def out call gate> | <def in call gate>

<reply gate> ::=
<def out reply gate> | <def in reply gate>

<create gate> ::=
<def create in gate> | <def create out gate>

<order gate> ::=
<def order in gate> | <def order out gate>

<msc body> ::=
<msc statement>*

<msc statement> ::=
<text definition> | <event definition>

<event definition> ::=
<instance name> **:** <instance event list>
| <instance name list> **:** <multi instance event list>

<instance event list> ::=
<instance event> +

<instance event> ::=
<orderable event> | <non-orderable event>

<orderable event> ::=
[**label** <event name> <end>]
{ <message event> | <incomplete message event> |
<method call event> | <incomplete method call event> | <create> |
<timer statement> | <action> }
[**before** <order dest list>] [**after** <order dest list>] <end>
[**time** <time dest list> <end>]

L'étiquette (**label**) facultative de nom d'événement <event name> ; est utilisée lorsque l'événement suit une relation d'ordre général.

<order dest list> ::=
<order dest> [, <order dest list>]

<time dest list> ::=
<time interval> [<time dest>] [, <time dest list>]

<time dest> ::=

<event name> | [**begin** | **end**] <reference identification>

La relation temporelle des événements pouvant suivre une relation d'ordre sert à exprimer une temporisation. Dans le cas d'une temporisation relative, le temps est indiqué par rapport à un événement précédent. En temporisation absolue, c'est le temps absolu qui est attribué.

Le nom de porte <gate name> contenu dans le symbole <order dest> désigne une porte de type <def order out gate>, <actual order in gate>, <inline order out gate> ou <def order out gate>. Le nom d'événement <event name> contenu dans le symbole <order dest> désigne un événement pouvant suivre une relation d'ordre <orderable event>.

<non-orderable event> ::=
<start method> | <end method> | <start suspension> | <end suspension> |
<start coregion> | <end coregion> | <shared condition> |
<shared msc reference> | <shared inline expr> |
<instance head statement> | <instance end statement> | <stop>

<instance name list> ::=
<instance name> { **,** <instance name> }***** | **all**

<multi instance event list> ::=
<multi instance event> $^{+}$

<multi instance event> ::=
<condition> | <msc reference> | <inline expr>

Exigences statiques

A chaque déclaration de début d'instance <instance head statement> doit correspondre une déclaration de fin d'instance <instance end statement> ou un événement d'arrêt <stop>. Pour chaque instance, il ne doit pas y avoir d'événements avant la définition de sa déclaration de début d'instance <instance head statement>. Pour chaque instance, il ne doit pas y avoir d'événements après sa déclaration de fin d'instance <instance end statement>. Pour chaque instance, il ne doit pas y avoir plus d'une seule déclaration de début d'instance <instance head statement> et pas plus d'une seule déclaration de fin d'instance <instance end statement>.

Les instances spécifiées dans les diagrammes <msc> ou <hmsc> doivent constituer un sous-ensemble des instances spécifiées dans la clause englobante <textual msc document>.

Si elle est présente, la clause englobante <containing-clause> doit contenir les mêmes instances que spécifié dans le diagramme <msc> ou <hmsc>.

Dans le cas des diagrammes MSC de haut niveau (<hmsc>), la décomposition des instances doit être décrite dans la clause de décomposition de la clause englobante <containing-clause> contenue dans l'en-tête de ces diagrammes MSC.

Grammaire graphique concrète

<msc diagram> ::=
<simple msc diagram> | <hmsc diagram>

<simple msc diagram> ::=
<msc symbol>
contains <msc heading> <msc body area>

<hmsc diagram> ::=
<msc symbol>
contains {<msc heading> [<containing-clause>]} <mscexpr area>

<msc symbol> ::=
<frame symbol> ***is attached to*** { <def gate area>* } ***set***

<frame symbol> ::=

<msc heading> ::=
msc <msc name>[<msc parameter decl>] [<time offset>]

<msc body area> ::=
{ <instance layer> <text layer> <gate def layer>
<event layer> <connector layer> } ***set***

<instance layer> ::=
{ <instance area>* } ***set***

<text layer> ::=
{ <text area>* } ***set***

<gate def layer> ::=
{ <def gate area>* } ***set***

<event layer> ::=
<event area> | <event area> ***above*** <event layer>

<connector layer> ::=
{ <message area>* | <incomplete message area>* |
<method call area>* | <incomplete method call area>*
<reply area>* | <incomplete reply area>*} ***set***

<event area> ::=
<instance event area>
| <shared event area>
| <create area>

<instance event area> ::=
{<message event area>
| <method call event area>
| <reply event area>
| <timer area>
| <concurrent area>
| <method area>
| <suspension area>
| <action area>}
[***is followed by*** <general name area>]

<shared event area> ::=
| <condition area>
| <msc reference area>
| <inline expression area>

Exigences statiques

Tous les messages et tous les temporisateurs comportant des paramètres doivent être déclarés dans le document MSC englobant. Les instances utilisées dans les diagrammes MSC doivent également être définies dans le document MSC englobant.

Les sortes d'instance des paramètres d'instance doivent être définies dans le document MSC englobant. Les sortes d'instance sont définies par le fait que les instances de la sorte spécifiée sont contenues dans le document MSC. Lorsque la sorte d'instance n'est pas présente dans la déclaration des paramètres, cette sorte est équivalente à une sorte quelconque.

Sémantique

Un diagramme MSC décrit la communication entre des composantes d'un système, et entre ces composantes et le reste du monde, appelé environnement. Pour chaque composante de système décrit par un diagramme MSC, il existe un axe d'instance. La communication entre les composantes d'un système est assurée au moyen de messages. L'envoi et la consommation de messages sont deux événements asynchrones. On suppose que l'environnement d'un diagramme MSC est capable de recevoir des messages du diagramme des séquences de messages et d'en envoyer à celui-ci; il n'y a pas de relation d'ordre au niveau des événements liés aux messages dans l'environnement.

On suppose que chaque diagramme des séquences de messages comporte un axe de temps global. Le long de chaque axe d'instance, le temps se déroule du haut vers le bas; cependant, il n'existe pas d'échelle des temps particulière. S'il n'existe aucune corégion ou expression en ligne (voir 7.1 et 7.2), les événements sont censés être totalement ordonnés dans le temps, le long de chaque axe d'instance. Les événements des différentes instances sont uniquement ordonnés à travers les messages – un message doit d'abord être envoyé avant d'être consommé – ou par ce qu'on appelle le mécanisme de mise en relation d'ordre général. Avec ce mécanisme de mise en relation d'ordre général, "les événements ordonnés" sur différentes instances (même dans différents MSC) peuvent être ordonnés de façon explicite. Aucun autre ordre n'est imposé. Par conséquent, un diagramme des séquences de messages impose un ordre partiel sur l'ensemble des événements qu'il contient. Une relation binaire transitive, antisymétrique et non réflexive est appelée ordre partiel.

Par exemple, pour les messages en entrée du diagramme des séquences de messages de la Figure 6a) (étiquetés par in(m*i*)) et en sortie (étiquetés par out(m*i*)), nous avons la relation d'ordre suivante: out(m2) < in(m2), out(m3) < in(m3), out(m4) < in(m4), in(m1) < out(m2) < out(m3) < in(m4), in(m2) < out(m4) et sa fermeture transitive.

L'ordre partiel peut être décrit en forme minimale (sans représentation explicite de la fermeture transitive) au moyen de son graphe de connexité [voir Figure 6b)].

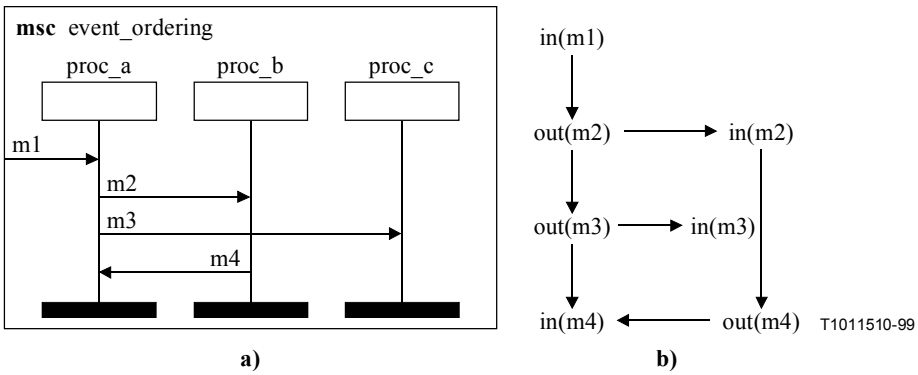


Figure 6/Z.120 – Diagramme des séquences de messages et graphe de connexité correspondant

Une sémantique formelle des diagrammes MSC, fondée sur un processus algébrique, est présentée dans l'Annexe B. La sémantique d'un diagramme MSC peut être apparentée à la sémantique du SDL par la notion de graphe d'accessibilité. Chaque mise en séquence d'un diagramme MSC décrit une trace d'un nœud à un autre nœud (ou à un ensemble de nœuds) du graphe d'accessibilité décrivant le comportement d'une spécification d'un système SDL. Un graphe d'accessibilité est constitué de nœuds et d'arcs. Les nœuds représentent les états globaux du système. L'état global d'un système est déterminé par les valeurs des variables et l'état d'exécution de chaque processus et les contenus des files de messages. Les arcs correspondent aux événements exécutés par le système, par exemple l'envoi et la consommation d'un message ou l'exécution d'une tâche. La mise en séquence d'un diagramme MSC représente un ordre total d'événements compatible avec l'ordre partiel défini par le MSC.

Dans la représentation textuelle, la définition <event definition> est fournie soit par un nom <instance name> suivi de la liste rattachée <event list>, soit par la liste <instance name list> suivie des événements rattachés <multi instance event> pour lesquels les deux points servent de séparateur. Le non terminal <instance event> représente soit un événement attaché à une instance simple, par exemple une <action> ou un objet partagé, comme une condition partagée <shared condition> où le mot clé **shared** suivi de la liste des instances ou du mot clé **all** est utilisé pour définir l'ensemble des instances partageant la condition. Un objet partagé peut être représenté également par des événements d'instances multiples <multi instance event>. Les notations différentes sont introduites pour faciliter une description *orientée instance (instance oriented)*, d'une part, et une description *orientée événement (event oriented)* d'autre part. Ces deux notations peuvent être utilisées ensemble.

La description *orientée instance* énumère les événements associés à une instance.

Dans la représentation textuelle de la description *orientée événement*, les événements peuvent être énumérés sous forme d'une trace d'exécution possible et ne sont pas ordonnés par rapport aux instances.

La déclaration facultative <msc interface> décrivant l'interface d'un diagramme MSC avec son environnement consiste en une interface de type <msc inst interface> et une interface de type <msc gate interface>. L'interface de type <msc inst interface> fournit une déclaration des instances, c'est-à-dire un nom d'instance <instance name> et éventuellement une sorte d'instance <instance kind>. Puisqu'un diagramme MSC ne représente que l'exécution d'une partie d'un système, l'interface <msc inst interface> décrit les points de connexion des instances à l'environnement. L'interface <msc gate interface> fournit une définition des portes de message et des portes d'ordre contenues dans le diagramme MSC. Les "portes" associées aux messages définissent les points de connexion des messages à l'environnement. Eventuellement, des noms de porte peuvent être associés aux portes.

4.2 Instance

Un diagramme des séquences de messages est composé d'instances d'entités qui interagissent. L'instance d'une entité est un objet qui a les propriétés de cette entité. Si on se réfère au SDL, une entité peut être un processus SDL, un bloc, un processus ou un service. Au niveau de l'en-tête de l'instance, le nom de l'entité, par exemple le nom du processus, peut être ajouté au nom de l'instance.

Au niveau du corps de l'instance, l'ordre des événements est spécifié.

Grammaire textuelle concrète

<instance head statement> ::=
instance [<instance kind>] [<decomposition>] <end>

<instance end statement> ::=
endinstance <end>

Grammaire graphique concrète

<instance area> ::=
<instance fragment area> [***is followed by*** <instance area>]

<instance fragment area> ::=
<instance head area> ***is followed by*** <instance body area>

<instance head area> ::=
<instance head symbol>
is associated with <instance heading>
[***is attached to*** <createline symbol>
[***is attached to***
{{<int symbol> | <abs time symbol>}*}***set***]]

<instance heading> ::=
<instance name> [[**:**] <instance kind>] [<decomposition>]

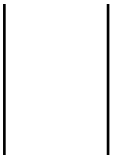
<instance head symbol> ::=

<instance body area> ::=
<instance axis symbol>
is followed by { <instance end symbol> | <stop symbol> }

<instance axis symbol> ::=
{ <instance axis symbol1> | <instance axis symbol2> }
is attached to { <event area>* } ***set***

<instance axis symbol1> ::=

<instance axis symbol2> ::=



<instance end symbol> ::=

L'en-tête d'instance <instance heading> peut être placé au-dessus ou à l'intérieur du symbole correspondant <instance head symbol>, ou bien être fractionné de telle sorte que le nom d'instance <instance name> soit placé à l'intérieur du symbole d'en-tête d'instance <instance head symbol>

tandis que la sorte d'instance <instance kind> et la décomposition <decomposition> sont placées au-dessus. Si l'en-tête d'instance <instance heading> est divisé, le symbole des deux points est facultatif.

Tous les fragments d'instance ayant le même nom doivent être placés directement les uns au-dessous des autres lorsqu'ils apparaissent sur la même page.

Pour indiquer qu'une référence de diagramme MSC ne contient qu'une création ou qu'une terminaison d'instance, les extrémités du fragment doivent être associées au symbole de référence MSC.

Sémantique

Les instances sont définies dans le corps du diagramme des séquences de messages. Le symbole de fin d'instance correspond à la fin de la description de l'instance au sein du diagramme MSC. Il ne correspond pas à la terminaison de l'instance (voir 4.11). De même, le symbole de début d'instance correspond au commencement de la description de l'instance au sein du diagramme MSC. Il ne correspond pas à la création de l'instance (voir 4.10). Tous les fragments d'instance ayant le même nom constituent la même instance.

Dans le cadre du SDL, une instance peut se rapporter à un processus (mot clé **process**), à un service (mot clé **service**) ou à un bloc (mot clé **block**) ou à un système (mot clé **system**). En dehors de ce cadre, une instance peut se rapporter à toute sorte d'entité. La définition d'instance fournit, en termes d'événements, une description pour les entrées et sorties de messages, les appels et réponses de méthode, les actions, les conditions locales et partagées, les événements de temporisation, la création d'instances et la terminaison d'instances. Mise à part l'utilisation de corégions (voir 7.1) et des expressions en ligne (voir 7.2), l'on part du principe qu'il existe une relation d'ordre total des événements le long de chaque axe d'instance. Au sein des corégions, on ne garantit pas un ordre temporel des événements si aucune autre construction de synchronisation sous forme de relations d'ordre général n'est spécifiée.

4.3 Message

Un message contenu dans un diagramme MSC est une relation entre une sortie et une entrée. La sortie peut provenir soit de l'environnement (à travers une porte) soit d'une instance ou peut être trouvée (**found**); une entrée est destinée soit à l'environnement (à travers une porte) soit à une instance ou peut être perdue (**lost**).

Un message échangé entre deux instances peut être décomposé en deux événements: le message en entrée et le message en sortie; par exemple, le deuxième message de la Figure 6 a) peut être décomposé en out(m2) (sortie) et in(m2) (entrée). Des paramètres peuvent être insérés dans un message (voir 5.8).

La correspondance entre les messages en sortie et les messages en entrée doit être biunivoque. Dans la représentation textuelle, le mappage entre les messages en entrée et les messages en sortie se fait, en principe, par identification du nom de message et de l'adresse spécifiée. Dans la représentation graphique, un message est représenté par une flèche.

La perte d'un message, c'est-à-dire si un message est envoyé mais n'est pas consommé, est indiquée par le mot clé **lost** dans la représentation textuelle et par un "trou noir" dans la représentation graphique.

Inversement, un message spontanément trouvé, c'est-à-dire dont l'origine est inconnue, est défini par le mot clé **found** dans la représentation textuelle et par un "trou blanc" dans la représentation graphique.

Dans la représentation textuelle, les mots clés **before** et **after** permettent de définir un ordre temporel des événements de messagerie concernant différentes instances. Dans la représentation graphique, ces concepts de relation d'ordre généralisé sont définis à l'aide des constructions de synchronisation sous forme de lignes de connexion.

L'intervalle de temps relatif à un message temporisé indique le délai entre la sortie et l'entrée de ce message. De même, le message en entrée d'un message d'entrée incomplet, ainsi que le message en sortie d'un message de sortie incomplet, peut être soumis à des contraintes de temps. Par ailleurs, l'événement de message en sortie et de message en entrée peut être soumis à des contraintes concernant d'autres événements.

Grammaire textuelle concrète

<message event> ::=
<message output> | <message input>

<message output> ::=
out <msg identification> **to** <input address>

<message input> ::=
in <msg identification> **from** <output address>

<incomplete message event> ::=
<incomplete message output> | <incomplete message input>

<incomplete message output> ::=
out <msg identification> **to lost** [<input address>]

<incomplete message input> ::=
in <msg identification> **from found** [<output address>]

<msg identification> ::=
<message name> [**,** <message instance name>] [(<parameter list>)]

Le nom d'instance de message <message instance name> n'est nécessaire qu'en notation textuelle.

<output address> ::=
<instance name> | { **env** | <reference identification>} [**via** <gate name>]

<reference identification> ::=
reference <msc reference identification>
| **inline** <inline expr identification>

Le nom <gate name> fait référence à une porte définie <def in gate>. Si seul le mot clé **env** est utilisé alors l'adresse <output address> désigne une porte définie <def in gate> ayant un nom implicite (construit à l'aide de l'identification du message <msg identification> correspondante et du sens **in**).

<input address> ::=
<instance name> | { **env** | <reference identification>} [**via** <gate name>]

Le nom <gate name> fait référence à une porte définie <def out gate>. Si seul le mot clé **env** est utilisé alors l'adresse <input address> désigne une porte définie <def out gate> ayant un nom implicite (construit à l'aide de l'identification du message <msg identification> correspondante et du sens **out**).

Exigences statiques

Pour les messages échangés entre les instances, les règles suivantes doivent être appliquées: pour chaque sortie de message <message output> une entrée de message <message input> correspondante doit être spécifiée et vice versa. Si le nom du message <message name> et l'adresse <address> ne suffisent pas pour un mappage biunivoque, le nom d'instance de message <message instance name> doit être précisé.

Il n'est pas autorisé que la sortie de message <message output> soit en relation de dépendance causale avec son entrée de message <message input> au moyen d'autres messages ou de constructions de mise en relation d'ordre général. Une telle dépendance causale existe si le graphe de connexité (voir Figure 6) contient des boucles. Si une liste de paramètres <parameter list> est spécifiée pour une entrée de message <message input>, cette liste doit également être spécifiée pour la sortie <message output> correspondante et vice versa. La liste de paramètres <parameter list>

pour la sortie <message output> n'est composée que d'expressions <expression>. La liste de paramètres pour l'entrée <message input> n'est composée que de structures <pattern>.

Grammaire graphique concrète

<message event area> ::=
{ <message out area> | <message in area> }
{***is followed by*** <general order area> }*
{***is attached to*** <general order area> }*

Les messages peuvent être mis en relation d'ordre général à l'aide de plusieurs relations d'ordre général différentes. Les événements de messagerie apparaissent de part et d'autre de la relation d'ordre.

<message out area> ::=
<message out symbol>
is attached to <instance axis symbol>
is attached to <message symbol>
[***is attached to***
{<int symbol> | <abs time symbol> }*]

<message out symbol> ::=
<void symbol>

<void symbol> ::= .

Le symbole <void symbol> est un point géométrique sans dimensions.

Le symbole <message out symbol> n'est en fait qu'un point sur l'axe de l'instance. La fin du symbole du message sans pointe de flèche est aussi sur ce point de l'axe.

<message in area> ::=
<message in symbol>
is attached to <instance axis symbol>
is attached to <message symbol>
[***is attached to***
{<int symbol> | <abs time symbol> }*]

<message in symbol> ::=
<void symbol>

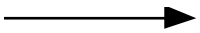
Le symbole <message in symbol> n'est en fait qu'un point sur l'axe de l'instance. La fin du symbole du message, qui est la pointe de la flèche, est aussi dirigée vers ce point de l'axe de l'instance.

<message area> ::=
<message symbol>
is associated with <msg identification> [**time** <time interval>]
is attached to {<message start area> | <message end area>}

<message start area> ::=
<message out area> | <actual out gate area>
| <def in gate area> | <inline gate area>

<message end area> ::=
<message in area> | <actual in gate area>
| <def out gate area> | <inline gate area>

<message symbol> ::=



L'image miroir du symbole <message symbol> est autorisée. La pointe de la flèche devra être sur l'axe de l'instance.

Dans la représentation graphique, le nom d'instance de message n'est pas nécessaire pour une description biunivoque.

<incomplete message area> ::=
{ <lost message area> | <found message area> }
{ ***is followed by*** <general order area> }*
{ ***is attached to*** <general order area> }*

<lost message area> ::=
<lost message symbol> ***is associated with*** <msg identification>
[***is associated with*** { <instance name> | <gate name> }]
is attached to <message start area>

<lost message symbol> ::=

Le symbole représentant la perte de message <lost message symbol> décrit l'événement du côté sortie: le trait plein part de la zone <message start area> correspondant au début de l'événement. La cible d'un message est facultative et peut être désignée par un identificateur associé au symbole. Il y a lieu que l'identification de la cible soit écrite près du cercle noir, et l'identification du message près de la flèche.

L'image miroir du symbole peut aussi être utilisée.

<found message area> ::=
<found message symbol> ***is associated with*** <msg identification>
[***is associated with*** { <instance name> | <gate name> }]
is attached to <message end area>

<found message symbol> ::=

Le symbole représentant le message trouvé <found message symbol> décrit l'événement du côté entrée (la tête de la flèche) qui doit toujours être sur une zone de fin de message <message end area>. L'instance ou la porte supposée être à l'origine du message est indiquée par l'identification facultative associée au cercle du symbole. Il y a lieu que l'identification du message soit écrite près de la flèche.

L'image miroir du symbole peut aussi être utilisée.

Exigences statiques

Une liste de paramètres <parameter list> d'une zone de message <message area> dont les deux extrémités sont associées à des zones d'événement d'instance <instance event area> se compose d'associations <bindings>. Une liste de paramètres <parameter list> d'une zone de message <message area> dont une des extrémités est associée à une zone d'événement de sortie <output event area> et dont l'autre extrémité est associée à une porte ou est perdue, se compose d'expressions <expression>. Une liste de paramètres <parameter list> d'une zone de message <message area> dont une des extrémités est associée à une zone d'événement d'entrée <input event area> et l'autre à une porte ou est trouvée, se compose de structures <pattern>.

Sémantique

Pour un diagramme MSC, le message en sortie indique un envoi de message et le message en entrée indique une consommation de message. Aucune construction spéciale n'est fournie pour la réception de message (entrée dans la file).

Un message incomplet est un message qui est soit une sortie (dont l'entrée est perdue/inconnue) soit une entrée (dont la sortie est trouvée/inconnue).

Si des événements de messagerie coïncident avec d'autres événements, voir les règles de dessin du 2.4.

4.4 Fluences

Le diagramme MSC peut décrire des fluences, non seulement au moyen de messages asynchrones mais aussi au moyen d'appels et de réponses.

Une méthode est une unité de comportement nommée à l'intérieur d'une instance. Une méthode peut être invoquée à distance et les résultats des calculs effectués selon la méthode peuvent être retournés au demandeur au moyen d'une réponse. Celle-ci aura le même nom que l'appel correspondant.

Un appel de méthode peut être asynchrone ou synchronisateur. Un appel asynchrone implique que le demandeur peut continuer sans attendre la réponse à son appel. D'autre part, un appel synchronisateur implique que le demandeur entrera dans une région de suspension, dans laquelle aucun événement ne se produira avant le retour de la réponse à l'appel. Comme les appels de méthode peuvent se produire à l'intérieur d'instances décomposées, les méthodes et les régions de suspension peuvent être omises. Si le symbole de méthode <method symbol> et le symbole de suspension <suspension symbol> sont utilisés, le symbole de méthode <method symbol> indique qu'une instance est active. Le symbole de suspension <suspension symbol> indique qu'une instance est suspendue, normalement pour attendre la résolution d'une sorte de condition de blocage (par exemple en attendant la réponse à un appel synchrone) ou pour obtenir l'accès à une ressource partagée (comme une unité centrale) afin de continuer une tâche en cours. Le symbole normal d'axe d'instance (<instance axis symbol1> ou <instance axis symbol2>) signifie, dans ce contexte, que l'instance est inactive et en attente de l'activation d'un événement entrant (<message input> ou <call in>) et du commencement d'une des tâches qu'il est capable d'effectuer. Le symbole qui décrit le niveau d'activation d'une instance n'implique pas d'effets dynamiques concernant la sémantique formelle, qui ne prend en considération que la relation d'ordre des événements, mais ne fait que poser des exigences concernant le point où les messages peuvent faire entrer et sortir leurs événements.

Les appels de méthode et les réponses de méthode peuvent également être incomplets, de sorte que l'appel ou la réponse soit perdu.

Comme dans le cas des contraintes temporelles exercées sur des messages, les appels de méthode et les réponses de méthodes peuvent obéir à des contraintes temporelles.

Grammaire textuelle concrète

<method call event> ::=
<call out> | <call in> |<reply out> | <reply in>

<call out> ::=
call <msg identification> **to** <input address>

<call in> ::=
receive <msg identification> **from** <output address>

<reply out> ::=
replyout <msg identification> **to** <input address>

<reply in> ::=
replyin <msg identification> **from** <output address>

<incomplete method call event> ::=
<incomplete call out> | <incomplete call in> |
<incomplete reply out> | <incomplete reply in>

<incomplete call out> ::=
call <msg identification> **to lost** [<input address>]

<incomplete call in> ::=
receive <msg identification> **from found** [<output address>]

<incomplete reply out> ::=
replyout <msg identification> **to lost** [<input address>]

<incomplete reply in> ::=
replyin <msg identification> **from found** [<output address>]

<start method> ::=
method <end>

<end method> ::=
endmethod <end>

<start suspension> ::=
suspension <end>

<end suspension>::=
endsuspension <end>

Grammaire graphique concrète

<method call area> ::=
<message symbol>
is associated with <method identification>[**time** <time interval>]
is attached to { <method call start area> <method call end area> }
is attached to {<instance axis symbol>}
[***is attached to*** {<method area>}]

<method identification> ::=
call <msg identification>

<method call start area> ::=
<call out area> | <actual out gate area> |
<def in gate area> | <inline gate area>
is attached to <instance axis symbol>
is attached to <message symbol>
[***is attached to*** <suspension symbol>]
[***is attached to***
{<int symbol> | <abs time symbol> }*]

<method call end area> ::=
<call in area> | <actual in gate area> |
<def out gate area> | <inline gate area>
is attached to <instance axis symbol>
is attached to <message symbol>
[***is attached to*** <method symbol>]
[***is attached to***
{<int symbol> | <abs time symbol> }*]

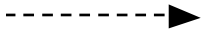
<reply area> ::=
<reply symbol>
is associated with <msg identification>[**time** <time interval>]
is attached to { <reply start area> <reply end area>}

<reply start area> ::=
<reply out area> | <actual out gate area> |
<def in gate area> | <inline gate area>
is attached to <instance axis symbol>
is attached to <reply symbol>
[***is attached to*** <method symbol>]
[***is attached to***
{<int symbol> | <abs time symbol> }*]

<reply end area> ::=
<reply in area> | <actual in gate area> |
<def out gate area> | <inline gate area>
is attached to <instance axis symbol>
is attached to <reply symbol>
[***is attached to*** <suspension symbol>]

[***is attached to***
{<int symbol> | <abs time symbol> }*]

<reply symbol> ::=



<incomplete method call area> ::=
{ <lost method call area> | <found method call area> }
{ ***is followed by*** <general order area> }*
{ ***is attached to*** <general order area> }*

<lost method call area> ::=
<lost message symbol> ***is associated with*** <method identification>
[***is associated with*** { <instance name> | <gate name> }]
is attached to <method call start area>

<found method call area> ::=
<found message symbol> ***is associated with*** <method identification>
[***is associated with*** { <instance name> | <gate name> }]
is attached to <method call end area>

<incomplete reply area> ::=
{ <lost reply area> | <found reply area> }
{ ***is followed by*** <general order area> }*
{ ***is attached to*** <general order area> }*

<lost reply area> ::=
<lost reply symbol> ***is associated with*** <msg identification>
[***is associated with*** { <instance name> | <gate name> }]
is attached to <reply start area>

<lost reply symbol> ::=

<found reply area> ::=
<found reply symbol> ***is associated with*** <msg identification>
[***is associated with*** { <instance name> | <gate name> }]
is attached to <reply end area>

<found reply symbol> ::=

<method call event area> ::=
{<call out area> | <call in area>}
{***is followed by*** <general order area> }*
{***is attached to*** <general order area> }*

<call out area> ::=
<call out symbol>
is attached to <instance axis symbol>
is attached to <message symbol>

<call out symbol> ::=
<void symbol>

<call in area> ::=
<call in symbol>
is attached to <instance axis symbol>
is attached to <message symbol>

<call in symbol> ::=
<void symbol>

<reply event area> ::=
{<reply out area> | <reply in area>}
{***is followed by*** <general order area> }*
{***is attached to*** <general order area> }*

<reply out area> ::=
<reply out symbol>
is attached to <instance axis symbol>
is attached to <message symbol>

<reply out symbol> ::=
<void symbol>

<reply in area> ::=
<reply in symbol>
is attached to <instance axis symbol>
is attached to <message symbol>

<reply in symbol> ::=
<void symbol>

<method area> ::=
<method symbol>
is attached to <instance axis symbol>
[***contains*** <method event layer>]

<method event layer>::=
<method event area> | <method event area> ***above*** <method event layer>

<method event area>::=
<event area>

<method symbol> ::=

<suspension area> ::=
<suspension symbol>
is attached to <instance axis symbol>
[***contains*** <suspension event layer>]

<suspension event layer>::=
<method invokation area>
| <method invokation area> ***above*** <suspension event layer>

<method invokation area> ::=
<method start area>
is followed by <method area>
is followed by <method end area>

<method start area> ::=
<call in area> | <found method call area>

<method end area> ::=
<reply out area> | <lost reply area>

<suspension symbol> ::=

Exigences statiques

Les règles grammaticales suivantes se rapportent aux événements d'une instance spécifique:

<start method> <instance event list> <end method>

<start suspension> <instance event list> <end suspension>

<call out> <instance event list> <reply in>, où <reply in> correspond à <call out>.

<call in> <instance event list> <reply out>, où <reply out> correspond à <call in>.

Une méthode de démarrage <start method> ne peut que venir directement après un événement d'appel entrant <call in>, d'appel entrant incomplet <incomplete call in>, d'événement de message entrant <message input>, d'événement de message entrant incomplet <incomplete message input>, d'événement de fin de suspension <end suspension> ou de création <create> si l'instance créée relève de la méthode de démarrage <start method>. Un symbole de méthode peut se terminer à tout moment, mais une fin de méthode <end method> doit toujours suivre directement un événement de réponse centrifuge <reply out> ou de réponse centrifuge incomplète <incomplete reply out>.

Les (éventuels) événements d'appel et de réponse dynamiques doivent toujours se produire en paires. Une instance spécifique qui émet un événement de type <call out> ne doit pas émettre d'autre événement d'appel centrifuge avant d'avoir reçu un événement correspondant <reply in> ou un événement <call in> qui est en relation de dépendance causale avec son événement <call out> (c'est-à-dire que les appels et les réponses peuvent être imbriqués). Si une région de suspension commence immédiatement après un appel sortant, cette région se termine avec la réponse correspondante.

Tous les événements de type <instance event> sont autorisés dans le cadre des symboles de méthode. Des méthodes peuvent être superposées à des méthodes et à des régions de suspension. Une région "superposée" est telle qu'elle puisse être dessinée avec un petit décalage horizontal par rapport à la région de méthode ou à la région de suspension dans laquelle elle est contenue.

Les régions de suspension ne sont pas autorisées à contenir d'événements quelconques. Une région de méthode peut être superposée à une région de suspension lorsque celle-ci commence par un appel de type <call out>; par conséquent, elle est appelée par récurrence (directement ou indirectement) au moyen d'un appel de type <call in>.

Une région de suspension peut se terminer à tout moment si elle ne suit pas directement un événement de type <call out>, auquel cas elle se termine par l'événement <reply in> correspondant[1].

[1] Des exigences statiques additionnelles peuvent être *ajoutées* aux exigences statiques pour l'adaptation à un modèle d'exécution spécifique ou pour l'expression d'une conséquence lors du dessin du modèle. Par exemple, si la fluence d'un langage purement procédural est décrite (c'est-à-dire sans parallélisme), un événement de type <call out> doit toujours être suivi d'un événement de type <start suspension>; ou encore, si chaque instance est toujours capable de traiter des événements (parce qu'elle possède sa propre unité CPU), une région de suspension ne peut commencer qu'après un événement <call out>.

4.5 Environnement et portes

Les portes représentent l'interface entre le diagramme MSC et son environnement. Un message ou une relation d'ordre attaché au cadre du diagramme MSC constitue une porte. Voir Figure 7 pour des exemples de portes.

Une porte de message a toujours un nom. Celui-ci peut être défini explicitement par un nom associé à la porte attachée au cadre. Sinon, ce nom est donné implicitement par le sens du message traversant la porte et par le nom du message; par exemple "in_X" représente une porte recevant un message X de son environnement.

Les portes de message sont utilisées lorsque les références à un diagramme MSC étendent le contexte à un autre diagramme MSC. Les portes réelles de la référence au MSC sont alors connectées à d'autres "portes de message" ou à d'autres instances. De même que pour leurs définitions, les portes réelles peuvent avoir des noms explicites ou implicites.

Les portes d'ordre représentent des relations d'ordre incomplètes dans lesquelles un événement du diagramme MSC sera ordonné par rapport à un événement de l'environnement. Les "portes d'ordre" sont toujours nommées explicitement. Les "portes d'ordre" sont considérées comme ayant un sens – allant de l'événement ordonné en premier jusqu'à l'événement lui succédant.

Les portes d'ordre sont également utilisées pour les références de diagramme MSC dans d'autres MSC. Les "portes d'ordre" réelles d'une référence MSC sont connectées à d'autres portes d'ordre ou à des événements.

Les portes de création représentent la possibilité de diviser l'événement de création à partir de l'instance créée.

Les portes des expressions en ligne sont identiques aux portes des cadres des diagrammes MSC et des références MSC. La seule différence est que le cadre de l'expression en ligne est à la fois le cadre de la définition de la porte (à l'intérieur) et le symbole de l'utilisation de la porte réelle (à l'extérieur). Si la porte d'expression en ligne proprement dite est directement dessinée vers un autre cadre (expression en ligne ou cadre de diagramme MSC), on peut omettre les flèches à l'intersection afin d'éviter une surcharge du diagramme avec des flèches très proches.

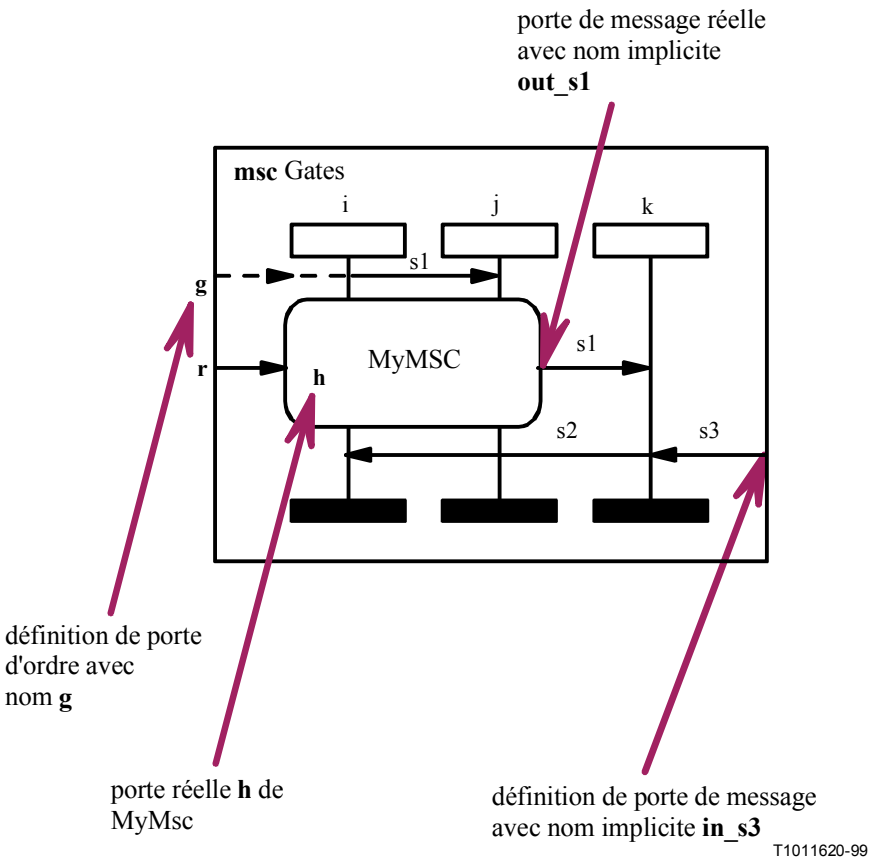


Figure 7/Z.120 – Exemple de portes

Grammaire textuelle concrète

<actual out gate> ::=
[<gate name>] **out** <msg identification> **to** <input dest>

<actual in gate> ::=
[<gate name>] **in** <msg identification> **from** <output dest>

<input dest> ::=
lost [<input address>] | <input address>

<output dest> ::=
found [<output address>] | <output address>

<def in gate> ::=
[<gate name>] **out** <msg identification> **to** <input dest>

<def out gate> ::=
[<gate name>] **in** <msg identification> **from** <output dest>

Si un nom de porte <gate name> n'est pas spécifié, un nom implicite est construit à l'aide du sens et de l'identificateur <msg identification> correspondant.

<actual order out gate> ::=
<gate name> **before** <order dest>

<order dest> ::=
<event name> | { **env** | <reference identification>} **via** <gate name>

Le nom <event name> fait référence à un événement ordonnable. Le nom <gate name> fait référence à l'un des ordres suivants: <def order out gate>, <actual order in gate>, <inline order out gate> ou <inline order in gate>.

<actual order in gate> ::=
<gate name>
[**after** <order dest list>]

<def order in gate> ::=
<gate name> **before** <order dest>

Le premier nom <gate name> définit le nom de cette "porte d'ordre".

<def order out gate> ::=
<gate name>
[**after** <order dest list>]

<actual create out gate> ::=
create out <create gate identification> **create** <create target>

<actual create in gate> ::=
create in <create gate identification>

<create target> ::=
<instance name> | { **env** | <reference identification>} [**via** <gate name>]

<def create in gate> ::=
create out [<create gate identification>] **create** <create target>

<def create out gate> ::=
create in <create gate identification>

Si le nom de porte <gate name> est omis, l'on suppose un nom implicite, donné par le sens et par l'identification de message <msg identification> correspondante. Le nom <gate name> définit le nom de cette porte d'ordre.

<inline out gate> ::=
<def out gate>
[**external out** <msg identification> **to** <input dest>]

<inline in gate> ::=
<def in gate>
[**external in** <msg identification> **from** <output dest>]

<inline out call gate> ::=
<def out call gate>
[**external call** <msg identification> **to** <input dest>]

<inline in call gate> ::=
<def in call gate>
[**external receive** <msg identification> **from** <output dest>]

<inline out reply gate> ::=
<def out reply gate>
[**external replyout** <msg identification> **to** <input dest>]

<inline in reply gate> ::=
<def in reply gate>
[**external replyin** <msg identification> **from** <output dest>]

<inline create out gate> ::=
<def create out gate>
[**external** <create>]

<inline create in gate> ::=
<def create in gate>
[**external create from** <create source>]

<create source> ::=
<instance name> |
{**env** | <reference identification>} [**via** <create gate identification>]

<inline order out gate> ::=
<gate name>
[[**after** <order dest list>] **external before** <order dest>]

<inline order in gate> ::=
<gate name> **before** <order dest>
[**external** [**after** <order dest list>]]

<actual out call gate> ::=
[<gate name>] **call** <msg identification> **to** <input dest>

<actual in call gate> ::=
[<gate name>] **receive** <msg identification> **from** <output dest>

<def in call gate> ::=
[<gate name>] **call** <msg identification> **to** <input dest>

<def out call gate> ::=
[<gate name>] **receive** <msg identification> **from** <output dest>

Si le nom de porte <gate name> est omis, l'on suppose un nom implicite, donné par le sens et par l'identification de message <msg identification> correspondante.

<actual out reply gate> ::=
[<gate name>] **replyout** <msg identification> **to** <input dest>

<actual in reply gate> ::=
[<gate name>] **replyin** <msg identification> **from** <output dest>

<def in reply gate> ::=
[<gate name>] **replyout** <msg identification> **to** <input dest>

<def out reply gate> ::=
[<gate name>] **replyin** <msg identification> **from** <output dest>

Si le nom de porte <gate name> est omis, l'on suppose un nom implicite, donné par le sens et par l'identification de message <msg identification> correspondante.

Grammaire graphique concrète

<inline gate area> ::=
{ <inline out gate area> | <inline in gate area> |
<inline create out gate area> | <inline create in gate area> |
<inline out call gate area> | <inline in call gate area> |
<inline out reply gate area> | <inline in reply gate area> }
[***is associated with*** <gate identification>]

<inline out gate area> ::=
<void symbol>
is attached to <inline expression symbol>
[***is attached to*** { <message symbol>| <found message symbol> }]
[***is attached to*** { <message symbol> | <lost message symbol> }]

<inline in gate area> ::=
<void symbol>
is attached to <inline expression symbol>
[***is attached to*** { <message symbol> | <lost message symbol>}]
[***is attached to*** { <message symbol> | <found message symbol>}]

<inline out call gate area> ::=
<void symbol>
is attached to <inline expression symbol>
[***is attached to*** { <message symbol>| <found message symbol> }]
[***is attached to*** { <message symbol> | <lost message symbol> }]

<inline in call gate area> ::=
<void symbol>
is attached to <inline expression symbol>
[***is attached to*** { <message symbol> | <lost message symbol>}]
[***is attached to*** { <message symbol> | <found message symbol>}]

<inline out reply gate area> ::=
<void symbol>
is attached to <inline expression symbol>
[***is attached to*** { <reply symbol>| <found reply symbol> }]
[***is attached to*** { <reply symbol> | <lost reply symbol> }]

<inline in reply gate area> ::=
<void symbol>
is attached to <inline expression symbol>
[***is attached to*** { <reply symbol> | <lost reply symbol>}]
[***is attached to*** { <reply symbol> | <found reply symbol>}]

<inline create out gate area> ::=
<void symbol>
is attached to <inline expression symbol>
[***is attached to*** <createline symbol>]
[***is attached to*** <createline symbol>]

<inline create in gate area> ::=
<void symbol>
is attached to <inline expression symbol>
[***is attached to*** <createline symbol>]
[***is attached to*** <createline symbol>]

Une porte d'expression en ligne est normalement attachée à un seul symbole de message à l'intérieur du cadre de l'expression en ligne, et à un symbole de message en dehors du cadre de l'expression en ligne. Lorsque aucun symbole n'est attaché à l'intérieur de la porte, cela signifie qu'un message incomplet est associé à cette porte.

<inline order gate area> ::=
<inline order out gate area> | <inline order in gate area>

<inline order out gate area> ::=
<void symbol>
is attached to <inline expression symbol>
is attached to <general order symbol>
[***is attached to*** <general order symbol>]

Le premier symbole <general order symbol> est une relation d'ordre général dans l'expression en ligne telle que l'événement dans l'expression est spécifié avant la porte. Le symbole facultatif <general order symbol> fait référence à une relation d'ordre général attachée à la porte en dehors de l'expression en ligne.

<inline order in gate area> ::=
<void symbol>
is attached to <inline expression symbol>
is attached to <general order symbol>
[***is attached to*** <general order symbol>]

Le premier symbole <general order symbol> est une relation d'ordre général dans l'expression en ligne telle que la porte est spécifiée avant l'événement dans l'expression. Le symbole facultatif <general order symbol> fait référence à une relation d'ordre général attachée à la porte en dehors de l'expression en ligne.

<def gate area> ::=
{ <def out gate area> |<def in gate area>|
<def order out gate area> | <def order in gate area>
<def out call gate area> | <def in call gate area>
<def out reply gate area> | <def in reply gate area>
<def create out gate area> | <def create in gate area>}
is attached to <msc symbol>

<def out gate area> ::=
<void symbol> [***is associated with*** <gate identification>]
is attached to { <message symbol> | <found message symbol> }

Le symbole de message <message symbol> ou le symbole de message trouvé <found message symbol> doit avoir sa pointe de flèche attachée à la zone de définition de la porte de sortie <def out gate area>.

<def in gate area> ::=
<void symbol>[***is associated with*** <gate identification>]
is attached to { <message symbol> | <lost message symbol> }

Le symbole de message <message symbol> ou le symbole de message perdu <lost message symbol> doit avoir son extrémité ouverte attachée à la zone de définition de la porte d'entrée <def in gate area>.

<def out call gate area> ::=
<void symbol> [***is associated with*** <gate identification>]
is attached to { <message symbol> | <found message symbol> }

Le symbole de message <message symbol> ou le symbole de message trouvé <found message symbol> doit avoir sa pointe de flèche attachée à la zone de définition de la porte d'appel centrifuge <def out call gate area>.

<def in call gate area> ::=
<void symbol>[***is associated with*** <gate identification>]
is attached to { <message symbol> | <lost message symbol> }

Le symbole de message <message symbol> ou le symbole de message perdu <lost message symbol> doit avoir son extrémité ouverte attachée à la zone de définition de la porte d'appel entrant <def in call gate area>.

<def out reply gate area> ::=
<void symbol> [***is associated with*** <gate identification>]
is attached to { <reply symbol> | <found reply symbol> }

Le symbole de réponse <reply symbol> ou le symbole de réponse trouvée <found reply symbol> doit avoir sa pointe de flèche attachée à la zone de définition de la porte de réponse centrifuge <def out reply gate area>.

<def in reply gate area> ::=
<void symbol>[***is associated with*** <gate identification>]
is attached to { <reply symbol> | <lost reply symbol> }

Le symbole de réponse <reply symbol> ou le symbole de réponse perdue <lost reply symbol> doit avoir son extrémité ouverte attachée à la zone de définition de la porte de réponse entrant <def in reply gate area>.

<def order out gate area> ::=
<void symbol>[***is associated with*** <gate identification>]
is attached to <general order area>

<def order in gate area> ::=
<void symbol>[***is associated with*** <gate identification>]
is followed by <general order area>

<def create out gate area> ::=
<void symbol> [***is associated with*** <create gate identification>]
is attached to <createline symbol>

Le symbole de création de ligne <createline symbol> doit avoir sa pointe de flèche attachée à la zone de définition de la porte de création centrifuge <def create out gate area>.

<def create in gate area> ::=
<void symbol>[***is associated with*** <create gate identification>]
is attached to <createline symbol>

Le symbole de création de ligne <createline symbol> doit avoir son extrémité ouverte attachée à la zone de définition de la porte de création entrant <def create in gate area>.

<gate identification> ::=
<gate name>

<create gate identification> ::=
[<gate identification> :] <instance kind>

Le symbole <instance kind> désigne la sorte de l'instance à créer. La sorte d'instance peut également correspondre à un nom d'instance <instance name> particulier.

<actual gate area> ::=
<actual out gate area> | <actual in gate area> |
<actual out call gate area> | <actual in call gate area> |
<actual out reply gate area> | <actual in reply gate area> |
<actual create out gate area> | <actual create in gate area> |
<actual order out gate area> | <actual order in gate area>

<actual out gate area> ::=
<void symbol> [***is associated with*** <gate identification>]
is attached to <msc reference symbol>
[***is attached to*** { <message symbol> | <lost message symbol> }]

Le symbole <actual out gate area> est attaché à l'extrémité ouverte du symbole <message symbol> ou <lost message symbol>.

<actual in gate area> ::=
<void symbol> [***is associated with*** <gate identification>]
is attached to <msc reference symbol>
[***is attached to*** { <message symbol> | <found message symbol> }]

Le symbole <actual in gate area> est attaché à la pointe de flèche du symbole <message symbol> ou <found message symbol>.

<actual out call gate area> ::=
<void symbol> [***is associated with*** <gate identification>]
is attached to <msc reference symbol>
[***is attached to*** { <message symbol> | <lost message symbol> }]

Le symbole <actual out call gate area> est attaché à l'extrémité ouverte du symbole <message symbol> ou <lost message symbol>.

<actual in call gate area> ::=
<void symbol> [***is associated with*** <gate identification>]
is attached to <msc reference symbol>
[***is attached to*** { <message symbol> | <found message symbol> }]

Le symbole <actual in call gate area> est attaché à la pointe de flèche du symbole <message symbol> ou <found message symbol>.

<actual out reply gate area> ::=
<void symbol> [***is associated with*** <gate identification>]
is attached to <msc reference symbol>
[***is attached to*** { <reply symbol> | <lost reply symbol> }]

Le symbole <actual out reply gate area> est attaché à l'extrémité ouverte du symbole <reply symbol> ou <lost reply symbol>.

<actual in reply gate area> ::=
<void symbol> [***is associated with*** <gate identification>]
is attached to <msc reference symbol>
[***is attached to*** { <reply symbol> | <found reply symbol> }]

Le symbole <actual in reply gate area> est attaché à la pointe de flèche du symbole <reply symbol> ou <found reply symbol>.

<actual create out gate area> ::=
<void symbol> [***is associated with*** <create gate identification>]
is attached to <msc reference symbol>
[***is attached to*** <createline symbol>]

Le symbole <actual create out gate area> est attaché à l'extrémité ouverte du symbole <createline symbol>.

<actual create in gate area> ::=
<void symbol> [***is associated with*** <create gate identification>]
is attached to <msc reference symbol>
[***is attached to*** <createline symbol>]

Le symbole <actual create in gate area> est attaché à la pointe de flèche du symbole <createline symbol>.

<actual order out gate area> ::=
<void symbol> [***is associated*** with <gate identification>]
is attached to <msc reference symbol>
is followed by <general order area>

<actual order in gate area> ::=
<void symbol>[***is associated*** with <gate identification>]
is attached to <msc reference symbol>
is attached to <general order area>

Exigences statiques

Les noms de porte définis <gate name> d'un diagramme MSC sont autorisés à être ambigus mais les références à des noms de porte ambigus <gate name> ne sont pas autorisées.

Pour les connexions de porte, la définition de porte associée doit toujours correspondre à la porte réelle. Pour des messages, cette correspondance signifie que le message attaché à la définition de porte est du même type que le message attaché à la porte réelle. Il y également lieu que les sens correspondent.

La sorte de la définition de porte doit toujours être la même que celle de la porte réellement connectée. Les sortes des portes sont les suivantes: portes de messages, portes d'ordre et portes de création.

4.6 Relation d'ordre général

La relation d'ordre général est utilisée pour imposer des ordonnancements additionnels à des événements qui ne sont pas définis par la relation d'ordre général indiquée par la sémantique MSC. Par exemple, pour spécifier qu'un événement relatif à une instance doit se produire avant un événement relatif à une autre instance, non associé par ailleurs, ou pour spécifier des ordonnancements d'événements contenus dans une corégion.

Grammaire textuelle concrète

La grammaire textuelle est décrite au 4.1.

Grammaire graphique concrète

<general order area> ::=
<general order symbol>
is attached to <ordered event area>

<general order symbol> ::=
<general order symbol1> | <general order symbol2>

<general order symbol1> ::=

Le symbole <general order symbol1> peut avoir une forme d'escalier, c'est-à-dire une succession de segments horizontaux et verticaux consécutifs dans la même relation d'ordre. En d'autres termes, cette relation peut comporter des segments verticaux et des segments horizontaux. Les segments d'un

symbole <general order symbol1> peuvent se superposer, mais seulement si aucune ambiguïté avec les autres symboles <general order symbol1> impliqués n'apparaît. Cela signifie qu'aucun nouvel ordre ne peut être déduit.

Le symbole <general order symbol1> ne peut apparaître que dans un symbole <instance axis symbol2> (forme de colonne). Les lignes de connexion entre le symbole <general order symbol1> et la zone des événements <ordered event area> ordonnés doivent être horizontales.

<general order symbol2> ::=

Le symbole <general order symbol2> peut avoir n'importe quelle orientation et même être incurvé.

<ordered event area> ::=
<actual order in gate area>
| <actual order out gate area>
| <def order in gate area>
| <def order out gate area>
| <inline order gate area>
| <message event area>
| <incomplete message area>
| <method call event area>
| <incomplete method call area>
| <reply event area>
| <incomplete reply area>
| <timer area>
| <create area>
| <action symbol>

Exigences statiques

L'ordre partiel des événements défini par les constructions d'ordonnancement général et par les messages n'est pas réflexif.

Sémantique

Les symboles d'ordre général décrivent l'ordre des événements qui sinon ne seraient pas ordonnés. Cet ordre est particulièrement utile lorsque les événements d'une corégion doivent suivre une relation d'ordre plus stricte que toutes les permutations possibles.

En notation graphique, lorsque l'on utilise le symbole <general order symbol1>, l'événement qui est graphiquement supérieur doit toujours se produire avant l'événement inférieur. Lorsque l'on utilise le symbole <general order symbol2>, la flèche va d'un événement à celui qui le suit dans le temps.

La grammaire textuelle définit les relations d'ordre partiel par les mots clés **before** et **after**, qui indiquent directement l'ordre dans lequel les événements en cause doivent apparaître dans les traces légales. Les relations temporelles sont définies par le mot clé **time**. Elles indiquent l'ordre quantifié dans le temps dans lequel les événements en cause doivent apparaître dans les traces légales.

Exemples de relation d'ordre général

Les quatre exemples de la Figure 8 illustrent le même ordonnancement général. Les ordres suivants sont définis:

a **avant** b, a **avant** d, c **avant** b, c **avant** d. a et c ne sont pas ordonnés et b et d ne sont pas ordonnés.

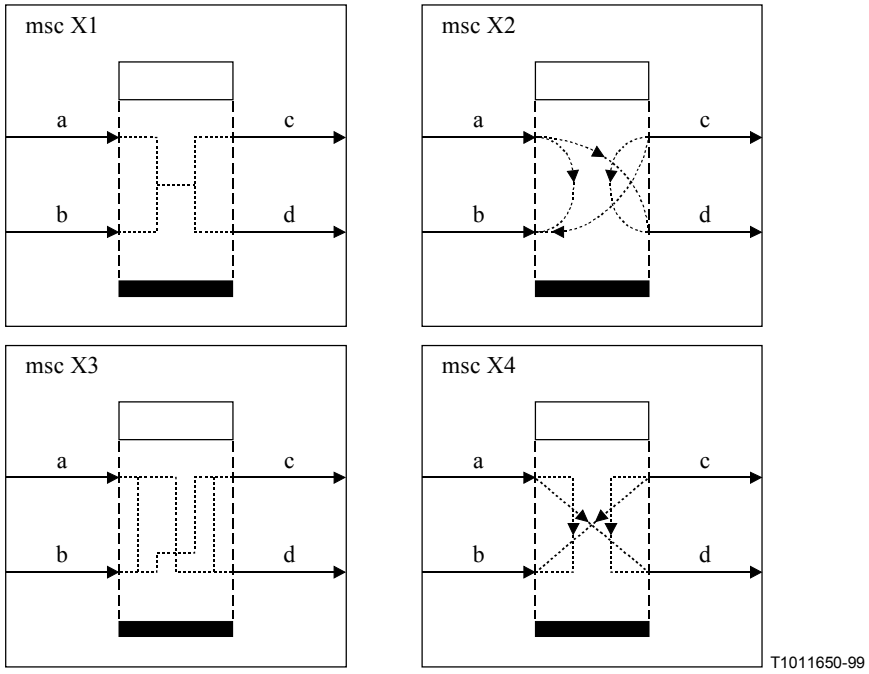


Figure 8/Z.120 – Exemple de relation d'ordre général

4.7 Condition

Les conditions globales peuvent être utilisées pour restreindre les traces possibles d'un diagramme MSC, ou la façon dont ces traces entrent dans la composition des diagrammes MSC de haut niveau. Il existe deux types de condition: les conditions de *réglage* et les conditions de *garde*. Les conditions de réglage définissent ou décrivent l'état actuel du système global (condition globale) ou un certain état non global (condition non globale). Dans ce dernier cas, la condition peut être locale, c'est-à-dire attachée à une seule instance.

Les conditions de garde restreignent le comportement d'un diagramme MSC en n'autorisant l'exécution d'événements dans une certaine partie du diagramme MSC qu'en fonction de leurs valeurs. Cette partie, qui est le domaine de visibilité de la condition, est soit le diagramme MSC complet soit l'opérande le plus intérieur d'une expression en ligne ou une branche d'un diagramme HMSC. La garde doit être placée au début de ce domaine de visibilité. La condition est soit un état (global ou non global), dans lequel le système doit se trouver (selon la définition donnée par les conditions de réglage préalable), soit une expression booléenne dans le langage des données.

Dans la représentation textuelle, la condition doit être définie pour chaque instance à laquelle elle est attachée au moyen du mot clé **condition** accompagné du nom de condition. Si la condition se rapporte à plusieurs instances, il faut ajouter le mot clé **shared** ainsi que la liste des instances auxquelles la condition se rapporte. Une condition globale (se rapportant à toutes les instances) est définie par le mot clé **shared all**. Les conditions de garde sont représentées avec les mots clés **condition** et **when**. Les conditions de réglage ne sont représentées qu'avec le mot clé **condition**. Dans la syntaxe graphique, le mot clé **when** est inclus dans le symbole de condition pour les conditions de garde; pour les conditions de réglage, il n'y a pas de mot clé ou de symbole supplémentaire.

Les conditions de garde peuvent être utilisées par exemple dans l'expression en ligne (voir 7.2) afin de déterminer quel opérande d'expression **alt** est applicable en fonction des choix qui ont été faits dans les expressions **alt** précédentes, ou dans les diagrammes HMSC (voir 7.5).

Grammaire textuelle concrète

<shared condition> ::=
[<shared>] <condition identification> <shared> <end>

<condition identification> ::=
condition <condition text>

<condition text> ::=
<condition name list> | **when** { <condition name list> | (<expression>) } |
otherwise

<condition name list> ::=
<condition name>{ , <condition name> }*

<shared> ::=
shared { [<shared instance list>] | **all** }

<shared instance list> ::=
<instance name> [**,** <shared instance list>]

<condition> ::=
[<shared>] <condition identification> <end>

Le symbole facultatif <shared> avant le symbole <condition identification> dans la grammaire textuelle concrète est analogue au symbole facultatif <shared> dans la grammaire graphique concrète (voir ci-dessous).

Exigences statiques

Pour chaque nom d'instance <instance name> contenu dans une liste d'instances partagées <shared instance list> d'une <condition>, une instance avec une <condition> partagée correspondante doit être spécifiée. Si l'instance *b* fait partie de la liste <shared instance list> d'une <condition> partagée attachée à une instance *a*, alors l'instance *a* doit faire partie de la liste <shared instance list> de la <condition> partagée correspondante attachée à l'instance *b*. Si les instances *a* et *b* partagent la même <condition>, alors pour chaque message échangé entre ces instances, le <message input> et le <message output> doivent être placés tous les deux, soit avant, soit après la <condition>.

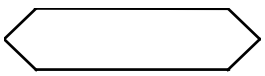
Si deux conditions sont ordonnées directement (parce qu'elles ont une instance en commun) ou ordonnées indirectement par des conditions relatives à d'autres instances, cet ordre doit être respecté pour toutes les instances partageant ces deux conditions. En fonction des possibilités d'une description syntaxique *orientée vers les instances* ou *orientée vers les événements* (voir 4.1), il existe deux formes syntaxiques pour les conditions: la description *orientée vers les instances* utilise la forme <shared condition>, tandis que la description *orientée vers les événements* utilise une forme à instances multiples faisant appel à une liste d'événements à instances multiples <multi instance event list> suivie du signe deux-points et du symbole non terminal <condition> (voir 4.1). Dans la forme <shared condition>, le mot clé **shared** est utilisé également, de façon logique, pour les conditions locales. A l'intérieur du symbole non terminal <shared>, la liste d'instances partagées <shared instance list> est donc facultative.

Le mot clé **otherwise** ne peut apparaître que comme garde d'exactement un seul opérande d'une expression en alternative.

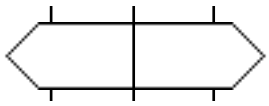
Grammaire graphique concrète

<condition area> ::=
<condition symbol> ***contains*** <condition text> [<shared>]
is attached to { <instance axis symbol>* } ***set***

<condition symbol> ::=



La zone <condition area> peut faire référence à une seule instance, ou être attachée à plusieurs instances. Si une <condition> partagée croise un symbole d'axe d'instance <instance axis symbol> qui n'est pas concerné par la condition, ce symbole est dessiné comme suit:



Les instances partagées <shared> ne possèdent pas de zone d'instance <instance area> dans le diagramme car il n'y a pas d'événements relatifs à ces instances mais ces instances partagées restent couvertes par la condition. S'il y a de quelconques ambiguïtés concernant la relation d'ordre applicable à des instances partagées, ces ambiguïtés doivent être représentées explicitement dans le diagramme avec une zone d'instance <instance area>.

Exigences statiques

Une condition de garde doit toujours être placée au début du domaine de visibilité correspondant, ce domaine étant soit un diagramme MSC complet soit un opérande d'expression en ligne ou une branche de diagramme HMSC.

Une condition de garde doit toujours couvrir toutes les instances prêtes de son champ de visibilité. Une instance est dite prête si elle contient une évènement pouvant être exécuté avant tout autre évènement du champ de visibilité.

Si une garde contient une expression de données, cette expression doit être de type booléen. Si cette expression contient en outre des variables dynamiques, elle ne peut couvrir qu'une seule instance, qui doit alors être la seule instance prête du domaine de visibilité.

Sémantique

Les conditions de réglage définissent l'état système réel de l'instance (des instances) partageant ces conditions. Les conditions de garde peuvent être utilisées pour limiter les possibilités de continuité d'un diagramme MSC.

Les événements du domaine gardé par une condition de garde ne peuvent être exécutés que si cette condition de garde est "Vrai" au moment de l'exécution du premier de ces événements. Si la condition est une expression dans le langage de données, cela signifie qu'elle a la valeur "Vrai". Si la garde est un ensemble de noms de condition, la *dernière* condition de réglage de cet ensemble d'instances doit avoir une intersection non vide avec cette garde. Seules sont vérifiées les conditions de réglage concernant exactement les mêmes instances; les conditions qui règlent l'état d'un sous-ensemble ou d'un surensemble de ces instances ne sont pas vérifiées.

Des interprétations spécifiques des gardes figurant dans des expressions en ligne sont données en 7.2. La garde du mot clé **otherwise** a la valeur "Vrai" *(true)* si les gardes de tous les autres opérandes de l'expression en alternative ont la valeur "Faux" *(false)*.

4.8 Temporisateur

Les diagrammes MSC permettent de spécifier soit l'armement d'un temporisateur puis son expiration, soit l'armement d'un temporisateur puis son désarmement (arrêt). De plus, les constructions indépendantes des temporisateurs – l'armement, le désarmement et l'expiration d'un temporisateur – peuvent être utilisées séparément, par exemple dans le cas où une expiration ou une supervision d'un temporisateur est partagée entre différents MSC. Dans la représentation graphique, le symbole d'armement a la forme d'un sablier relié à l'axe de l'instance par une ligne. L'expiration est décrite par une flèche de message pointant sur l'instance attachée au symbole du sablier. Le symbole de désarmement a la forme d'une croix connectée à l'instance par une ligne.

La spécification d'un nom d'instance de temporisateur et de la durée de temporisation est facultative, aussi bien dans la représentation textuelle que graphique.

Un temporisateur peut être armé avec ou sans durée. Pour la durée, il est possible d'exprimer une limite supérieure et une limite inférieure de l'événement d'expiration. La limite supérieure d'expiration peut être définie comme étant l'infini, qui est représenté par le mot clé **inf**. Les événements d'armement, de désarmement et d'expiration d'un temporisateur peuvent être également soumis à des contraintes temporelles. Ce type de temporisation est alors représenté dans la relation temporelle (**time**) des événements ordonnables.

La période d'expiration d'un temporisateur est définie comme suit:

[0, **inf**),	si aucune durée n'est indiquée
[<duration min>, **inf**),	si seule la durée minimale est indiquée
[0, <duration max>],	si seule la durée maximale est indiquée
[<duration min>, <duration max>],	si une durée minimale et une durée maximale sont indiquées

Grammaire textuelle concrète

<timer statement> ::=
<starttimer> | <stoptimer> | <timeout>

<starttimer> ::=
starttimer <timer name> [**,** <timer instance name>]
[<duration>] [(<parameter list>)]

<duration> ::=
<left square bracket>
[<min durationlimit>] [, <max durationlimit>] <right square bracket>

<durationlimit> ::=
<expression string> | **inf**

<stoptimer> ::=
stoptimer <timer name> [**,** <timer instance name>]

<timeout> ::=
timeout <timer name> [**,** <timer instance name>]
[(<parameter list>)]

Si le nom de temporisateur <timer name> n'est pas suffisant pour un mappage biunivoque, le nom d'instance <timer instance name> doit être utilisé.

Exigences statiques

La liste de paramètres <parameter list> du temporisateur de déclenchement <starttimer> n'est composée que d'expressions <expression>. La liste de paramètres <parameter list> de l'expiration <timeout> n'est composée que de structures <pattern>.

Grammaire graphique concrète

<timer area> ::=
{ <timer start area> | <timer stop area> | <timeout area> }
{***is followed by*** <general order area> }*
{***is attached to*** <general order area> }*

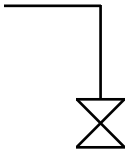
<timer start area> ::=
<timer start area1> | <timer start area2>

<timer start area1> ::=
<timer start symbol> ***is associated with*** <timer name>
[<duration>] [(<parameter list>)]
is attached to <instance axis symbol>
[***is attached to***
{<int symbol> | <abs time symbol> }*]
[***is attached to*** {<restart symbol> | <timer stop symbol2>
| <timeout symbol3> }]

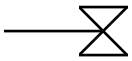
<timer start area2> ::=
<restart symbol> ***is associated with*** <timer name>
[<duration>] [(<parameter list>)]
is attached to <instance axis symbol>
[***is attached to***
{<int symbol> | <abs time symbol> }*]
is attached to <timer start symbol>
[***is attached to*** { <timer stop symbol2> | <timeout symbol3> }]

<timer start symbol> ::=
<start symbol1> | <start symbol2>

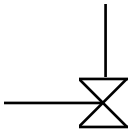
<start symbol1> ::=



<start symbol2> ::=



<restart symbol> ::=



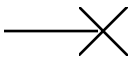
<timer stop area> ::=
<timer stop area1> | <timer stop area2>

<timer stop area1> ::=
<timer stop symbol1> ***is associated with*** <timer name>
is attached to <instance axis symbol>
[***is attached to***
{<int symbol> | <abs time symbol> }*]

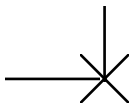
<timer stop area2> ::=
<timer stop symbol2> ***is associated with*** <timer name>
is attached to <instance axis symbol>
[***is attached to***
{<int symbol> | <abs time symbol> }*]
is attached to { <timer start symbol> | <restart symbol> }

<timer stop symbol> ::=
<timer stop symbol1> | <timer stop symbol2>

<timer stop symbol1> ::=



<timer stop symbol2> ::=



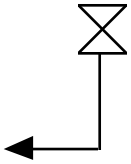
<timeout area> ::=
<timeout area1> | <timeout area2>

<timeout area1> ::=
<timeout symbol> ***is associated with*** <timer name>
[(<parameter list>)]
is attached to <instance axis symbol>
[***is attached to***
{<int symbol> | <abs time symbol> }*]

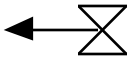
<timeout symbol> ::=

<timeout symbol1> | <timeout symbol2>

<timeout symbol1> ::=



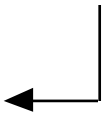
<timeout symbol2> ::=



<timeout area2> ::=

<timeout symbol3> [***is associated with*** <timer name>
(<parameter list>)]
is attached to <instance axis symbol>
[***is attached to***
{<int symbol> | <abs time symbol> }*]
is attached to { <timer start symbol> | <restart symbol> }

<timeout symbol3> ::=



Exigences statiques

La liste de paramètres <parameter list> d'une zone d'armement de temporisateur <timer start area> n'est composée que d'expressions <expression>. La liste de paramètres <parameter list> d'une zone d'expiration <timeout area> n'est composée que de structures <pattern>.

Sémantique

L'armement désigne le déclenchement du temporisateur et le désarmement désigne l'arrêt du temporisateur. L'expiration du temporisateur correspond à la consommation du signal de temporisation.

Si des événements de temporisation coïncident avec d'autres événements, voir les règles de dessin du 2.4.

4.9 Action

En plus de l'échange de messages, des actions peuvent être spécifiées dans les diagrammes MSC. Une action est un événement atomique auquel on peut associer soit un texte informel soit des déclarations formelles de données. Un texte informel est délimité par des apostrophes simples pour le distinguer des déclarations de données, qui se composent d'une liste de déclarations séparées par des virgules, chaque déclaration pouvant être de type association, de type définition ou de type non-définition.

Grammaire textuelle concrète

<action> ::=

action <action statement>

<action statement> ::=

<informal action> | <data statement list>

<informal action> ::=

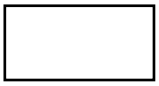
<character string>

Grammaire graphique concrète

<action area> ::=

<action symbol>
is attached to <instance axis symbol>
[***top or bottom is attached to top or bottom***
{ {<int symbol> | <abs time symbol> }* } ***set***]
{ ***is followed by*** <general order area> }*
{ ***is attached to*** <general order area> }*
contains <action statement>

<action symbol> ::=



Si l'axe d'instance a la forme d'une colonne, la largeur du symbole d'action <action symbol> doit coïncider avec la largeur de la colonne.

Sémantique

Une action décrit une activité atomique interne à une instance. Lorsqu'une action contient des déclarations de données, l'événement modifie l'état par évaluation simultanée de chaque déclaration. Cette simultanéité reflète l'atomicité d'une action.

4.10 Création d'instance

De même qu'en SDL, la création et la terminaison d'une instance peuvent être spécifiées dans les diagrammes MSC. Une instance peut être créée par une autre instance. Aucun événement de messagerie ne peut être attaché à une instance avant la création de celle-ci.

Grammaire textuelle concrète

<create> ::=

create <instance name> [(<parameter list>)]

Pour chaque symbole <create>, il doit y avoir une instance correspondante ayant le nom spécifié. Une instance ne peut être créée qu'une seule fois, c'est-à-dire que, à l'intérieur d'un seul diagramme MSC *simple*, deux ou plus de deux symboles <create> ayant le même nom ne doivent pas apparaître.

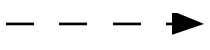
Grammaire graphique concrète

<create area> ::=

<createline symbol> [***is associated with*** <parameter list>]
is attached to
{ {<instance axis symbol> | <def create in gate area> |
<actual create out gate area> }
{ <instance head area> | <def create out gate area> |
<actual create in gate area> }} ***set***
{ ***is followed by*** <general order area> }*
{ ***is attached to*** <general order area> }*
[***is attached to***
{{<int symbol> | <abs time symbol>}*}***set***]

L'événement de création est désigné par l'extrémité du symbole de création de ligne <createline symbol> qui n'a pas de pointe de flèche. L'événement de création est attaché à un axe d'instance. Si la zone de création <create area> est en relation d'ordre général, cet ordre s'applique à l'événement de création. Si une contrainte temporelle est assignée, elle s'applique à l'événement de création. La pointe de flèche désigne le symbole d'en-tête d'instance <instance head symbol> de l'instance créée.

<createline symbol> ::=



L'image miroir du symbole <createline symbol> est autorisée.

Sémantique

L'événement de création définit la création dynamique d'une instance par une autre. Dynamiquement, il ne peut y avoir qu'une seule création au cours de la durée de vie d'une instance et aucun événement relatif à l'instance ne peut avoir lieu avant la création de celle-ci.

Si la création de l'instance coïncide avec d'autres événements, voir les règles de dessin du 2.4.

4.11 Terminaison d'instance

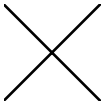
La terminaison d'une instance est le pendant de la création d'une instance, sauf qu'une instance décide elle-même de sa terminaison, alors qu'elle a été créée par une autre instance.

Grammaire textuelle concrète

<stop> ::=
stop <end>

Grammaire graphique concrète

<stop symbol> ::=



Sémantique

L'arrêt à la fin d'une instance provoque la terminaison de cette instance.

Dynamiquement, il ne peut y avoir qu'une seule création au cours de la durée de vie d'une instance et aucun événement relatif à l'instance ne peut avoir lieu avant la création de celle-ci.

5 Concepts relatifs aux données

5.1 Introduction

Les données sont incorporées aux diagrammes MSC à un certain nombre d'emplacements, comme la transmission de messages, les cadres d'action et les références MSC. Les données sont utilisées de deux manières distinctes: statiquement comme lors du paramétrage d'un diagramme MSC ou dynamiquement comme lors de l'acquisition d'une valeur par réception de message. Pour que les diagrammes MSC puissent être utilisés avec les langages de données choisis par l'utilisateur, aucune langage spécifique n'est défini pour les diagrammes MSC. Ceux-ci disposent en revanche d'un certain nombre d'emplacements dans leur définition où il faut introduire une chaîne représentant un aspect de l'usage des données, comme des expressions ou des déclarations de type. Pour définir la sémantique des diagrammes MSC au moyen de données, il faut un certain nombre de fonctions qui extraient les informations requises de ces chaînes afin de les traduire en concepts de données MSC.

5.2 Interface syntaxique avec des langages externes de données

Lorsqu'une chaîne terminale du langage de données est utilisée, le nom de la production est préfixé de mots soulignés pour indiquer son rôle d'interface réel. Par exemple, la structure <variable string> est une chaîne terminale qui doit être analysée comme étant une variable dans le langage de données. La liste des chaînes de données terminales est la suivante:

- <variable string>;
- <type ref string>;
- <data definition string>;

- <expression string>.

Afin que les données puissent être distinguées dans les diagrammes MSC, des délimiteurs de parenthèse et des caractères d'échappement sont définis. La syntaxe propre aux chaînes de données est indiquée au 2.1. L'objet visé est que l'on puisse reconnaître les chaînes de données contenues dans le langage de données sans usage excessif de séquences d'échappement et de délimiteurs spéciaux.

Grammaire textuelle concrète

```
<parenthesis declaration> ::=
                        parenthesis <par decl list> <end>

<par decl list> ::=
                        { <nestable par pair> | <equal par delim> | <non-nestable par pair>
                        | <escape decl>}
                        [, <par decl list>]

<nestable par pair> ::=
                        nestable <left delim>, <right delim> <end>

<equal par delim> ::=
                        equalpar <left delim> <end>

<non-nestable par pair> ::=
                        nonnestable <left delim>, <right delim> <end>

<escape decl> ::=
                        escape <escapechar>

<left delim> ::=
                        <delim> <open par> <delim>

<right delim> ::=
                        <delim> <close par> <delim>

<delim> ::=
                        <apostrophe>
                    |   <alphanumeric>
                    |   <other character>
                    |   <special>
                    |   <full stop>
                    |   <underline>

<open par> ::=
                        <character string>

<close par> ::=
                        <character string>

<escapechar> ::=
                        <character string>
```

Les caractères délimiteurs <delim> sont les mêmes des deux côtés des délimiteurs gauche et droit déclarés. L'interprétation est que le caractère délimiteur choisi ne doit pas apparaître dans le texte définissant les parenthèses d'ouverture et de fermeture.

Sémantique

L'interface syntaxique donne des instructions sur la façon dont l'analyseur lexical doit extraire les chaînes de données à partir de la notation MSC dans son ensemble. L'analyseur lexical reçoit le début d'une chaîne de données et applique les déclarations de parenthèses afin de déterminer le moment de terminer cette chaîne et de la transmettre à l'analyseur de langage de données qui est extérieur à l'interprétation des diagrammes MSC proprement dite.

Il existe 3 types de parenthèses et un seul mécanisme d'échappement. Voir 2.1 pour l'explication syntaxique plus précise de ces quatre concepts. Une parenthèse imbricable est une parenthèse dont la

chaîne comprise entre les délimiteurs de gauche et de droite peut aussi contenir des parenthèses qui devront être reconnues. La mise entre parenthèses doit finalement être équilibrée.

Les parenthèses égales utilisent le même délimiteur à gauche et à droite. De telles parenthèses ne peuvent donc pas être imbriquées car on ne peut pas distinguer les délimiteurs de gauche et de droite.

Finalement, il y a les parenthèses non imbricables dont l'intérieur sera simplement ignoré. L'analyseur lexical recherchera le délimiteur droit correspondant.

Le mécanisme d'échappement est interprété de manière que la chaîne qui suit le caractère d'échappement doit être considérée comme formant l'intérieur d'une parenthèse et non un caractère de délimitation ou d'échappement.

5.3 Interface sémantique avec des langages externes de données

Afin de définir la sémantique d'un diagramme MSC au moyen de données, il faut fournir un certain nombre de données relatives aux chaînes de données. Ces fonctions vont de celles qui garantissent que les chaînes de données sont bien formées en termes syntaxiques aux fonctions sémantiques qui permettent d'évaluer les expressions de données. En d'autres termes, la sémantique d'un diagramme MSC est paramétrée par ces fonctions de données et un outil d'analyse MSC ne devra être fourni qu'avec les instances de ces fonctions pour calculer la sémantique MSC. Les fonctions peuvent être séparées en celles qui sont requises pour les exigences statiques et les autres pour la sémantique dynamique.

Fonctions d'interface avec les exigences statiques

Pour vérifier que chacune des quatre sortes de chaîne <string> de données est conforme à leur grammaire, quatre prédicats de *bienfacture* sont requis. Ils ne doivent être vérifiés que lorsque leurs chaînes d'argument sont analysées correctement. Il faut donc quatre prédicats, Wf1, Wf2, Wf3 et Wf4, ayant les signatures suivantes:

Wf1: <variable string> → *Bool*

Wf2: <data definition string> → *Bool*

Wf3: <type ref string> → *Bool*

Wf4 <expression string> → *Bool*

Pour s'assurer que les chaînes de données non seulement sont analysées correctement mais encore sont conformes aux exigences statiques du langage, il faut définir trois fonctions de *vérification de type*. La première de ces fonctions (Tc1 pour les chaînes de définition de données <data definition string>) ne nécessite pas d'arguments additionnels.

Tc1: <data definition string> → *Bool*

La deuxième fonction, Tc2, permet de vérifier les chaînes de référence de type <type ref string>. Mais comme ces chaînes peuvent contenir des informations définies dans les chaînes de définition de données <data definition string>, comme pour la définition de constantes utilisées pour fournir des limites de tableau ou des références à des définitions de type, cette fonction doit prendre ces informations comme paramètres.

Tc2: <data definition string> → <type ref string> → *Bool*

La fonction de vérification de type pour les chaînes d'expression <expression string>, Tc3, doit être paramétrée par la chaîne de définition de données <data definition string>, en même temps que les informations définies par les déclarations de variable et de caractère générique. Ces dernières informations peuvent être codées sous la forme d'une série de paires dont le premier élément est une chaîne de variable <variable string> et dont le second est sa chaîne de référence de type <type ref string> correspondante, que cette chaîne de variable se rapporte à une déclaration de

variable ou à une déclaration de caractère générique. La signature de la fonction Tc3 est donc la suivante:

Tc3: <data definition string> × (<variable string> × <type ref string>)-*set* →

<expression string> → *Bool*

En plusieurs endroits, les exigences statiques des diagrammes MSC prescrivent que les noms de variable soient uniques: il faut donc une fonction EqVar pour comparer les noms de variable dans l'interface:

EqVar: <variable string> × <variable string>) → *Bool*

La dernière fonction est utilisée pour vérifier la conformité de type entre différentes chaînes de données, par exemple pour vérifier que le type d'une variable est le même que celui d'une expression à laquelle il va être assigné. Il suffit d'avoir une fonction permettant de vérifier qu'une expression a un type requis. Il faut de nouveau extraire des informations contextuelles des chaînes de définition de données et des déclarations de variable ou de caractère générique.

Tc4: <data definition string> × (<variable string> × <type ref string>)-*set* →

(<type ref string> × <expression string>) → *Bool*

Fonctions d'interface avec la sémantique dynamique

Quatre fonctions sont nécessaires pour définir avec des données la sémantique dynamique des diagrammes MSC. Les trois premières sont des fonctions agissant sur la syntaxe du langage de données. Seule la quatrième, Eval, est une fonction d'évaluation sémantique. Les fonctions syntaxiques sont requises pour traiter des expressions de données qui contiennent des caractères génériques car celles-ci ont une sémantique différente des expressions simples que l'on trouve dans de nombreux langages de données. Au moyen des trois premières fonctions, la sémantique dynamique effectue des transformations à partir d'expressions contenant des caractères génériques afin de créer des expressions n'en contenant plus.

La première fonction, Vars, sert à extraire les chaînes de variable <variable string> d'une chaîne d'expression <expression string>. Elle est nécessaire aussi bien pour effectuer des contrôles de sémantique dynamique (comme l'écriture de variables avec leur lecture) que pour identifier des caractères génériques. Comme la sémantique traite de multiples caractères génériques apparaissant dans une même expression comme des grandeurs indépendantes, la fonction Vars doit déterminer combien de fois une chaîne de variable <variable string> apparaît dans une expression. La fonction Vars est exactement comme la fonction Tc4, qui est paramétrée par des chaînes de définition de données <data definition string> et par des informations de déclaration de variable ou de caractère générique. Le résultat de l'application de la fonction Vars à une expression de données est un ensemble de paires, dont le premier élément est une chaîne de variable <variable string> et le second le nombre d'apparitions de la variable dans l'expression. Seules les variables apparaissant dans l'expression doivent apparaître dans le résultat de fonction. La signature de la fonction *Vars* est donnée par les expressions suivantes:

Vars: <data definition string> × (<variable string> × <type ref string>)-*set*→

<expression string> → (<variable string> × *Nat*)-*set*

Par exemple, l'application de la fonction Vars à l'expression "f(x, c, x + y)", dans laquelle x et y sont déclarés comme des variables/caractères génériques et c est une constante, produira l'ensemble:

{ (x, 2), (y, 1) },

car x apparaît deux fois dans l'expression et y une seule fois.

Afin qu'une expression contenant de multiples caractères génériques identiques soit évaluée correctement, il faut que les occurrences distinctes soient remplacées par des noms de variable

uniques. Cela nécessite les deux fonctions suivantes: Replace qui peut remplacer une occurrence de chaîne de variable ou de caractère générique dans une expression par une autre chaîne de variable, et NewVar qui peut produire une nouvelle chaîne de variable. Cette fonction de substitution prend comme arguments: une chaîne de définition de données <data definition string>, la chaîne de variable à remplacer, un nombre définissant quelle occurrence doit être remplacée, la chaîne de variable de remplacement, et finalement l'expression à transformer. La chaîne de définition de données <data definition string> doit être fournie en tant qu'argument parce que, selon le langage de données, les identificateurs peuvent être surchargés en tant que constantes/variables, etc. et parce que la distinction entre un identificateur de variable et d'autres identificateurs nécessite des informations relatives au contexte. La signature de cette fonction est la suivante:

Replace: <data definition string> →

<variable string> × *Nat* × <variable string> → <expression string> →

<expression string>

Le résultat de l'opération Replace(data_defs)(x, 2, z)(f(x, c, x + y)) pourrait être l'expression f(x, c, z + y), ou f(z, c, x + y), selon la façon dont l'indexation est effectuée. En fournissant une fonction de remplacement pour un langage de données, l'on ne spécifie pas la manière dont les occurrences de variables doivent être indexées car cette fonction est utilisée dans la sémantique dynamique pour remplacer tous les caractères génériques, leur ordre de remplacement n'étant pas déterminant.

La substitution de caractères génériques implique la création de nouvelles chaînes de variable. A cette fin, l'on fournit une fonction NewVar à l'interface. En présence d'un ensemble de chaînes de <variable string> qui est différente de chacune de celles qui font partie de l'ensemble fourni. Comme dans le cas de la fonction Replace, les noms des autres identificateurs définis devront être pris en compte à partir des chaînes de définition de données. La signature de la fonction NewVar est la suivante:

NewVar: <data definition string> → <variable string>-*set* → <variable string>

La fonction Eval sert à évaluer une expression de données. Elle assure l'interface sémantique avec le diagramme MSC. La fonction Eval renvoie la valeur d'une expression de données dans un domaine de données quelconque. Par exemple, l'évaluation d'une expression d'entier produira une valeur d'entier. En général, les expressions renverront des valeurs plus complexes représentant des types structurés tels que des tableaux, etc. Implicitement, les domaines de données font partie de l'interface sémantique et seront représentés par U, qui est un univers de valeurs de données qui comprend des valeurs structurées. La fonction Eval prend comme premier argument une chaîne de définition de données <data definition string> afin de fournir des informations contextuelles au sujet du deuxième argument, qui est la chaîne d'expression à évaluer. L'argument final représente l'information d'état. Il se compose d'associations entre les chaînes de variable et leurs valeurs domaniales. Le résultat de la fonction est une valeur de domaine.

Eval: <data definition string> → <expression string> → (<variable string> × U)-*set* → U

La fonction Eval est partielle et n'est définie que si toutes les variables apparaissant dans l'expression ont des valeurs définies dans l'argument d'état.

5.4 Déclaration de données

La déclaration de données intervient principalement dans un document MSC, la seule exception étant les variables statiques, qui sont déclarées dans un en-tête de diagramme MSC. Les déclarations de document MSC sont les suivantes:

- messages et temporisateurs possédant des paramètres de données;
- variables dynamiques;

- symboles de caractères génériques;
- définitions de données.

Par ailleurs, les parenthèses et les caractères d'échappement des chaînes de données sont également déclarés dans un document MSC. Les messages qui possèdent des paramètres sont déclarés de façon que le type et le nombre de ces paramètres soient définis. Les messages qui n'ont pas de paramètres n'ont pas besoin d'être déclarés. Les variables dynamiques sont déclarées à l'intérieur d'une liste d'instances d'un document MSC car les variables dynamiques appartiennent à des instances. Une déclaration indique le nom des variables et définit leur type. Les variables qui doivent représenter des caractères génériques sont déclarées en même temps que leur type dans le document MSC, de même que les définitions de données. Celles-ci se composent de données textuelles dans le langage de données, par exemple pour définir des types structurés, des constantes et des signatures de fonction. Elles doivent donner toutes les informations requises pour vérifier les types et pour évaluer les expressions de données utilisées dans les diagrammes MSC conformément au domaine de visibilité du document MSC englobant.

Grammaire textuelle concrète

<message decl list> ::=
<message decl> <end> [<message decl list>]

<message decl> ::=
<message name list> [: (<type ref list>)]

<message name list> ::=
<message name> [, <message name list>]

<timer decl list> ::=
<timer decl> <end> [<timer decl list>]

<timer decl> ::=
<timer name list> [<duration>] [: (<type ref list>)]

<timer name list> ::=
<timer name> [, <timer name list>]

<type ref list> ::=
<type ref string> [, <type ref list>]

<dynamic decl list> ::=
variables <variable decl list>

<variable decl list> ::=
<variable decl item> <end> [<variable decl list>]

<variable decl item> ::=
<variable list> : <type ref string>

<variable list> ::=
<variable string> [, <variable list>]

<data definition> ::=
[**language** <data language name> <end>]
[<wildcard decl>]
[**data** <data definition string> <end>]

<wildcard decl> ::=
wildcards <variable decl list>

Grammaire graphique concrète

Les concepts de données sont contenus dans les parties textuelles et aucune grammaire graphique n'est nécessaire.

Exigences statiques

Tous les messages et séquences de temporisation qui possèdent des paramètres doivent être déclarés dans l'en-tête de document <document head>. Le nombre et le type des paramètres d'un message ou d'une séquence de temporisation contenant des données doivent toujours être indiqués lors de la déclaration de ce message. Un type est exprimé dans le langage de données paramétriques qui est défini par la chaîne de référence de type <type ref string>. Toutes les références de type utilisées dans une déclaration de variable, de caractère générique, de séquence de temporisation et de message doivent être correctement définies dans le contexte des chaînes de définition de données <data definition string> conformément aux règles du langage de données, c'est-à-dire que les références de type doivent toujours être correctes, conformément à la fonction Wf3 pour la syntaxe et à la fonction Tc2 pour le type. Toutes les déclarations de variable et de caractère générique doivent être correctes conformément à la fonction Wf1 et tous leurs noms doivent être univoques tels qu'évalués par la fonction EqVar. La chaîne de définition de données <data definition string> doit toujours contenir toutes les informations à utiliser pour effectuer les contrôles d'exigences statiques et de sémantique dynamique requis par la présente Recommandation. Bien que cette chaîne puisse former des fragments incorrects dans le langage de données prévu, ces fragments doivent toujours représenter une abstraction formalisée de ce langage et doivent avoir une syntaxe et une sémantique (statique) correctes, c'est-à-dire qu'ils doivent toujours être corrects, conformément à la fonction Wf2 pour la syntaxe et à la fonction Tc1 pour le type. Par exemple, il serait logique de n'indiquer que des en-têtes de procédure définissant la signature d'une fonction et non pas son en-tête/contenu complet. Mais une telle abstraction serait à formaliser et devrait être vérifiable au moyen des fonctions d'interface.

Sémantique

Un caractère générique est utilisé dans les expressions de données telles qu'une valeur indifférente. Lors de la définition de la signification d'un diagramme MSC contenant des données, un caractère générique produira un ensemble de traces concrètes correspondant à chaque trace non interprétée, où chaque trace concrète est issue de la trace non interprétée par substitution d'une valeur concrète différente pour le caractère générique. Si une expression contient de multiples occurrences d'un caractère générique, chacune de ces occurrences représente une référence différente, de sorte que des valeurs concrètes différentes seront, en général, substituées à chaque occurrence.

5.5 Données statiques

En option, un diagramme MSC peut définir une liste de paramètres formels. Une référence MSC correspondante doit définir une liste de paramètres réels. Une liste de paramètres MSC déclare une liste de variables paramétriques formelles dont le domaine de visibilité est le corps du diagramme MSC. Lorsqu'un paramètre apparaît dans ce corps, il ne doit être utilisé que dans les expressions qui font référence à sa valeur. Il ne peut donc pas être modifié dynamiquement.

Grammaire textuelle concrète

```
<data parameter decl> ::=
                        <variable decl list>

<actual data parameters> ::=
                        <actual data parameter list>

<actual data parameter list> ::=
                        <expression string> [ , <actual data parameter list> ]
```

Exigences statiques

Le symbole <data parameter decl> déclare les paramètres formels d'un diagramme MSC. Chaque paramètre formel contenu dans la déclaration doit être unique dans celle-ci ainsi que différent de toutes les variables dynamiques et de tous les caractères génériques déclarés dans le document MSC englobant. Chaque variable doit être correctement formée selon la fonction Wf1. Sa référence de type doit être bien formée et son type doit être correct dans le contexte de la chaîne de données du

document MSC englobant, conformément aux fonctions Wf3 et Tc2 respectivement. Le nombre et le type des paramètres doivent toujours être observés dans une référence MSC par l'intermédiaire de sa liste de paramètres de données réels <actual data parameter list>. La conformité du type est vérifiée par la fonction Tc4 selon le contexte de la chaîne de définition de données MSC et la déclaration des paramètres formels. Chaque paramètre réel est une expression qui peut contenir des variables et des caractères génériques. Toutes les variables apparaissant dans la liste de paramètres de données réels <actual data parameter list> doivent être statiques et déclarées dans le document MSC englobant. Les variables dynamiques sont interdites.

Sémantique

La signification d'une référence MSC visant des paramètres réels est "appel par valeur", où les paramètres formels sont remplacés par les paramètres réels chaque fois qu'ils apparaissent dans le corps. Si cependant des caractères génériques figurent dans la liste de paramètres de données réels <actual data parameter list>, chacun de ces caractères représente une référence distincte et unique qui sera sémantiquement reconnue dans le corps du diagramme MSC référencé en étant remplacée par de nouveaux noms de variable distincte, chaque variable pouvant prendre une valeur quelconque dans son domaine. L'évaluation des paramètres réels est effectuée par la fonction Eval qui extrait d'abord toutes les variables/caractères génériques de l'expression au moyen de la fonction Vars puis qui utilise les déclarations de caractère générique du document MSC pour déterminer celles qui sont des caractères génériques d'expression. Finalement, les caractères génériques sont remplacées, au moyen de la fonction Subst, par de nouvelles variables produites par la fonction NewVar. Ces nouvelles variables peuvent prendre une valeur quelconque dans leur domaine de données.

5.6 Données dynamiques

Les données dynamiques se rapportent aux variables MSC qui peuvent être des valeurs assignées et réassignées au moyen de cadres d'action, de paramètres de message et de temporisation, et de créations d'instance. La valeur qu'une variable dynamique peut posséder à un point quelconque d'une trace d'exécution dépendra en général des événements précédents de cette trace. La résidence d'une variable dynamique est une instance et sa déclaration figure dans le document MSC. Seuls des événements relatifs à l'instance englobante d'une variable dynamique sont autorisés à modifier la valeur de celle-ci, bien que d'autres instances puissent y faire référence à certaines conditions.

Le mécanisme d'attribution ou de modification de la valeur d'une variable dynamique est celui d'une association, qui se compose d'une structure et d'une expression. Les associations apparaissent à la suite de la transmission de messages, du déclenchement ou de l'arrêt de la temporisation, ou de la création d'instances au moyen des listes de paramètres correspondants, ou sous forme de cadres d'action. La sémantique dynamique définit un ensemble d'associations actuelles pour chaque événement d'une trace d'exécution, cet ensemble étant considéré comme l'état de cet événement. L'utilisation de caractères génériques dans des expressions se traduit par une sous-spécification dans les diagrammes MSC. Chaque caractère générique est autorisée à englober toutes les valeurs de son type de domaine.

Dans un diagramme définisseur, il ne doit pas y avoir de trace d'exécution MSC dans laquelle une variable serait référencée sans avoir été définie. En d'autres termes, chaque variable apparaissant dans une expression doit être liée à l'état utilisé pour calculer la valeur de cette expression. Dans un diagramme MSC utilitaire, les références à des variables indéfinies sont autorisées.

Un état associé à un événement actuel est calculé à partir des états précédents ainsi que des données contenues dans cet événement. Les états précédents qui sont utilisés pour calculer le nouvel état dépendent du type d'événement. Ils dérivent tous au moins du dernier événement non créateur qui a été exécuté sur la même instance que l'événement courant. Par ailleurs, pour les événements de réception de message et pour le premier événement d'une instance créée, l'état des événements d'envoi ou de création correspondants est également utilisé pour le calcul. En fait, cela signifie qu'un état est maintenu par chaque instance et qu'un nouvel état est déduit de l'état précédent de cette

instance ainsi que les informations d'état transmises à l'instance par des messages, ou est déduit de l'instance créatrice dans le cas d'une création d'instance. Etant donné que des informations peuvent circuler entre des instances par transmission de messages et par création d'instances, l'état associé à chaque événement peut contenir des associations avec des variables non possédées par l'instance dans laquelle l'événement se produit. Les règles régissant l'accès à la valeur des variables possédées par des instances étrangères sont définies comme suit.

Si un événement d'envoi ou de création possède une association avec une variable x dans son état associé et si cette variable x apparaît dans l'une de ses expressions paramétriques et que la variable x ne soit pas possédée par l'instance réceptrice ou créée, cette association doit être ajoutée à l'état résultant de l'événement récepteur ou doit remplacer une association pouvant exister dans l'état précédent. En d'autres termes, si la valeur de x est enregistrée dans l'état de l'instance d'envoi/de création, cette association est implicitement héritée par l'état de l'instance réceptrice/créée, à condition que la variable x ne soit pas possédée par l'instance réceptrice/créée.

Si la variable x est possédée par l'instance réceptrice, l'association définie dans l'événement d'envoi ne peut être héritée que si x n'est pas lié dans l'état de l'instance réceptrice et n'est pas lié dans la liste de paramètres. En d'autres termes, l'événement ne peut être hérité que si x n'est pas défini par l'instance réceptrice ou par les paramètres du message.

Si x est possédé par l'instance réceptrice/créée et est lié dans les paramètres de message, le nouvel état prend cette association. En d'autres termes, x devient lié à la valeur définie par la partie expression de l'association paramétrique. Tel est le mode normal d'association paramétrique.

Si x est possédé par l'instance réceptrice et est lié dans l'état précédent de cette instance mais n'est pas lié dans la liste de paramètres, cette précédente association est conservée dans le nouvel état. En d'autres termes, l'association de l'événement expéditeur n'est pas héritée. Cela reflète l'hypothèse qu'une variable, une fois définie dans son instance, ne peut changer de valeur qu'explicitement. Toutes les références à cette variable doivent prendre la dernière valeur explicitement définie.

En résumé, si x n'est lié ni dans l'état précédent ni dans la liste de paramètres, l'association issue de l'événement expéditeur peut être héritée. Intuitivement, l'association d'une variable ne peut être héritée d'une autre instance que si la variable est expédiée à cette instance par insertion dans une expression de paramètre. Une association ne peut cependant pas être héritée si la variable est possédée et visible par l'instance réceptrice car l'association locale a priorité. La valeur d'une variable peut donc être transmise par une chaîne de messages vers d'autres instances, à condition que chaque message fasse explicitement référence à cette variable dans sa liste de paramètres.

5.7 Associations

Les associations sont plus générales que les simples assignations trouvées dans les langages de programmation en raison de l'utilisation de caractères génériques qui permettent une sous-spécification. Une association se compose d'une partie expression et d'une partie structure, associées par un symbole d'association. Ce symbole possède une forme gauche et une forme droite, qui sont toutes les deux équivalentes mais qui permettent une lecture plus naturelle d'une association associée à un message ou à une séquence de temporisation. Les caractères génériques peuvent être utilisés dans une association chaque fois qu'une variable pourrait être utilisée. Mais ils ont une interprétation différente des variables car ils peuvent prendre toutes les valeurs admissibles au lieu d'une seule valeur assignée.

La partie "structure" d'une association se compose soit d'un caractère générique soit d'une variable dynamique: la partie expression est formée de données, qui peuvent contenir des caractères génériques, des variables dynamiques et des variables statiques. En langage MSC, les caractères génériques sont définis par l'utilisateur mais on utilisera le caractère souligné "_" comme caractère générique dans les exemples suivants. L'exemple ci-dessous montre des associations gauche et droite équivalentes, exemptes de caractères génériques:

x := y + 3, y + 3 =: x

Les associations précédentes peuvent être lues comme une attribution ou comme une association à la variable x de la valeur de l'expression 'y + 3'. Si un caractère générique est utilisé à la place de x, comme dans l'exemple suivant, aucune assignation n'est faite et la grammaire des paramètres du message permet une notation abrégée équivalente d'une simple utilisation de l'expression, également représentée. Cette notation abrégée est utile lorsque la valeur d'envoi d'un paramètre est spécifiée mais que l'instance réceptrice n'associe cette valeur à aucune de ses variables dynamiques.

_ := y + 3, y + 3

Les caractères génériques sont utilisés dans les expressions pour représenter des valeurs indifférentes. Si la sémantique dynamique prescrit une unique trace non interprétée dans un diagramme MSC qui contient un caractère générique, n'importe quelle valeur peut être substituée à ce caractère générique pour donner une trace concrète correcte. Par ailleurs, s'il existe de multiples occurrences de caractères génériques (même si elles utilisent le même caractère générique), chacune de ces occurrences est indépendante et peut donc être remplacée par différentes valeurs. Supposons que, dans l'expression _+3, le caractère générique '_' soit de type nombre naturel (c'est-à-dire un entier non négatif), on pourra sémantiquement la remplacer par l'une quelconque des valeurs 0, 1, 2, … . Les valeurs possibles de l'expression sont donc 3, 4, 5 … . Dans l'expression _ + _, les deux occurrences du symbole de caractère générique sont indépendantes et la première comme la seconde occurrence peuvent avoir une étendue quelconque dans le domaine des nombres naturels. Par exemple, le premier caractère générique peut prendre la valeur 1 et le second la valeur 3 pour donner le résultat 4. Si les mêmes valeurs sont requises, une variable peut être liée à un caractère générique à l'intérieur d'un cadre d'action et cette variable peut être utilisée dans l'expression à la place des caractères génériques. Etant donné l'association initiale de x avec un caractère générique, l'expression x + x ne peut être évaluée que par un entier naturel pair.

Grammaire textuelle concrète

<binding> ::=
    <left binding> | <right binding>

<left binding> ::=
    <pattern> <left bind symbol> <expression>

<left bind symbol> ::=
    :=

<right binding> ::=
    <expression> <right bind symbol> <pattern>

<right bind symbol> ::=
    =:

<expression> ::=
    <expression string>

<pattern> ::=
    <variable string> | <wildcard>

<wildcard> ::=
    <wildcard string>

Exigences statiques

La partie expression d'une association doit toujours être conforme aux règles des exigences statiques du langage de données compte tenu du contexte de la chaîne de définition de données <data definition string> définie dans l'en-tête de document <document head> conformément aux fonctions Wf3 et Tc3. Toutes les variables apparaissant dans une expression doivent être déclarées comme des variables statiques, des variables dynamiques ou des caractères génériques; tous les autres identificateurs doivent soit être définis au moyen de la chaîne de définition de données <data definition string> soit être prédéfinis dans le langage de données. Toutes les variables de

structure doivent être déclarées comme étant des variables dynamiques possédées dans l'instance correcte, celle-ci étant déterminée par l'événement dans lequel l'association apparaît.

Le type de la structure doit correspondre à celui de l'expression dans une association. L'équivalence des types peut être vérifiée par application de la fonction Tc4 à la structure comme à l'expression.

Sémantique

Une association ou une liste d'associations est évaluée dans le contexte d'un état et donne naissance à un nouvel état. Un état se compose d'un ensemble d'associations entre des variables et leurs valeurs. Pour évaluer une association, son expression est d'abord évaluée au moyen de l'état actuel, dans lequel les valeurs des variables d'expression sont extraites de son association dans cet état. S'il n'y a pas d'association de variable d'expression dans l'état actuel, il y a référence illégale à une variable indéfinie dans l'expression. La valeur résultante, calculée d'après l'expression, est liée à la variable de structure et la nouvelle association résultante est ajoutée à l'état actuel pour former le nouvel état. Si la variable de structure est déjà liée dans l'état actuel, elle est remplacée dans le nouvel état par la nouvelle association.

Si la partie structure d'une association est un caractère générique, cette association ne modifie pas l'état. Si la partie expression d'une association contient des caractères génériques, chacune de celles-ci est remplacée par de nouvelles variables uniques qui sont autorisées à prendre une valeur quelconque dans leur domaine. L'évaluation d'une expression est effectuée par la fonction Eval une fois que les caractères génériques ont été identifiés et remplacés par de nouvelles variables uniques. Ce remplacement est effectué d'abord par extraction de toutes les variables/caractères génériques de l'expression au moyen de la fonction Vars puis par utilisation des déclarations de caractère générique du document MSC pour identifier les caractères génériques de l'expression. Finalement, les caractères génériques sont remplacés au moyen de la fonction Replace par de nouvelles variables produites par la fonction NewVar. Ces variables peuvent prendre une valeur quelconque dans la sémantique de trace d'un diagramme MSC.

5.8 Données dans les paramètres de message et de temporisation

Les données dynamiques participent aux messages et aux séquences de temporisation au moyen de leurs listes de paramètres. Les paramètres énumérés dans une telle liste peuvent être des associations, des expressions ou des structures. Les exigences statiques déterminent laquelle de ces trois options peut ou doit être utilisée. Pour les messages complets dans un diagramme MSC, une association ou une expression peut être utilisée mais non une structure. Dans ce contexte, une expression est une notation abrégée d'une association avec un caractère générique. Des règles plus complexes s'appliquent aux messages incomplets.

Les messages incomplets apparaissent lorsqu'un message a son origine ou sa terminaison à une porte ou est perdu ou trouvé. Si un message a sa terminaison ou son origine à une porte, des associations ne peuvent pas être explicitement définies dans sa liste de paramètres puisque celles-ci n'ont de raison d'être que lorsque les instances de source comme de destination sont connues. Les associations seront déduites dynamiquement dans la sémantique par appariement de structures et d'expressions figurant dans les deux messages reconnus égaux de part et d'autre de la porte. Pour un message qui est envoyé par une instance et qui aboutit à une porte, seule l'expression peut être donnée. Pour un message qui est extrait d'une porte et envoyé à une instance, seule la structure peut être donnée. Des restrictions similaires sont imposées aux messages perdus et trouvés. Dans l'exemple de la Figure 9, la sémantique dynamique fera correspondre les *demandes* de message de part et d'autre de la porte *g* et en déduira la association $y + 3 =: x$. Si un message a à la fois son origine et sa terminaison à une porte, aucune information paramétrique n'est autorisée.

L'ensemble des associations définies ou formées par les paramètres d'un message se traduit par un changement d'état dès l'événement de réception. Les associations sont évaluées en parallèle et sont utilisées pour mettre à jour l'état précédent. Comme les associations sont concurrentes, l'ensemble

des variables apparaissant comme des variables de structure dans les associations doit être distinct pour éviter que deux associations tentent de relier deux valeurs à une même variable. Par ailleurs, les variables de structure doivent toutes être possédées par l'instance réceptrice car celle-ci n'est autorisée à modifier que ses propres variables et seulement les siennes.

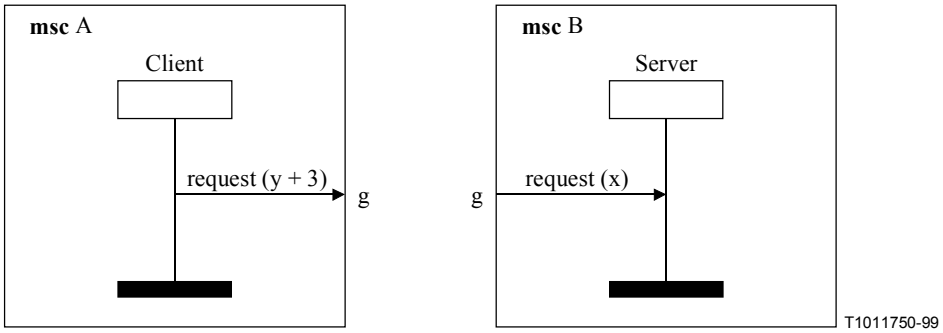


Figure 9/Z.120 – Messages, expressions, structures et portes

Grammaire textuelle concrète

<parameter list> ::=
<parameter defn> [, <parameter list>]

<parameter defn> ::=
<binding> | <expression> | <pattern>

Exigences statiques

Le nombre et le type des paramètres de message ou de temporisation doivent être conformes aux types définis dans la déclaration de message ou de temporisation. Le type d'une association est déterminé soit d'après sa structure soit d'après son expression, qui doivent en effet lui correspondre. La conformité du type est vérifiée au moyen de la fonction Tc4. Dans une liste de paramètres, l'ensemble des variables de structure doit être unique et doit toujours être possédé par l'instance réceptrice du message. Etant donné que les associations d'une liste de paramètres de message sont évaluées en parallèle, toute ambiguïté est ainsi évitée.

Seule une association ou une expression peut être donnée comme paramètre de message dans un message complet, alors qu'une structure ne le peut pas. Seule une expression peut être donnée pour des événements d'envoi incomplet, alors qu'une association ou une structure ne le peut pas. Seules des structures peuvent être indiquées pour des événements de réception incomplète, alors que des associations ou des expressions ne le peuvent pas.

Sémantique

Les paramètres de message ont pour rôle de mettre à jour les états dans les événements de réception de message. Les événements de sortie ne changent pas d'état, celui-ci étant simplement hérité du dernier événements non créateur dans leur instance.

Pour les messages complets, toutes les associations contenues dans la liste de paramètres de message de l'événement de réception sont évaluées au moyen de l'état antérieur et les associations résultantes sont utilisées pour mettre à jour cet état antérieur afin de former le nouvel état. En d'autres termes, une nouvelle association est ajoutée à l'état antérieur ou, si une variable déjà liée dans l'état antérieur est liée de nouveau dans la liste de paramètres, la plus récente association remplace l'ancienne dans le nouvel état. Les variables qui apparaissent dans les parties d'expression d'une liste de paramètres contribuent également à la mise à jour de l'état antérieur. Les associations de ces variables référencées sont extraites de l'état de l'événement d'envoi et sont également utilisées pour mettre à

jour l'état antérieur, sauf pour les variables qui sont possédées par l'instance réceptrice. Dans ce dernier cas, l'association n'est ajoutée que si la variable n'était pas liée dans l'état antérieur et n'est pas liée par la liste de paramètres.

Pour les événements de réception de message incomplets, les associations sont d'abord créées dynamiquement par appariement, avec les structures de l'événement de réception, des expressions paramétriques correspondantes à partir de l'événement d'envoi correspondant. Les associations obtenues sont ensuite évaluées comme pour les messages complets et l'état est modifié de la même façon.

5.9 Données dans les paramètres de création d'instance

Les données dynamiques contenues dans les événements de création d'instance sont traitées comme les paramètres de message. Cependant, comme il n'y a pas d'événement créé correspondant à l'événement créateur, l'état est modifié par l'événement de création conformément aux éventuelles associations figurant dans la liste de paramètres. Mais cet état modifié n'est utilisé que pour évaluer l'événement suivant affectant l'instance créée et non pas l'instance créatrice. L'état existant avant l'événement de création est utilisé pour évaluer l'événement suivant de l'instance créatrice. Toutes les variables utilisées comme variables de structure dans une liste de paramètres de création doivent être la propriété de l'instance créée.

5.10 Données dans les cadres d'action

Les données peuvent apparaître dans les cadres d'action sous la forme d'une liste de déclarations séparées par des virgules. Une déclaration est soit une déclaration de définition, soit une déclaration de non-définition ou une association. Un énoncé de définition sert à indiquer qu'une valeur non spécifiée a été affectée à une variable; c'est l'équivalent de l'association d'une variable à un caractère générique. En d'autres termes, "**def** x" est l'équivalent de "x := _" où le caractère "_" est un caractère générique. Une déclaration de non-définition sert à indiquer qu'une variable n'est plus liée, c'est-à-dire qu'elle ne peut plus être légalement référencée ou est sortie du domaine de visibilité. Dans un même cadre d'action, les déclarations sont évaluées en parallèle et non en séquence, en conséquence de quoi les règles des exigences statiques interdisent toute ambiguïté entre différentes déclarations tentant de lier différentes valeurs à la même variable. La mise en séquence peut être réalisée par mise en séquence des cadres d'action.

Grammaire textuelle concrète

<data statement list> ::=
<data statement> [, <data statement list>]

<data statement> ::=
<define statement> | <undefine statement> | <binding>

<define statement> ::=
def <variable identifier>

<undefine statement> ::=
undef <variable identifier>

Exigences statiques

Toutes les variables apparaissant dans une déclaration de définition, dans une déclaration de non-définition ou dans la partie structure d'une association doivent être des variables dynamiques possédées par l'instance dans laquelle leur cadre d'action englobant apparaît. Elles doivent également être distinctes, ce qui est nécessaire parce que les déclarations d'un cadre d'action sont évaluées en parallèle.

Sémantique

Chaque déclaration contenue dans un cadre d'action est évaluée au moyen de l'état du précédent événement non créateur de la même instance. L'état résultant est déduit du précédent par mise à jour des associations effectuées ou supprimées par chacune des déclarations. En application des exigences statiques, il ne peut pas y avoir d'ambiguïté lors de la formation de l'état résultant. Pour les associations, les variables de structure se lient dans le nouvel état aux valeurs de leurs expressions. Une déclaration de définition ajoute ou remplace une association à partir de l'état précédent, la variable de déclaration de la nouvelle association étant liée à un caractère générique.

5.11 Types de données supposés

Dans la présente Recommandation, il y a trois expressions où le langage MSC suppose l'existence de types de données. Il s'agit des suivantes:

- les expressions évaluées en opérateurs booléens et utilisées dans les conditions de garde (voir 4.7);
- les expressions évaluées en nombres naturels et utilisées pour définir des limites de boucle (voir 7.2);
- les expressions temporelles utilisées pour spécifier des contraintes de temporisation (voir paragraphe 6).

Conformément à la méthode de traitement des données utilisée dans la présente Recommandation, ces types doivent être définis dans le cadre du langage de données choisi par l'utilisateur et non dans celui du langage MSC. Ces types sont cependant nécessaires pour avoir les interprétations spécifiques car la sémantique MSC est définie dans les termes de ces interprétations. Le type des expressions utilisées dans les conditions de garde doit donc avoir un domaine booléen normal composé seulement d'un élément "*true*" et d'un élément "*false*".

Pour les limites de boucle, le domaine du type doit être celui des nombres naturels, muni de l'opération de soustraction habituelle (nécessaire pour calculer la différence entre limite supérieure et limite inférieure d'une boucle). Cette condition peut être allégée pour permettre que le domaine soit un sous-ensemble des nombres naturels car la plupart des langages (de programmation) n'ont que des représentations finies. Cela n'affecte pas la sémantique des boucles car seules ces valeurs limites seront bornées par le sous-ensemble autorisé. Les boucles infinies restent en particulier exprimables au moyen du mot clé **inf**.

Les prescriptions relatives au type temporel sont plus abstraites et sont définies comme suit:

- le domaine doit être un ordre total ayant comme plus petit élément ou origine le temps zéro;
- le domaine doit être fermé après une opération d'addition utilisée pour calculer les décalages temporels.

Il existe des exigences minimales qui permettent de définir la sémantique des diagrammes MSC temporisés. En pratique, un utilisateur aura un type plus riche, prenant en charge des opérations temporelles supplémentaires, déclarées dans la partie définition de données du document MSC. Comme dans le cas des nombres naturels, le domaine réel peut être autorisé à être un sous-ensemble du domaine idéal requis pour permettre d'apporter des limitations aux langages de données utilisés. La prescription d'ordre total reflète le modèle temporel linéaire utilisé dans les diagrammes MSC, par opposition au modèle temporel à embranchements conditionnels, par exemple. La fermeture après une opération d'addition signifie que le temps ne peut jamais être épuisé.

6 Concepts temporels

Les concepts temporels sont introduits dans les diagrammes MSC pour prendre en charge la notion de temps quantifié pour la description des systèmes en temps réel avec une interprétation précise de

la séquence temporelle des événements. Les événements MSC sont instantanés. Des contraintes temporelles peuvent être spécifiées afin de définir le moment où des événements peuvent se produire.

Chaque diagramme MSC contient des instances auxquelles des événements sont associés. Un diagramme MSC classique ne tenant pas compte du temps peut être interprété comme un ensemble de traces d'événements. Dans cette interprétation non temporisée, l'écoulement du temps est représenté explicitement de façon quantifiée, c'est-à-dire que les traces des événements sont enrichies par un événement spécial qui représente l'écoulement du temps. L'interprétation non temporisée d'un diagramme MSC est un surensemble de l'interprétation temporisée d'un MSC, réduite aux caractéristiques non temporisées.

La temporisation d'un diagramme MSC en enrichit les traces par des valeurs quantitatives de temps qui représentent la distance temporelle entre paires d'événements. L'écoulement du temps (c'est-à-dire le pointage temporel) est égal pour toutes les instances contenues dans un diagramme MSC. De même, toutes les valeurs de pointage temporel sont égales, c'est-à-dire que l'on suppose une horloge globale. Tous les événements sont instantanés, c'est-à-dire atomiques, et ne consomment pas de temps.

6.1 Sémantique temporisée

La sémantique temporisée d'un diagramme MSC peut être représentée par des traces avec des événements temporels spéciaux tels que:

{e1, e2, t3, e4, t5, e6, e7, e8, … }

Le triplet (e4, t5, e6) signifie par exemple qu'après l'occurrence de l'événement e4, le temps t4 s'écoule jusqu'à ce que l'événement e6 se produise. Les événements qui ne sont pas séparés par des événements temporels intermédiaires (comme e1 et e2) se produisent simultanément, c'est-à-dire sans aucun retard. L'on part du principe que le temps est progressif et non statique. Sa progression est interprétée par le fait qu'après chaque événement d'une trace, il y a finalement un événement temporel. Sa non-staticité est interprétée par le fait qu'il existe une limite supérieure au nombre des "événements normaux" situés entre un événement temporisé et l'événement temporisé suivant.

La trace ci-dessus est égale à la trace ci-dessous (la durée zéro étant exprimée explicitement):

{e1, 0, e2, t3, e4, t5, e6, 0, e7, 0, e8, … }

La sémantique non temporisée d'un diagramme MSC contenant les traces:

{e1, e2, e3, … }

correspond à un ensemble de traces ayant une durée arbitraire entre événements, soit:

{e1, durée quelconque, e2, durée quelconque, e3, durée quelconque, …}

Par ailleurs, la réduction non temporisée d'une trace temporisée:

{e1, t1, e2, t2, e3, t3, e4, t4, e5, t5, … }

est:

{e1, e2, e3, … }

Le fait que, dans la sémantique temporisée, une trace de diagramme MSC commence par un événement "normal" (c'est-à-dire qu'un événement précédent existe pour chaque événement sauf pour le premier) est intentionnel. La temporisation peut être définie par rapport à l'événement précédent.

En général, il n'existe pas d'événement unique de démarrage pour une instance de diagramme MSC ou pour un diagramme MSC dans son ensemble. Un diagramme MSC définit un ensemble de traces pouvant avoir des événements de démarrage différents.

6.2 Temporisation relative

La temporisation relative utilise des paires d'événements (précédents et subséquents), dans lesquelles l'événement précédent active (directement ou indirectement c'est-à-dire en passant par certains événements intermédiaires) l'événement subséquent. Au sujet de l'utilisation de la temporisation relative, voir 6.10 ci-dessous, concernant les intervalles temporels.

La temporisation relative peut être spécifiée au moyen d'expressions arbitraires de type Temps, c'est-à-dire par référenciation à des paramètres, à des caractères génériques et à des variables dynamiques. La valeur concrète d'une expression de temporisation relative est évaluée une fois que le nouvel état de l'événement relatif à cette temporisation relative a été évalué. Voir 5.6 concernant la notion de nouvel état.

6.3 Temporisation absolue

La temporisation absolue est utilisée pour définir l'occurrence d'événements à des points temporels qui se rapportent à la valeur de l'horloge globale.

La temporisation absolue peut être spécifiée au moyen d'expressions arbitraires de type Temps, c'est-à-dire par référenciation à des paramètres, à des caractères génériques et à des variables dynamiques. Les valeurs concrètes d'une contrainte temporelle sont évaluées au début d'un intervalle de temps une fois que le nouvel état de l'événement relatif au début de cet intervalle de temps a été évalué. Voir 5.6 concernant la notion de nouvel état.

6.4 Domaine temporel

Le domaine temporel peut être dense ou discret. Il doit être en ordre total avec au moins un élément, ou l'origine, de l'instant zéro. Il doit être fermé par une opération d'addition, utilisée pour calculer les décalages temporels.

Exemples de domaine temporel: nombres rationnels non négatifs enrichis d'une unité temporelle qui est soit h (indication des heures), min (indication des minutes), s (indication des secondes), ms (indication des millisecondes), μs (indication des microsecondes) et ns (indication des nanosecondes.

Exigences statiques

Le domaine temporel doit être en ordre total avec au moins un élément, ou l'origine, de l'instant zéro. Il doit être fermé par une opération d'addition, utilisée pour calculer les décalages temporels.

6.5 Variables temporelles statiques et dynamiques

Un diagramme MSC peut utiliser des variables temporelles statiques ou dynamiques. Ces variables sont analogues à toute autre variable, sauf qu'elles sont du type Temps dans le langage MSC.

Exigences statiques

Les variables temporelles statiques et dynamiques sont conformes aux exigences d'autres variables statiques et dynamiques comme décrit dans le paragraphe 5 concernant les concepts de données.

6.6 Décalage temporel

On peut attribuer un décalage temporel à un diagramme MSC afin de décaler toutes les valeurs temporelles absolues contenues dans ce diagramme MSC. Le décalage temporel est défini par une expression du domaine temporel du MSC. Dans un MSC sans décalage temporel explicite, on suppose une valeur par défaut 0 du décalage temporel.

Grammaire textuelle concrète

<time offset> ::=
offset <time expression>

Grammaire graphique concrète

Non nécessaire.

Exigences statiques

Dans la déclaration de décalage, seule une expression de type Temps doit être donnée. Cette expression ne doit faire référence qu'à des paramètres MSC déclarés.

6.7 Points, mesures et intervalles temporels

Les contraintes temporelles peuvent être définies sous forme de points temporels (c'est-à-dire de valeurs temporelles) ou d'intervalles temporels (c'est-à-dire d'étendues de valeurs temporelles entre des limites données). Les observations chronologiques sont décrites par des mesures.

6.8 Points temporels

Grammaire textuelle concrète

<time point> ::=
[<abs time mark>] <time expression>

Les points temporels sont définis par des expressions de type Temps. La marque facultative de temps absolue indique une temporisation absolue.

Grammaire graphique concrète

Non nécessaire.

Exigences statiques

L'évaluation d'un point temporel se traduit par un temps quantifié concret. Un événement sans contraintes temporelles peut se produire à un moment quelconque.

6.9 Mesures

Les mesures servent à observer la durée qui s'écoule entre l'activation et l'occurrence d'un événement (en temporisation relative) et à mesurer le temps absolu de l'occurrence d'un événement (en temporisation absolue). Afin de distinguer une mesure relative d'une mesure absolue, différentes marques de temps (c'est-à-dire '&' pour absolu et '?' pour relatif) sont utilisées.

Les mesures peuvent être associées à des intervalles de temps. Pour chaque mesure, une variable temporelle doit être déclarée pour l'instance considérée.

Grammaire textuelle concrète

<measurement> ::=
<rel measurement>
| <abs measurement>

<rel measurement>::=
<rel time mark> <time pattern>

<abs measurement>::=
<abs time mark> <time pattern>

Grammaire graphique concrète

Non nécessaire.

Exigences statiques

Les mesures ne doivent utiliser que des structures de type Temps, qui font référence à des variables dynamiques déclarées dans le document MSC englobant pour l'instance considérée.

Sémantique

L'évaluation de la durée (dans le cas du temps relatif) et de la valeur d'horloge globale (dans le cas du temps absolu) ainsi que l'association avec la variable temporelle font partie de l'évaluation du nouvel état de l'événement auquel se rapporte la mesure.

6.10 Intervalles temporels

Les intervalles temporels servent à définir des contraintes de temporisation pour l'occurrence d'événements: la durée s'écoulant entre une paire d'événements peut être contrainte par la définition d'une valeur limite, minimale ou maximale, de l'intervalle entre ces deux événements.

Un intervalle temporel n'implique pas que les événements doivent se produire. L'application d'une contrainte temporelle n'est validée que si l'événement correspondant à la fin de cet intervalle temporel se produit dans la trace. Une trace MSC doit appliquer toutes ses contraintes temporelles. En d'autres termes, si une trace viole une contrainte temporelle, cette trace est illégale.

Les intervalles temporels peuvent être utilisés pour la temporisation relative comme pour la temporisation absolue. Ils peuvent être spécifiés au moyen d'expressions arbitraires de type Temps, c'est-à-dire par référenciation à des paramètres, à des caractères génériques et à des variables dynamiques. Les valeurs concrètes d'une contrainte temporelle, imposées par un intervalle temporel, sont évaluées au début de celui-ci une fois que le nouvel état de l'événement relatif au début de cet intervalle de temps a été évalué. Voir 5.6 concernant la notion de nouvel état.

L'ordre d'une paire d'événements associés à une contrainte temporelle est déterminé par la sémantique dynamique et non par la contrainte temporelle.

Au moyen des limites d'intervalle, chaque événement peut imposer des contraintes aux intervalles temporels auxquels il est associé. Ces contraintes ne sont cependant prises en compte que si l'événement se rapporte au début d'un intervalle temporel, ce qui est déterminé dynamiquement. Si un événement se rapporte à la fin d'un intervalle temporel, ses contraintes temporelles n'ont pas d'importance. Les représentations textuelle et graphique d'un diagramme MSC n'indiquent que les paires d'événements possibles. Une telle paire d'événements est indiquée soit par la connexion des limites d'intervalle des deux événements, soit par la connexion de limites d'intervalle dissociées au moyen d'une étiquette d'intervalle, ou par des portes de messagerie.

Grammaire textuelle concrète

```
<time interval> ::=
                    [<interval label>] <singular time>
                |   [<interval label>] <bounded time>
                    [ <measurement> ]

<interval label> ::=
                    int_boundary < interval name>

<singular time> ::=
                    <left closed> <time point> <right closed>
                |   <measurement>

<bounded time> ::=
                    [<abs time mark>] { <left open> | <left closed> }
                    [<time point>] , [<time point>]
                    { <right open> | <right closed> }
```

Exigences statiques

A l'intérieur d'un intervalle temporel, l'on ne doit utiliser que des expressions de temps relatif ou que des expressions de temps absolu. La borne minimale, la borne maximale ou les deux bornes sont indiquées. Un intervalle doit toujours définir au moins une des deux bornes.

Un intervalle temporel absolu doit être de la forme [@1,@3) or @[1,3).

Des intervalles temporels peuvent être définis pour deux événements quelconques à l'intérieur d'un document MSC.

Grammaire graphique concrète

<time interval area> ::=
<interval area>
| <abs time area>

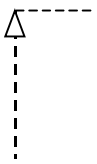
<interval area> ::=
<int symbol>
is associated with <time interval>
is followed by { <cont interval> | <interval area 2>}

<interval area 2> ::=
<int symbol>
[***is associated with*** <time interval>]

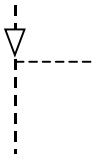
<cont interval> ::=
<cont int symbol>
is associated with <interval name>

<int symbol> ::=
{<int symbol 1> | <int symbol 2> }
is attached to
{ <message out symbol> |<message in symbol> |
<reply symbol> | <action symbol> |
<timer start symbol> | <timer stop symbol> |<timeout symbol> |
<restart symbol> | <createline symbol> |
<inline expression symbol> | <separator symbol>
<msc reference symbol> | <par frame symbol>
<instance head symbol> }

<int symbol 1> ::=



<int symbol 2> ::=



Les symboles d'intervalle temporel <int symbol 1> et <int symbol 2> peuvent être projetés en symétrie horizontalement et verticalement. Le symétrique du symbole <int_symbol 1> ne doit pas être rattaché au symbole <int_symbol 1>.

<cont int symbol> ::=

Le symbole de continuation pour les intervalles temporels <cont int symbol> peut être projeté en symétrie verticalement. Le symétrique du symbole <cont int symbol> ne doit pas être rattaché aux symboles <int symbol 1> et <int symbol 2>, de même que le symbole <cont int symbol> ne doit pas être rattaché au symétrique du symbole <int symbol 1> ou <int symbol 2>.

<abs time area>::=

<abs time symbol>
is associated with { <abs time expr> | <abs measurement>}
is attached to
{ <message out symbol> |<message in symbol> | <action symbol> |
<timer start symbol> | <timer stop symbol> |<timeout symbol> |
<inline expression symbol> | <separator symbol> |
<msc reference symbol> | <par frame symbol> |
<call in symbol> | <call out symbol> | <reply symbol>}

<abs time symbol>::=

<abs time expr>::=

<abs time mark> <time expression>

La représentation graphique d'un intervalle temporel relatif utilise des lignes tiretées avec deux pointes de flèche (vides par exemple) afin d'indiquer l'événement de début et l'événement de fin pour cet intervalle.

De petites variations sont autorisées quant à la forme réelle des pointes de flèche, comme le hachurage ou la forme de ces pointes. Il est cependant recommandé que les pointes de flèche des intervalles temporels puissent être distinguées des pointes de flèche des événements de messagerie ou de création.

La représentation graphique d'un intervalle temporel absolu utilise une seule ligne tiretée pour pointer sur l'événement à temporisation absolue.

Les intervalles temporels peuvent être subdivisés en plusieurs parties qui sont logiquement connectées au moyen d'une étiquette. En cours d'exécution, les intervalles disjoints sont reconstitués afin de déterminer le début et la fin de ces intervalles.

Les lignes horizontales des symboles temporels relatifs et absolus peuvent être représentées par des polylignes arbitraires. Elles peuvent également être élargies et rétrécies. Mais les lignes verticales des symboles temporels relatifs doivent toujours rester verticales. La partie finale d'une polyligne de temps absolu doit toujours être horizontale et être associée à l'expression de temps absolu ou, selon le cas, à la mesure absolue.

Sémantique

Dans chaque trace d'un diagramme MSC, il doit y avoir une correspondance unique entre le début et la fin d'un intervalle temporel divisé (c'est-à-dire que la réunion d'intervalles temporels divisés n'est effectuée qu'au moment de l'exécution). Par exemple, dans une trace, il ne doit pas y avoir deux fins du même intervalle.

7 Concepts structurels

Des concepts structurels de haut niveau sont introduits au 7.1. Les corégions permettent de décrire des zones dans lesquelles les événements peuvent apparaître dans un ordre quelconque. Les expressions en ligne contribuent à structurer les notions de composition en alternative, de composition en parallèle et de boucles. Les références MSC sont utilisées pour faire référence à d'autres diagrammes MSC à partir d'un diagramme MSC. Les diagrammes HMSC sont abstraits des instances et donnent un aperçu général de comportement plus complexes.

Les références MSC et les expressions en ligne peuvent avoir des contraintes temporelles. La temporisation est interprétée par rapport aux événements, déterminés dynamiquement, de début et de fin de la référence MSC ou de l'expression en ligne, selon le cas.

7.1 Corégion

L'ordre total des événements le long de chaque instance (voir 4.2) ne sera sans doute pas approprié aux entités faisant référence à des processus de niveau plus élevé que celui des processus simples.

De ce fait, la notion de corégion est introduite pour spécifier des événements non ordonnés dans une instance. Une telle notion de corégion couvre en particulier le cas, important en pratique, de deux messages entrants (ou plus) pour lesquels l'ordre de consommation peut être interverti. Inversement, lors de la diffusion de messages, l'envoi de deux ou plus de deux messages sortants peut être interverti. Un ordre généralisé peut être défini par des relations d'ordre général.

Grammaire textuelle concrète

<start coregion>::=
concurrent <end>

<end coregion> ::=
endconcurrent <end>

Grammaire graphique concrète

<concurrent area> ::=
<coregion symbol>
is attached to <instance axis symbol>
contains <coevent layer>

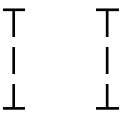
<coregion symbol> ::=
<coregion symbol1> | <coregion symbol2>

<coevent layer> ::=
<coevent area> | <coevent area> ***above*** <coevent layer>

<coevent area> ::=
{ <message event area> | <incomplete message area> | <action area> |
<timer area> | <create area> }*

<coregion symbol1> ::=

<coregion symbol2> ::=



Règle de dessin: la déclaration selon laquelle le symbole de corégion <coregion symbol> est attachée au symbole d'axe d'instance <instance axis symbol> signifie que le symbole de corégion <coregion symbol> doit toujours se superposer au symbole d'axe d'instance <instance axis symbol> comme dans l'exemple montré dans la Figure 10:

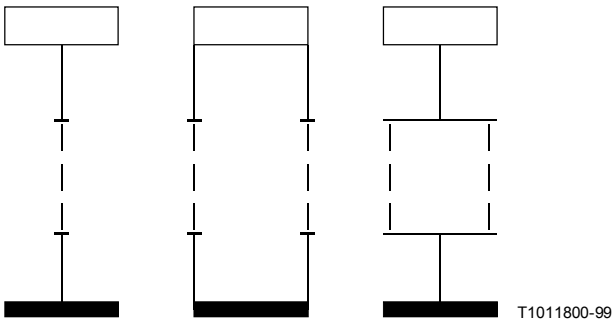


Figure 10/Z.120 – Différentes formes de corégion

Le symbole <coregion symbol1> ne doit jamais être attaché au symbole <instance axis symbol2>.

Exigences statiques

Dans une instance spécifique, il ne doit y avoir que des événements ordonnables <orderable event> entre un début de corégion <start coregion> et une fin de corégion <end coregion>.

Sémantique

Pour des diagrammes MSC, l'on part du principe que les événements contenus dans chaque instance sont en relation d'ordre total. Une corégion permet de faire exception à ce principe: les événements contenus dans la corégion ne sont pas ordonnés si aucune autre construction de synchronisation, sous forme de relation d'ordre général, ne sont prescrites.

Si un temporisateur est armé et que son expiration ou son arrêt correspondant soit contenu dans une corégion, l'on part du principe qu'une relation d'ordre général existe implicitement entre le début et l'expiration/l'arrêt de la temporisation.

7.2 Expression en ligne

Les expressions d'opérateur en ligne permettent de définir la composition de structures d'événements à l'intérieur d'un diagramme MSC. Les opérateurs se rapportent aux régions de composition en alternative, de composition en parallèle, d'itération, d'exception et d'option.

Les expressions en ligne temporisées permettent de contraindre ou de mesurer la durée d'exécution de régions de composition en alternative, de composition en parallèle, d'itération, d'exception et d'option. Des intervalles temporels aussi bien relatifs qu'absolus (avec ou sans mesures) peuvent être utilisés.

Ces intervalles temporels peuvent se rapporter au début et à la fin d'une expression en ligne. La valeur de début indique le moment où le premier événement intervient dynamiquement et la valeur de fin indique le moment où le dernier événement intervient. Les intervalles temporels s'appliquent donc à toutes les instances impliquées dans une expression en ligne.

Grammaire textuelle concrète

<shared inline expr> ::=
[<extra-global>]{ <shared loop expr> | <shared opt expr> |
<shared alt expr> | <shared par expr> | <shared exc expr> }
[**startbefore** <time dest list> <end>]
[**startafter** <time dest list> <end>]
[**endbefore** <time dest list> <end>]
[**endafter** <time dest list> <end>]

<extra-global> ::=
external

<shared loop expr> ::=
loop [<loop boundary>] **begin** [<inline expr identification>]<shared> <end>
[<inline gate interface>] [<instance event list>]
loop end [<time interval>] <end>

<shared opt expr> ::=
opt begin [<inline expr identification>] <shared> <end>
[<inline gate interface>] [<instance event list>]
opt end [<time interval>] <end>

<shared exc expr> ::=
exc begin [<inline expr identification>] <shared> <end>
[<inline gate interface>] [<instance event list>]
exc end [<time interval>] <end>

<shared alt expr> ::=
alt begin [<inline expr identification>] <shared> <end>
[<inline gate interface>] [<instance event list>]
{ **alt** <end> [<inline gate interface>] [<instance event list>] }*
alt end [<time interval>] <end>

<shared par expr> ::=
par begin [<inline expr identification>] <shared> <end>
[<inline gate interface>] [<instance event list>]
{ **par** <end> [<inline gate interface>] [<instance event list>] }*
par end [<time interval>] <end>

<inline expr> ::=
[<extra-global>] {<loop expr> | <opt expr> | <alt expr> |
<par expr> | <exc expr>}

<loop expr> ::=
loop [<loop boundary>] **begin** [<inline expr identification>] <end>
[<inline gate interface>] <msc body>
loop end [<time interval>] <end>

<opt expr> ::=
opt begin [<inline expr identification>] <end>
[<inline gate interface>] <msc body>
opt end [<time interval>] <end>

<exc expr> ::=
exc begin [<inline expr identification>] <end>
[<inline gate interface>] <msc body>
exc end [<time interval>] <end>

<alt expr> ::=
alt begin [<inline expr identification>] <end>
[<inline gate interface>] <msc body>
{ **alt** <end> [<inline gate interface>] <msc body> }*
alt end [<time interval>] <end>

<par expr> ::=
par begin [<inline expr identification>] <end>
[<inline gate interface>] <msc body>
{ **par** <end> [<inline gate interface>] <msc body> }*
par end [<time interval>] <end>

<loop boundary> ::=
<left angular bracket> <inf natural> [, <inf natural>]
<right angular bracket>

<inf natural> ::=
inf | <expression>

<inline expr identification> ::=
<inline expr name>

<inline gate interface> ::=
{ **gate** <inline gate> <end>}+

<inline gate> ::=
<inline out gate> | <inline in gate> |
<inline create out gate> | <inline create in gate> |
<inline out call gate> | <inline in call gate> |
<inline out reply gate> | <inline in reply gate> |
<inline order out gate> | <inline order in gate>

Grammaire graphique concrète

<inline expression area> ::=
{ <loop area> | <opt area> | <par area> | <alt area> | <exc area> }
[***top or bottom is attached to top or bottom***
{ {<int symbol> | <abs time symbol> }* } ***set***]
[***is attached to*** <msc symbol>]
[***is followed by*** <general name area>]

<loop area> ::=
<inline expression symbol> [***is attached to*** <time interval area>] ***contains***
{ **loop** [<loop boundary>] <operand area> }
is attached to { <instance axis symbol>* } ***set***
is attached to { <inline gate area>* | <inline order gate area>* } ***set***

<opt area> ::=
<inline expression symbol> [***is attached to*** <time interval area>] ***contains***
{ **opt** <operand area> }
is attached to { <instance axis symbol>* } ***set***
is attached to { <inline gate area>* | <inline order gate area>* } ***set***

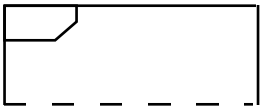
<exc area> ::=
<exc inline expression symbol>
[***is attached to*** <time interval area>] ***contains***
{ **exc** <operand area> }
is attached to { <instance axis symbol>* } ***set***
is attached to { <inline gate area>* | <inline order gate area>* } ***set***

<par area> ::=
<inline expression symbol> [***is attached to*** <time interval area>] ***contains***
{ **par** <operand area>
{ ***is followed by*** <separator area> ***is followed by*** <operand area> }* }
is attached to { <instance axis symbol>* } ***set***
is attached to { <inline gate area>* | <inline order gate area>* } ***set***

<alt area> ::=
<inline expression symbol> [***is attached to*** <time interval area>] ***contains***
{ **alt** <operand area>
{ ***is followed by*** <separator area> ***is followed by*** <operand area> }* }
is attached to { <instance axis symbol> }* ***set***
is attached to {<inline gate area>* | <inline order gate area>* } ***set***

<inline expression symbol> ::=

<exc inline expression symbol> ::=



<operand area> ::=
{ <event layer> | <inline gate area> | <inline order gate area> }* ***set***
[***is followed by*** <general name area>]

<separator area> ::=
<separator symbol>

<separator symbol> ::=
– – – – – – – – – – – –

Exigences statiques

Le symbole de zone d'expression en ligne <inline expression area> peut ne se rapporter qu'à une seule instance ou être rattaché à plusieurs instances. Si une expression en ligne partagée traverse une instance qui n'est pas visée par cette expression en ligne, une option permet de faire passer l'axe de cette instance à travers l'expression en ligne.

Toutes les expressions d'exception doivent être partagées par toutes les instances dans le diagramme MSC.

Les expressions en ligne extraglobales sont celles dont le mot clé **external** est exprimé dans la notation textuelle ou traverse le cadre MSC dans la notation graphique. Cette dernière situation graphique est décrite dans la grammaire comme étant attachée au symbole msc "***attached to*** <msc symbol>". Les expressions extraglobales doivent également couvrir toutes les instances dans le diagramme MSC.

Les expressions de données définissant des limites de boucle peuvent contenir des variables statiques et des caractères génériques; mais elles ne doivent pas contenir de variables dynamiques.

Sémantique

Les mots clés d'opérateur **seq**, **alt**, **par**, **loop**, **opt** et **exc**, qui sont placés dans le coin supérieur gauche de la représentation graphique, indiquent respectivement la composition en alternative, la composition en parallèle, l'itération, la région facultative et l'exception. Dans la forme graphique, un cadre entoure les opérandes et les lignes tiretées indiquent des séparateurs d'opérande.

L'opérateur **seq** représente l'opération de séquencement faible. Un opérande **seq** avec une garde ayant pour valeur *false* est dynamiquement illégal.

L'opérateur **alt** définit des variantes d'exécution de sections MSC, c'est-à-dire que si plusieurs sections MSC sont considérées comme étant des variantes, une seule d'entre elles sera exécutée. Si des sections MSC en variante ont un préambule commun, le choix de la section MSC qui sera exécutée est effectué après l'exécution du préambule commun. Des opérandes en variante, avec une garde dont l'évaluation donne la valeur *false*, ne peuvent pas être choisis. Si tous les opérandes ont des gardes fausses, aucune trace légale ne peut traverser cette expression **alt** qui est, de ce fait, dynamiquement illégale.

Un seul opérande d'une expression **alt** peut être gardé par le mot clé **otherwise**, qui est interprété comme la conjonction des négations des gardes de tous les autres opérandes. La garde de type **otherwise** n'a donc la valeur *true* que si les gardes de tous les autres opérandes de l'expression en alternative ont la valeur *false*. Un opérande sans garde est considéré comme ayant une garde de valeur *true* et la branche indiquée par le mot clé **otherwise** sera impossible à atteindre.

L'opérateur **par** définit l'exécution en parallèle de sections MSC, c'est-à-dire que tous les événements contenus dans les sections MSC en parallèle seront exécutés, mais que la seule restriction est que l'ordre de ces événements dans chaque section sera préservé. Les opérandes en parallèle possédant une garde dont la valeur est *false* sont exclus de la composition en parallèle. Si tous les opérandes ont des gardes de valeur *false*, l'expression **par** prend la valeur **empty** (vide).

La structure **loop** peut avoir plusieurs formes, dont la plus fondamentale est "**loop** <n,m>", où n et m sont des expressions de type nombres naturels. En d'autres termes, l'opérande peut être exécuté au moins n fois et au plus m fois. Ces expressions peuvent être remplacées par le mot clé **inf**, comme dans "**loop** <n,**inf**>", qui signifie que l'itération sera exécutée au moins n fois. Si le deuxième opérande est omis comme dans "**loop** <n>", cette expression est interprétée comme "**loop** <n,n>". L'expression "**loop** <**inf**>" désigne donc une boucle infinie. Si les limites de boucle sont omises

comme dans l'expression "**loop**", celle-ci sera interprétée comme "**loop** <1,**inf**>". Si le premier opérande est plus grand que le second, l'itération sera exécutée 0 fois. Les passes d'une boucle sont connectées au moyen de la composition séquentielle faible.

Lorsque l'opérande d'itération est gardé, l'itération est terminée lorsque la garde prend la valeur *false* et se poursuit lorsque la garde prend la valeur *true* aussi longtemps que la limite supérieure n'est pas atteinte. Une boucle sera donc égale à un diagramme MSC vide si la garde est fausse lorsque l'itération est commencée pour la première fois. Si la limite inférieure de la boucle n'est pas atteinte à cause de la garde, l'ensemble de l'itération est interprété comme dynamiquement illégal. La limite supérieure représente une borne maximale du nombre d'itérations de la boucle.

L'opérateur **opt** est le même que celui d'une variante dont le deuxième opérande est le diagramme MSC vide. Une expression **opt** gardée passera toujours par l'opérateur d'option si la garde prend la valeur *true*.

L'opérateur **exc** est un moyen compact de décrire des cas exceptionnels dans un diagramme MSC. La signification de cet opérateur est que soit les événements situés à l'intérieur du symbole <exc inline expression symbol> sont exécutés et qu'alors le diagramme MSC est achevé, soit que les événements situés après le symbole <exc inline expression symbol> sont exécutés. L'opérateur **exc** peut donc être considéré comme une variante dans laquelle le deuxième opérande est le reste entier du diagramme MSC. Toutes les expressions d'exception doivent être partagées par toutes les instances dans le diagramme MSC. L'expression d'exception est l'abrégé d'une expression en alternative dans laquelle le reste du cadre englobant est le deuxième opérande.

Dans la représentation textuelle, l'interface facultative <inline gate interface> définit les messages qui pénétrant ou quittant l'expression en ligne au moyen de portes. Le symbole <inline gate interface> définit également la connexion directe de messages entre deux expressions en ligne, par identification des noms de message et identification facultative des noms de porte.

Les expressions en ligne extraglobales sont associées à des expressions en ligne correspondantes concernant l'instance englobante. En d'autres termes, lorsque l'expression en ligne est interprétée comme une décomposition, cette expression combine chacun de ses opérandes avec ceux des autres expressions en ligne. Voir 7.4 pour plus de détails. Dans le contexte du document MSC englobant situé le plus à l'intérieur, une expression en ligne extraglobale est interprétée exactement comme n'importe quelle autre expression en ligne.

Les contraintes temporelles relatives aux événements de début et de fin d'une expression en ligne sont graphiquement rattachées au symbole d'expression en ligne <inline expression symbol> et sont indiquées dans la syntaxe textuelle après les mots clés **loop end**, **opt end**, **exc end**, **alt end** et **par end**. Les contraintes temporelles relatives au début ou à la fin d'une expression en ligne sont rattachées au sommet ou au bas (**attached to top or bottom**) du symbole d'expression en ligne <inline expression symbol>. Dans la syntaxe textuelle, les mots clés **startbefore**, **startafter**, **endbefore** et **endafter** servent à indiquer que l'événement de début de l'expression en ligne est situé avant ou après certains événements ou que l'événement de fin de l'expression en ligne est situé avant ou après certains événements, selon le cas. Voir 6.7 pour les concepts temporels.

7.3 Référence MSC

Les références MSC servent à faire référence à d'autres diagrammes MSC du document MSC. Les références MSC sont des objets du type indiqué par le diagramme cité en référence.

Les références MSC peuvent désigner non seulement un diagramme MSC donné mais aussi des expressions de référence MSC. Les expressions de référence MSC sont des expressions MSC textuelles qui sont construites à partir des opérateurs **alt, par, seq, loop, opt** et **exc** ainsi que de références MSC pouvant contenir des paramètres réels.

Les opérateurs **alt, par, loop, opt** et **exc** sont décrits en 7.2. L'opérateur **seq** désigne l'opération de composition séquentielle faible, dans laquelle seuls les événements relatifs à la même instance sont ordonnés.

Les références MSC simples peuvent contenir des paramètres réels qui doivent correspondre aux déclarations de paramètres correspondantes dans la définition du diagramme MSC.

Les portes effectives de la référence MSC peuvent établir une connexion avec des constructions correspondantes dans le diagramme MSC englobant, en ce sens qu'une porte de message effective peut établir une connexion avec une autre porte de message effective ou avec une instance ou avec une définition de porte de message du diagramme englobant. Par ailleurs, une porte d'ordre effective peut connecter à une autre porte d'ordre effective, ou à un événement ordonnable ou à une définition de porte d'ordre.

La libre utilisation de portes dans une référence MSC peut aisément conduire à la construction de diagrammes MSC comportant des interblocages imprévus. C'est la raison pour laquelle nous définissons des règles qui limitent l'utilisation graphique des portes. La règle de base est que l'ordre vertical graphique des messages commandés par porte, etc., figurant comme définitions de porte dans un diagramme MSC cité en référence doit être reproduit par les messages commandés par porte effectifs, etc., figurant dans le message MSC de référence. Cette règle empêche toute entité équivalant à une porte de dessiner des flèches de message inclinées vers le haut. La série complète de règles est donnée dans la section sur les exigences statiques graphiques.

Par ailleurs, nous notons que la présence d'un message commandé par porte sur une référence MSC ne garantit pas que ce message figurera dans toutes les traces (ni même dans une trace quelconque). Par exemple, le message commandé par porte peut commencer et prendre fin à l'intérieur d'expressions en ligne facultatives figurant dans les diagrammes MSC de définition correspondants.

Les références MSC temporisées servent à contraindre ou à mesurer la durée d'exécution de diagrammes MSC cités en référence et d'expressions de référence MSC.

Les intervalles temporels peuvent se rapporter au début ou/et à la fin d'une référence MSC. La valeur de début indique le moment où le premier événement intervient dynamiquement et la valeur de fin indique le moment du dernier événement.

Grammaire textuelle concrète

<shared msc reference> ::=
reference [<msc reference identification> **:**]
<msc ref expr> [**time** <time interval>] <shared> <end>
[**startbefore** <time dest list> <end>]
[**startafter** <time dest list> <end>]
[**endbefore** <time dest list> <end>]
[**endafter** <time dest list> <end>]
<reference gate interface>

<msc reference> ::=
reference [<msc reference identification>**:**]
<msc ref expr> [**time** <time interval>] <end>
<reference gate interface>

<msc reference identification> ::=
<msc reference name>

<msc ref expr> ::=
<msc ref par expr> { **alt** <msc ref par expr> }*

<msc ref par expr> ::=
<msc ref seq expr> { **par** <msc ref seq expr> }*

<msc ref seq expr> ::=
<msc ref ident expr> { **seq** <msc ref ident expr> }*

<msc ref ident expr> ::=
loop [<loop boundary>] <msc ref ident expr> |
exc <msc ref ident expr> |
opt <msc ref ident expr> |
empty |
<parent>* <msc name> [<actual parameters>] |
(<msc ref expr>)

<actual parameters> ::=
([<actual data parameters>]
[<actual instance parameters>][<actual message parameters>]
[<actual timer parameters>])

<actual instance parameters> ::=
inst <actual instance parm list> <end>

<actual instance parm list> ::=
<actual instance parameter> [,<actual instance parm list>]

<actual instance parameter> ::=
<instance name>

<actual message parameters> ::=
msg <actual message list> <end>

<actual message list> ::=
<message name> [, <actual message list>]

<actual timer parameters> ::=
timer <actual timer list> <end>

<actual timer list> ::=
<timer name> [, <actual timer list>]

<parent> ::=
#

<reference gate interface> ::=
{ <end> **gate** <ref gate> }*

<ref gate> ::=
<actual out gate> | <actual in gate> |
<actual order out gate> | <actual order in gate> |
<actual create out gate> | <actual create in gate> |
<actual out call gate> | <actual in call gate> |
<actual out reply gate> | <actual in reply gate>

Exigences statiques

Une référence MSC doit toujours se rattacher à chaque instance présente dans le diagramme englobant qui est contenu dans le diagramme MSC auquel cette référence MSC se rapporte. Si deux diagrammes, visés par deux références MSC contenues dans un diagramme englobant, partagent les mêmes instances, celles-ci doivent également apparaître dans le diagramme englobant.

L'interface de la référence MSC doit correspondre à l'interface des diagrammes MSC cités en référence dans l'expression. En d'autres termes, d'éventuelles portes attachées à la référence doivent avoir une définition de porte correspondante dans les diagrammes MSC cités en référence. Cette correspondance est indiquée par le sens et par le nom du message associé à la porte ainsi que par le nom de porte (s'il est présent), qui est unique dans l'expression citée en référence.

Si l'expression <msc ref expr> se compose d'une expression textuelle d'opérateur au lieu d'un simple nom <msc name> et si plus d'une seule référence MSC se rapporte au même diagramme MSC, le nom facultatif de référence msc <msc reference name> contenu dans l'identification <msc reference identification> doit être employé afin de désigner une référence MSC contenue dans la définition de message (voir 4.3).

L'interface de portes de référence <reference gate interface> doit énumérer toutes les portes du diagramme.

Grammaire graphique concrète

<msc reference area> ::=
<msc reference symbol>
[***top or bottom is attached to top or bottom***
{ {<int symbol> | <abs time symbol> }* } ***set***]
contains { <msc ref expr> [**time** <time interval>]
[<actual gate area>*] } ***set***
is attached to { <instance axis symbol>* } ***set***
is attached to { <actual gate area>* } ***set***

<msc reference symbol> ::=

La zone de référence <msc reference area> peut être attachée à une ou à plusieurs instances. Si une zone <msc reference area> croise un symbole d'axe d'instance <instance axis symbol> qui n'est pas englobé dans cette référence MSC, ce symbole d'axe d'instance est dessiné de façon à traverser cette zone.

Exigences statiques

L'ordre graphique des portes formelles de la définition MSC citée par la référence MSC doit être conservé par les portes réelles de la référence MSC. En notation textuelle, cela signifie que l'ordre des portes indiqué par l'interface <msc gate interface> est le même que l'ordre des portes réelles dans l'interface de portes de référence <reference gate interface>.

Lorsqu'une référence MSC contient des expressions de référence, la situation est plus complexe. En général, on doit construire un ensemble d'éléments ordonnés verticaux possibles à partir des parties constituantes de l'expression MSC. L'élément vertical effectivement retenu pour la référence MSC doit alors être un de ceux de cet ensemble.

Lorsqu'une expression n'est constituée que d'un nom MSC, l'ensemble d'éléments ordonnés verticaux est considéré comme n'étant constitué que d'un seul élément, à savoir l'élément vertical défini par le diagramme MSC cité en référence.

Pour une expression en alternative, chaque opérande doit définir le même ensemble d'éléments ordonnés verticaux, considéré alors comme représentant l'expression. Cette définition s'applique également aux expressions facultatives et exceptionnelles, considérées comme étant des formes abrégées d'alternatives.

Dans le cas d'une expression en parallèle, l'ensemble d'éléments ordonnés verticaux est construit par imbrication des éléments définis par chaque opérande. Par exemple, si on a *e* **par** *f*, où *e* définit l'ensemble des portes ordonnées verticalement {<a, b>, <b, a>} et *f* définit l'ensemble {<u, v>}, on doit alors imbriquer <u, v> avec <a, b> pour obtenir six éléments ordonnés dérivés, et également imbriquer <u, v> avec <b, a> pour obtenir six autres éléments ordonnés. Ainsi, *e* **par** *f* définit un ensemble de douze éléments ordonnés verticaux possibles. L'ordre réel des portes retenu pour la référence MSC doit donc être l'un de ces douze éléments ordonnés.

L'ensemble d'éléments ordonnés défini par *e* **seq** *f* est obtenu en ajoutant chacun des éléments ordonnés définis par *f* à la fin de chacun des éléments ordonnés définis par *e*. Par exemple, si *e* définit l'ensemble de portes ordonnées verticalement {<a, b>,<b, a>} et *f* définit l'ensemble {<u, v>}, l'ensemble obtenu est alors {<a, b, u, v>,<b, a, u, v>}.

Les références MSC ne doivent pas faire référence, que ce soit directement ou indirectement, à leur diagramme MSC englobant (par récurrence).

Les paramètres MSC réels doivent correspondre aux déclarations de paramètres correspondantes de la définition MSC. Les éléments séparés par le symbole <end> dans les paramètres réels <actual parameters> doivent correspondre de façon biunivoque aux éléments séparés par le symbole <end> dans la déclaration de paramètres <parameter declaration> de la définition MSC.

Les paramètres réels d'une instance doivent avoir la même sorte de déclaration de paramètres d'instance. La sorte des paramètres réels d'une instance peut être héritée de la sorte de la déclaration des paramètres de l'instance.

Les messages réels doivent avoir la même signature de message que la déclaration des paramètres de ces messages. Avoir la même signature pour un message signifie que la liste des paramètres correspondant à ce message est la même pour le message réel et pour la déclaration des paramètres du message.

Les temporisateurs réels doivent avoir la même signature de temporisateur que la déclaration des paramètres de ces temporisateurs. Avoir la même signature pour un temporisateur signifie que la liste des paramètres de ce temporisateur est la même pour le temporisateur réel et pour la déclaration des paramètres du temporisateur.

Un diagramme MSC contenant des références et des associations de paramètres réels est illégal si, après expansion répétée des références et association répétée des paramètres réels, un diagramme MSC illégal en résulte.

Sémantique

Chaque diagramme MSC peut être vu comme une définition d'un type MSC. Les types MSC peuvent être utilisés dans d'autres types MSC au moyen de références MSC.

Un type MSC peut être attaché à son environnement au moyen de portes. Celles-ci sont utilisées pour définir les points de connexion lorsqu'un type MSC est utilisé dans un autre type. Des identificateurs de porte peuvent être associés aux points de connexion sous forme de noms.

Les instances qui sont attachées à la référence MSC sont par défaut des paramètres réels d'instance de la référence MSC. En d'autres termes, si la sorte d'instance doit identifier de manière univoque le paramètre formel d'instance, les instances attachées n'ont pas besoin de figurer dans la liste des paramètres réels.

Une référence MSC peut être attachée à des instances qui ne sont pas contenues dans le diagramme cité en référence.

En général, la référence MSC peut se rapporter à une expression MSC textuelle. Dans le cas simple où l'expression MSC ne se compose que d'un nom de diagramme MSC, la référence MSC désigne une définition de type MSC correspondante. La correspondance est indiquée par le nom MSC, qui est unique à l'intérieur du document MSC. Lorsque le diagramme MSC possède des paramètres réels, le résultat est que les paramètres formels du diagramme cité en référence sont remplacés par les paramètres réels de la référence MSC.

Dans la représentation textuelle, l'interface des portes de référence <reference gate interface> définit les messages pénétrant ou quittant la référence MSC en passant par des portes. Au moyen de l'identification des noms de message et de l'identification facultative des noms de porte, l'interface des portes de référence définit également la connexion directe de messages entre deux références MSC.

Une référence MSC contenant le mot clé **empty** se rapporte à un diagramme MSC ne comportant ni événements ni instances.

Le préfixe <parent> devant une référence MSC indique que celle-ci se rapporte au diagramme MSC qu'elle redéfinit par héritage plutôt qu'au diagramme MSC portant ce nom à l'intérieur de la sorte d'instance héritière. Le préfixe <parent> peut être répété afin d'accéder à des ascendants de degré supérieur, etc.

Les contraintes temporelles concernant l'événement de début et l'événement de fin d'une expression MSC sont graphiquement attachées au symbole de référence MSC <msc reference symbol> et sont indiquées dans la syntaxe textuelle après le mot clé **time**. Les contraintes temporelles se rapportant soit au début soit à la fin d'une expression MSC sont attachées au sommet ou à la base (**attached to top or bottom**) du symbole de référence MSC <msc reference symbol>. Dans la syntaxe textuelle, les mots clés **startbefore**, **startafter**, **endbefore** et **endafter** servent à indiquer que l'événement de début ou de fin (selon le cas) de l'expression MSC intervient avant ou après certains événements.

7.4 Décomposition d'instance

Un document MSC contient un ensemble d'instances, dont chacune appartient à une sorte d'instance. Les instances qui ne possèdent pas de référence à leur sorte d'instance ont implicitement la sorte qui a le même nom de l'instance. Pour décrire l'interaction à différents niveaux de détail, le langage MSC introduit la décomposition.

La structure interne et le comportement d'une sorte d'instance sont définis au moyen d'un document MSC portant le même nom que la sorte d'instance. Il existera donc une hiérarchie de documents MSC définissant la hiérarchie d'instance. Pour indiquer la façon dont les comportements des différents niveaux sont rattachés les uns aux autres, l'on peut spécifier le comportement d'une instance à l'intérieur d'un diagramme MSC pour l'affiner dans un diagramme MSC du document MSC définissant l'instance à décomposer.

Un document MSC ne peut donc être interprété que par rapport à ses propres instances, sans tenir compte d'une quelconque décomposition. Il peut également être interprété par rapport à des niveaux inférieurs d'instances par application des relations de décomposition.

Le langage MSC exige qu'il y ait une analogie structurelle entre l'instance décomposée et la décomposition correspondante, mais il n'est pas obligatoire qu'il y ait un quelconque affinement comportemental.

Grammaire textuelle concrète

<decomposition> ::=
decomposed [<substructure reference>]

<substructure reference> ::=
as <message sequence chart name>

Grammaire graphique concrète

La grammaire graphique est indiquée au 4.2.

Exigences statiques

Pour chaque instance contenant le mot clé **decomposed**, il faut spécifier un diagramme MSC d'affinement correspondant, ayant le même nom que l'instance décomposée. Si la référence substructurelle <substructure reference> est ajoutée après le mot clé **decomposed**, le diagramme MSC d'affinement doit avoir le nom spécifié dans la référence substructurelle <substructure reference>.

L'instance décomposée peut être interprétée comme une séquence de structures linguistiques dont certaines peuvent être considérées comme des portes en relation avec le diagramme de décomposition. La Figure 11 donne un aperçu général des concepts assimilables à des portes. Il existe des *portes homologues* et des *portes de décomposition*. Les portes homologues sont également indiquées par la connexion de références MSC, tandis que les portes de décomposition ne sont applicables qu'à des instances décomposées.

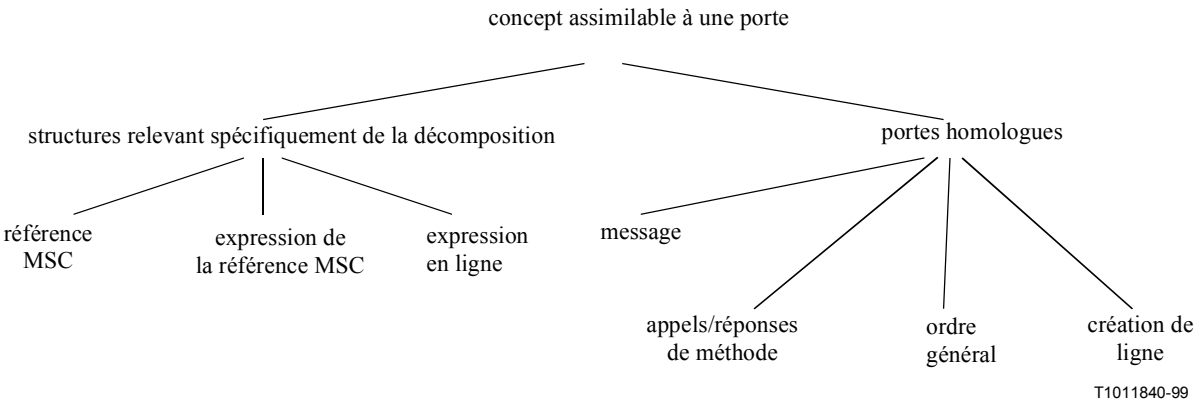


Figure 11/Z.120 – Concepts assimilables à des portes

L'interprétation des concepts de porte relevant spécifiquement de la décomposition est indiquée dans la section de sémantique ci-dessous. L'interprétation des portes homologues est indiquée dans le paragraphe relatif aux portes et aux appels de méthode.

L'ordre graphique des portes doit être conservé statiquement lors du passage de l'instance décomposée au diagramme de décomposition, c'est-à-dire que si les deux arcs verticaux du diagramme de décomposition sont superposés, la séquence des définitions de porte du diagramme doit être la même que celle de l'instance décomposée. En notation textuelle, cela signifie que l'ordre de l'interface de porte MSC <msc gate interface> est le même que l'ordre des événements correspondants dans l'instance décomposée.

Soit l'instance *i* dans le diagramme MSC *M* du document MSC *D* de la Figure 12. En même temps que cette instance, il y a en séquence: une expression en alternative, une simple référence MSC (à *A*), une sortie de message *s* et une expression de référence MSC (*B* **alt** *C*). La décomposition se trouve dans le diagramme MSC *iM* de la Figure 16, qui contient en séquence: une expression en ligne extraglobale, une simple référence MSC (à *iA*), une porte homologue de sortie de message *s* et une expression de référence MSC (*iB* **alt** *iC*).

Avec les portes homologues, ce séquencement est simple car il existe des événements dans l'instance décomposée qui correspondent aux portes homologues de la décomposition. La situation est un peu plus compliquée avec les concepts assimilables à des portes relevant spécifiquement de la décomposition. Les constructions d'une instance qui ne correspondent pas à un concept assimilable à une porte sont exclues des exigences statiques relatives à la décomposition. De telles constructions sont des actions, des temporisateurs et des corégions. Celles-ci sont considérées, à cet égard, comme une simple zone d'instance.

Lorsque l'instance décomposée est arrêtée, la décomposition doit inclure les arrêts relatifs à toutes les instances qu'elle contient.

Exigences statiques

L'exigence statique suivante doit s'appliquer aux références MSC concernant une instance donnée. Soit *i* une instance décomposée à l'intérieur du diagramme MSC *M* (Figure 12). L'instance *i* dans *M* est décomposée en *iM* (Figure 16). Soit une référence MSC *A* qui se rapporte à *i* dans *M*. Cette référence *A* est donc une référence MSC assimilable à une porte. Elle doit être assortie, par une référence globale[2] correspondante dans *iM*, à (par exemple) une instance *iA* (Figure 17) définie dans le document MSC qui définit l'instance *i*. A l'intérieur de la référence *A*, l'instance *i* doit alors être décomposée en *iA* (Figure 13).

[2] Ici, le qualificatif "globale" désigne toutes les instances contenues dans le document MSC.

L'exigence statique suivante s'applique aux expressions en ligne couvrant une certaine instance décomposée (voir Figure 12). La décomposition doit contenir une expression en ligne correspondante, de la même structure d'opération et d'opérande (voir Figure 16). L'expression en ligne contenue dans la décomposition doit être extraglobale (voir l'article relatif aux expressions en ligne), ce qui indique que les opérandes sont connectés à d'autres opérandes d'expressions en ligne similaires, lorsqu'ils sont interprétés par décomposition. Lorsqu'ils sont interprétés seulement dans le contexte du plus proche document MSC englobant, une expression en ligne extraglobale est traitée comme une simple expression en ligne globale.

Par récurrence, chaque opérande de l'expression en ligne est soumis aux mêmes exigences statiques que l'ensemble de l'instance décomposée.

L'exigence statique suivante est applicable aux expressions de référence MSC couvrant une certaine instance décomposée. La décomposition doit contenir une expression de référence MSC correspondante, ayant la même structure d'expression. Chacun des opérandes de l'expression de référence MSC est soumis aux exigences statiques pour les expressions de référence MSC ou pour les références MSC.

Une instance qui est décomposée en un seul diagramme MSC doit être décomposée en tous les diagrammes MSC qui dépendent de ce diagramme MSC par référenciation.

Sémantique

La sémantique de la décomposition est définie au moyen d'un modèle de transformation. L'algorithme ci-après transforme un diagramme MSC comportant la décomposition d'un seul document MSC en un autre diagramme MSC comportant un autre diagramme MSC (construit) dans lequel l'instance décomposée a été subdivisée en ses éléments constituants. Un exemple accompagnera ces transformations.

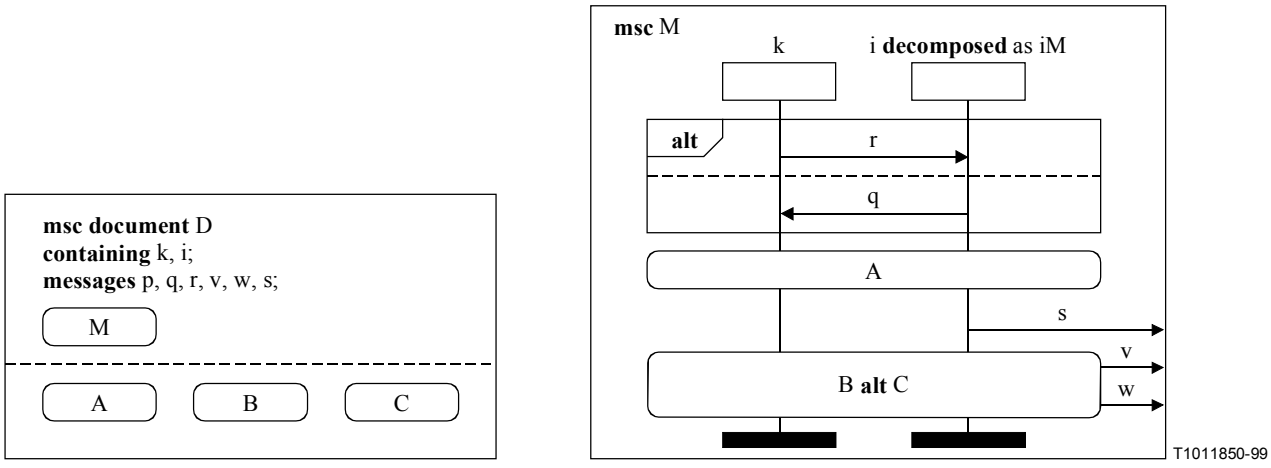


Figure 12/Z.120 – Document MSC de niveau supérieur et diagramme MSC définisseur

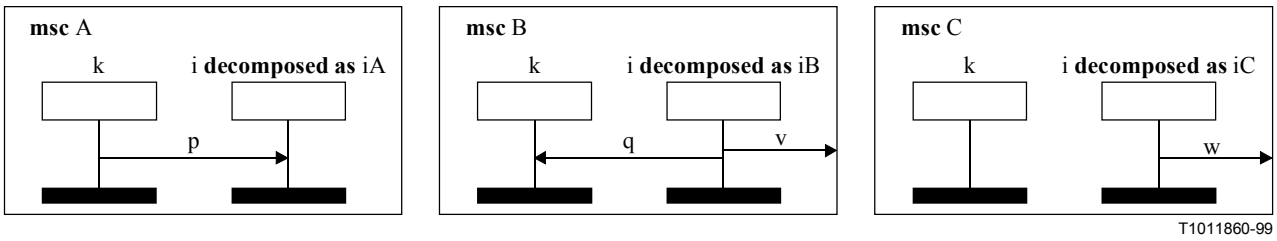


Figure 13/Z.120 – Utilitaires au niveau supérieur

Règle 1: transformation des expressions de référence MSC en expressions en ligne. Voir Figure 14.

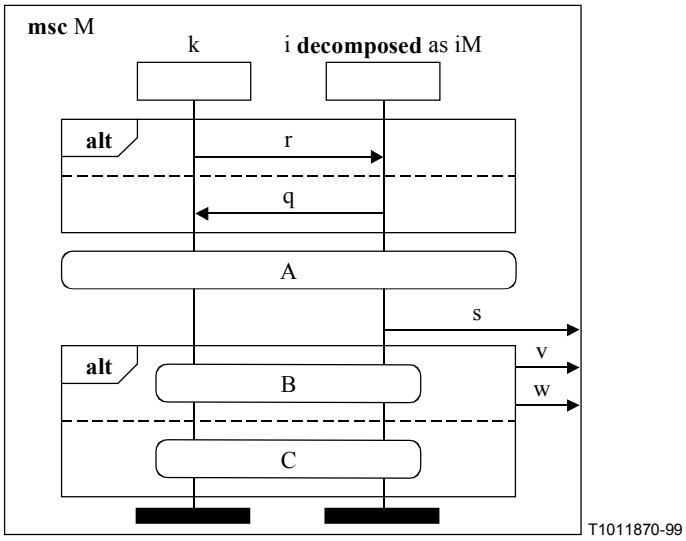


Figure 14/Z.120 – Après utilisation de la règle 1

Règle 2: résolution des références MSC par remplacement du contenu des diagrammes MSC chaque fois que ces références apparaissent. Correspondance avec les portes. Voir Figure 15.

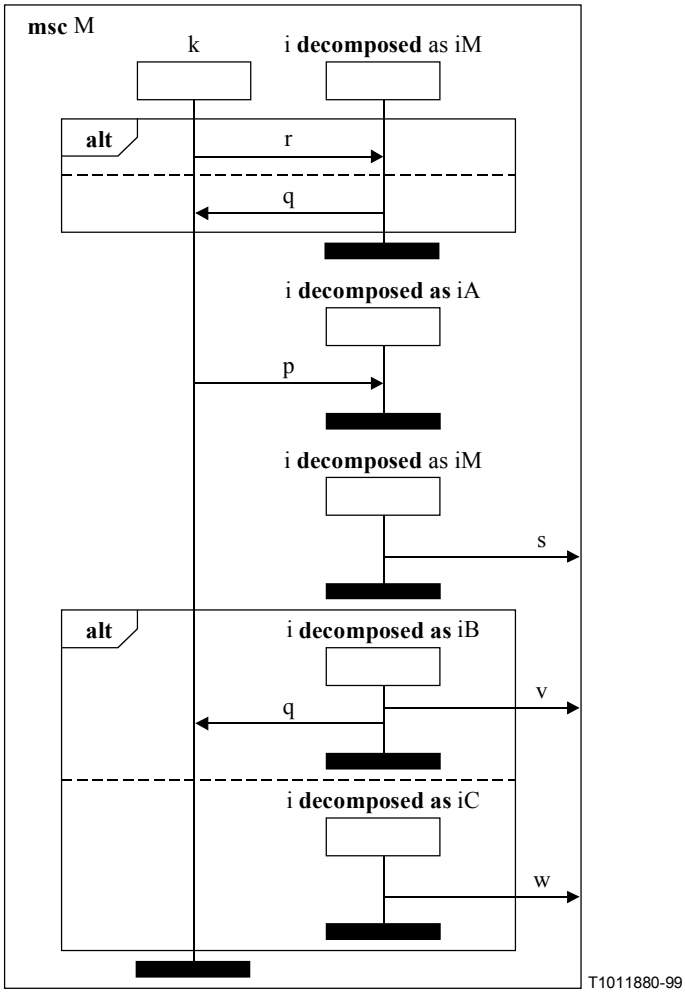


Figure 15/Z.120 – Après utilisation de la règle 2

L'algorithme a maintenant atteint un diagramme dans lequel les décompositions se produisent successivement et conformément aux exigences statiques. Les instances qui ne sont pas décomposées sont connectées trivialement par les portes.

L'algorithme passe ensuite à la décomposition et fait apparaître la nécessité de montrer la définition du document MSC *i*.

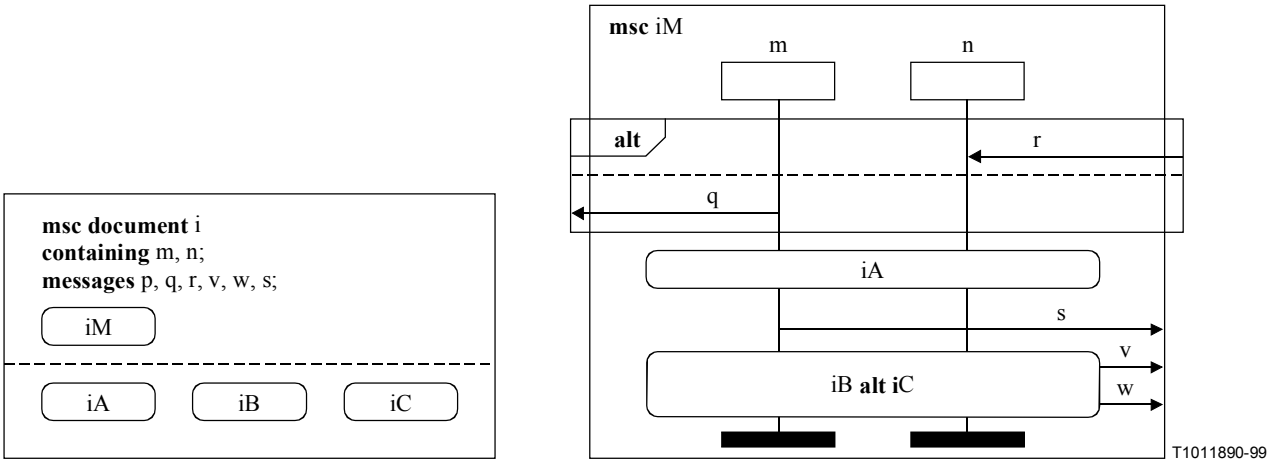


Figure 16/Z.120 – Document MSC de niveau inférieur

Il convient de remarquer l'expression en ligne extraglobale et le fait que les exigences statiques sont satisfaites dans *iM* par rapport à *i* dans *M* du document MSC *D* de la Figure 12.

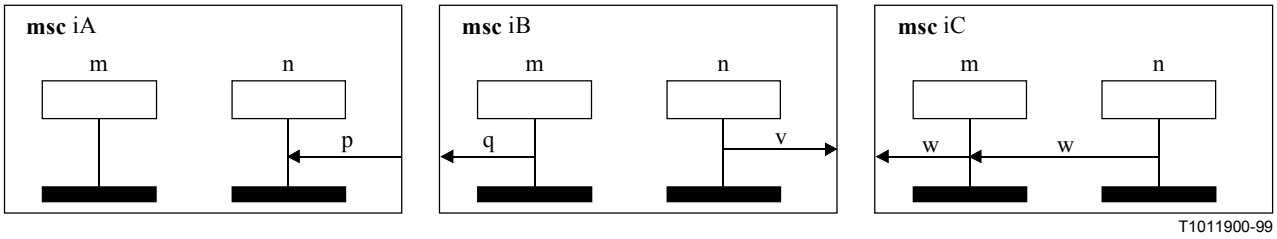


Figure 17/Z.120 – Utilitaires au niveau le plus bas

Règle 3: transformation des instances décomposées ayant seulement des portes homologues par simple remplacement du contenu du diagramme et mise en correspondance avec les portes homologues. Voir Figure 18.

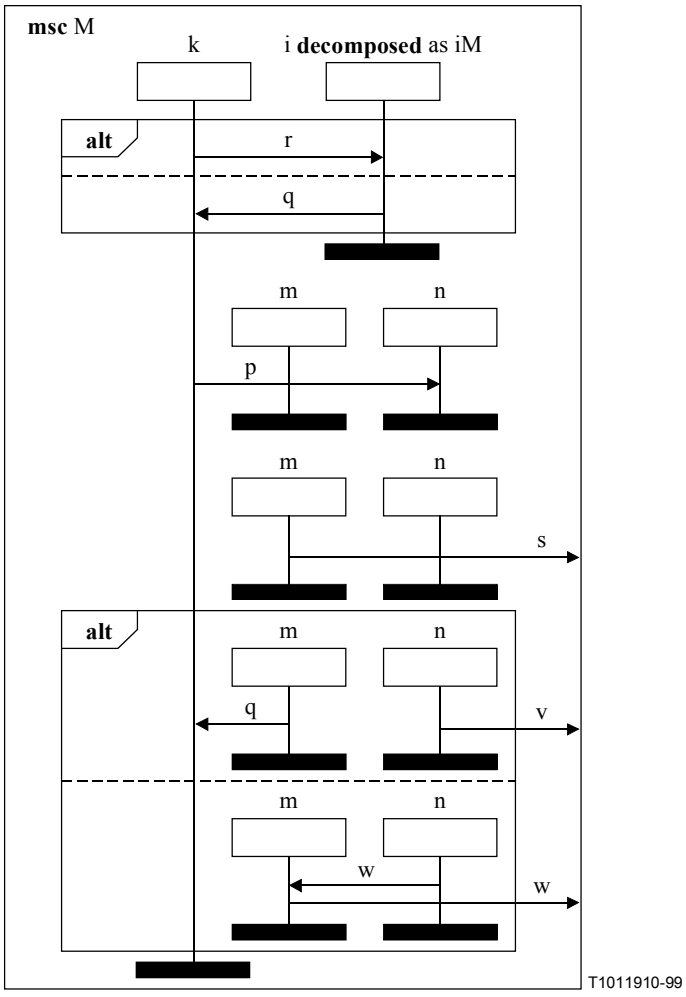


Figure 18/Z.120 – Après utilisation de la règle 3

La règle 3 donne des instructions permettant de remplacer de simples diagrammes MSC et de faire correspondre leurs portes. D'après cet exemple, l'on peut voir les substitutions et les substituts. A partir du point de départ de la Figure 15, considérons la décomposition de l'instance *i* dans *M*,

représentée par *iM* dans la Figure 16. Les remplacements suivants se produisent de haut en bas. Tout d'abord, l'on sort de l'expression en ligne pour passer à la règle inférieure suivante. En deuxième lieu, il existe dans *iM* la référence à *iA* qui correspond bien au fragment indiqué. L'on utilise *iA* de la Figure 17 pour le remplacement. En troisième lieu, l'on a la sortie de *s* décomposée directement dans *iM*. Comme substitut, l'on utilise la partie de *iM* qui est comprise entre la référence à *iA* et l'expression de référence MSC. En quatrième lieu, l'on arrive à l'expression de référence MSC (*iB* **alt** *IC*) et l'on constate qu'elle correspond bien à l'expression en ligne. L'on effectue la substitution conformément aux mêmes principes à l'intérieur de chaque opérande.

Règle 4: transformation des expressions en ligne par remplacement avec les expressions en ligne extraglobales correspondantes, de façon que les opérandes soient connectés selon une relation biunivoque. S'il y a des expressions imbriquées, cette règle s'applique par récurrence à chaque opérande. Voir Figure 19.

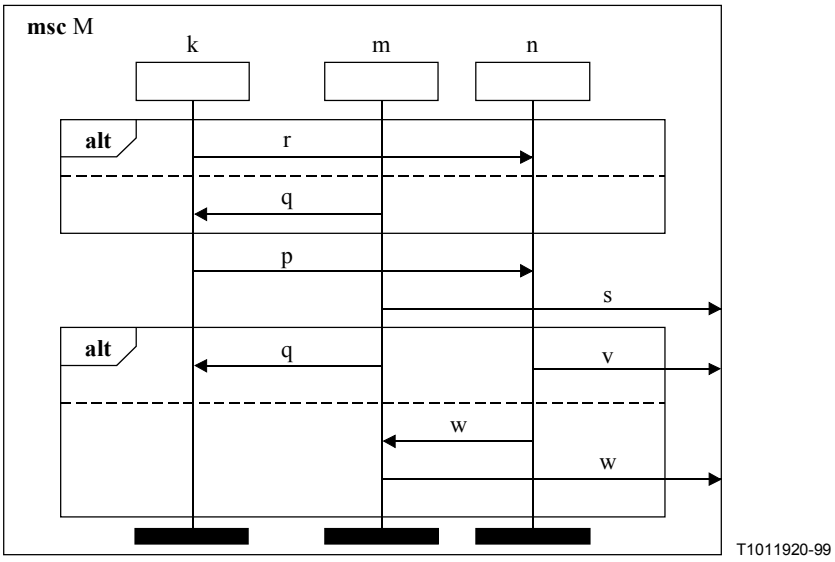


Figure 19/Z.120 – Après utilisation de la règle 4

Dans la Figure 19, la règle 4 a été appliquée et l'expression extraglobale a été remplacée par un opérande après l'autre. Par ailleurs, le diagramme a été simplifié par combinaison des fragments en une seule instance chacun.

Le diagramme MSC M résultant aurait été un MSC contenu dans un document MSC D' contenant les instances k, m, n.

Ce processus de transformation aurait tout aussi bien pu être exprimé en suivant d'abord la décomposition, avant de résoudre les références. Une notation supplémentaire aurait été nécessaire pour l'intersection entre une (expression de) référence MSC et le contenu du diagramme de décomposition.

7.5 Diagramme MSC de haut niveau (HMSC)

Les MSC de haut niveau permettent de définir graphiquement les combinaisons possibles d'un ensemble de diagrammes MSC. Un HMSC est essentiellement un graphe où chaque nœud est soit:

– un symbole de départ (il n'y a qu'un seul symbole de départ dans chaque HMSC);
– un symbole de fin;
– une référence MSC;
– une condition;

– un point de connexion;

– un cadre parallèle.

Les lignes de flux relient les nœuds dans le HMSC et indiquent la séquence possible entre les nœuds dans le HMSC. Les lignes de flux entrantes sont toujours reliées au bord supérieur du nœud alors que les lignes de flux sortantes sont reliées au bord inférieur. Si plus d'une ligne de flux sort d'un nœud, il s'agit d'une alternative.

Les références MSC peuvent être utilisées soit pour faire référence à un diagramme MSC simple soit à un certain nombre de diagramme MSC utilisant une expression MSC textuelle.

Les conditions dans les HMSC peuvent être utilisées pour définir des états globaux du système et imposent des restrictions sur les MSC référencés dans le HMSC.

Les cadres parallèles contiennent un ou plusieurs petits HMSC constituant les opérandes d'un opérateur parallèle, c'est-à-dire que les événements dans les différents petits HMSC peuvent être imbriqués.

Les points de connexion sont introduits pour simplifier la mise en page des HMSC et n'ont pas de portée sémantique.

Les diagrammes MSC de haut niveau peuvent être contraints et mesurés au moyen d'intervalles temporels pour expressions MSC. Par ailleurs, la durée d'exécution d'un cadre parallèle de diagramme HMSC peut être contrainte ou mesurée. L'interprétation est analogue à celle des expressions MSC temporisées.

Grammaire graphique concrète

<hmsc> ::=
expr <msc expression>

<msc expression> ::=
<start> {<node expression> | <text definition>} *

<start> ::=
<label name list> <end>

<node expression> ::=
<label name> **:** { <node> **seq** (<label name list>) | **end** }<end>

<label name list> ::=
<label name> { **alt** <label name> }*

<node> ::=
(<msc ref expr>) [<time interval>]
| <par expression> [<time interval>]
| <condition identification>
| **connect**

<par expression> ::=
expr <msc expression> **endexpr**
{ **par expr** <msc expression> **endexpr** }*

Exigences statiques

Tous les noms référencés <label name> dans <start> ou <node expression> doivent être définis dans le diagramme HMSC, c'est-à-dire apparaître sous la forme "<label name>**:**" dans une expression <node expression>. Chaque nœud dans un graphe HMSC doit être accessible à partir du symbole <start>, c'est-à-dire que le graphe doit être connecté.

Grammaire graphique concrète

<mscexpr area> ::=
{<text layer> <start area> <node expression area>*
<hmsc end area>* } ***set***

<start area> ::=
<hmsc start symbol> ***is followed by*** { <alt op area>$^{+}$ } ***set***

<hmsc start symbol> ::=

<hmsc end area> ::=
<hmsc end symbol> ***is attached to*** { <hmsc line symbol>$^{+}$} ***set***

<hmsc end symbol> ::=

<hmsc line symbol> ::=
<hmsc line symbol1> | <hmsc line symbol2>

<hmsc line symbol1> ::=

<hmsc line symbol2> ::=

<alt op area> ::=
<hmsc line symbol> ***is attached to*** { <node area> | <hmsc end symbol> }

<node expression area> ::=
<node area> ***is followed by*** { <alt op area>$^{+}$} ***set***
is attached to { <hmsc line symbol>$^{+}$} ***set***

<node area> ::=
<hmsc reference area>
| <connection point symbol>
| <hmsc condition area>
| <par expr area>

<hmsc reference area> ::= <msc reference symbol>
[***top or bottom is attached to top or bottom***
{ {<int symbol> | <abs time symbol> }* } ***set***]
contains <msc ref expr> [**time** <time interval>]

<connection point symbol> ::=

<hmsc condition area> ::=
<condition symbol> ***contains*** <condition text>

<par expr area> ::=
<par frame symbol>
is attached to <hmsc line symbol>
[***top or bottom is attached to top or bottom***
{ {<int symbol> | <abs time symbol> }* } ***set***]
contains { <mscexpr area>+} ***set***

<par frame symbol> ::=
<frame symbol>

Règles de dessin

Le symbole <hmsc line symbol> peut être une ligne brisée et avoir n'importe quel sens.

Le fait que le symbole <hmsc start symbol> soit suivi par le symbole <alt op area> signifie que les symboles <hmsc line symbol> sont reliés au coin inférieur du symbole <hmsc start symbol>. C'est ce que montre la Figure 20, où deux lignes <hmsc line symbol> suivent le symbole de départ <hmsc start symbol>:

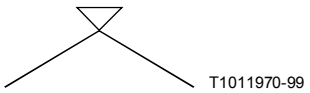


Figure 20/Z.120 – Règles de dessin HMSC

Le fait que le symbole <hmsc line symbol> soit attaché à un autre symbole dans la règle de production <alt op area> signifie que les symboles <hmsc line symbol> doivent être reliés au bord supérieur du symbole en question. La Figure 21 illustre les cas où le symbole est un symbole <msc reference symbol> et un symbole <hmsc end symbol>:

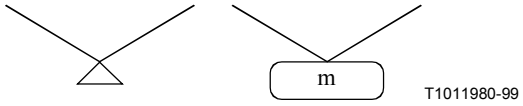


Figure 21/Z.120 – Règles de dessin HMSC

Le fait que la zone <node area> soit suivie par une zone <alt op area> dans la règle de production <node expression area> signifie que les symboles <hmsc line symbol> doivent être reliés au bord inférieur du symbole dans la zone <node area>. La Figure 22 montre comment un symbole <msc reference symbol> est suivi par une zone <alt op area>:

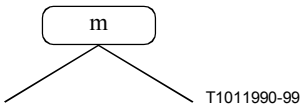


Figure 22/Z.120 – Règles de dessin HMSC

Sémantique

Quant à l'exécution, le graphe décrivant la composition de diagramme MSC à l'intérieur d'un diagramme HMSC est interprété de la façon suivante. L'exécution commence au symbole <hmsc start symbol>. Puis elle continue avec un nœud suivant l'un des arcs sortant de ce symbole. Ces nœuds sont considérés comme étant des opérandes de l'opérateur **alt** (voir 7.2). Après exécution du nœud sélectionné, le processus de sélection et d'exécution est répété pour les arcs sortant de ce nœud. L'exécution d'un nœud terminal signifie la fin de l'exécution du diagramme HMSC. Une référence MSC s'exécute conformément à la description du 7.3. L'exécution d'un point de connexion est une opération vide. L'exécution d'un cadre parallèle consiste en l'exécution en parallèle des opérandes de ce cadre comme décrit au 7.2 pour l'opérateur **par**. Une exécution séquentielle de deux nœuds reliés par un arc est décrite par l'opérateur **seq** (voir 7.3).

Une condition de garde signifie que l'exécution ne peut pas continuer au-delà de cette condition si son analyse logique donne la valeur *false*. Si toutes les branches disponibles sont bloquées par des gardes de valeur *false* et si aucune extrémité de diagramme HMSC n'a été atteinte, l'ensemble du diagramme HMSC n'a pas de traces légales.

EXEMPLES DE DIAGRAMMES DES SÉQUENCES DE MESSAGES

La présente partie est informative plutôt que normative.

8 Document de diagramme des séquences de messages

8.1 Documents MSC

Le diagramme de la Figure 23, représentant un document MSC, montre que l'instance *ACContext* contient les instances *ACSystem, User, Supervisor* et *NewUser*. Les diagrammes MSC définisseurs (ou publics) sont *UserAccess, PIN_Change* et *NewUser*. Noter que *NewUser* est à la fois le nom d'une instance englobante et le nom d'un diagramme MSC contenu. Cela est légal puisque les instances et les diagrammes appartiennent à des classes d'entité différentes.

L'interface avec le langage de données est également indiquée. Le langage de données est le langage "C" et le caractère de soulignement est utilisé comme caractère générique variable. Les définitions de données sont contenues dans le fichier cdefs.h.

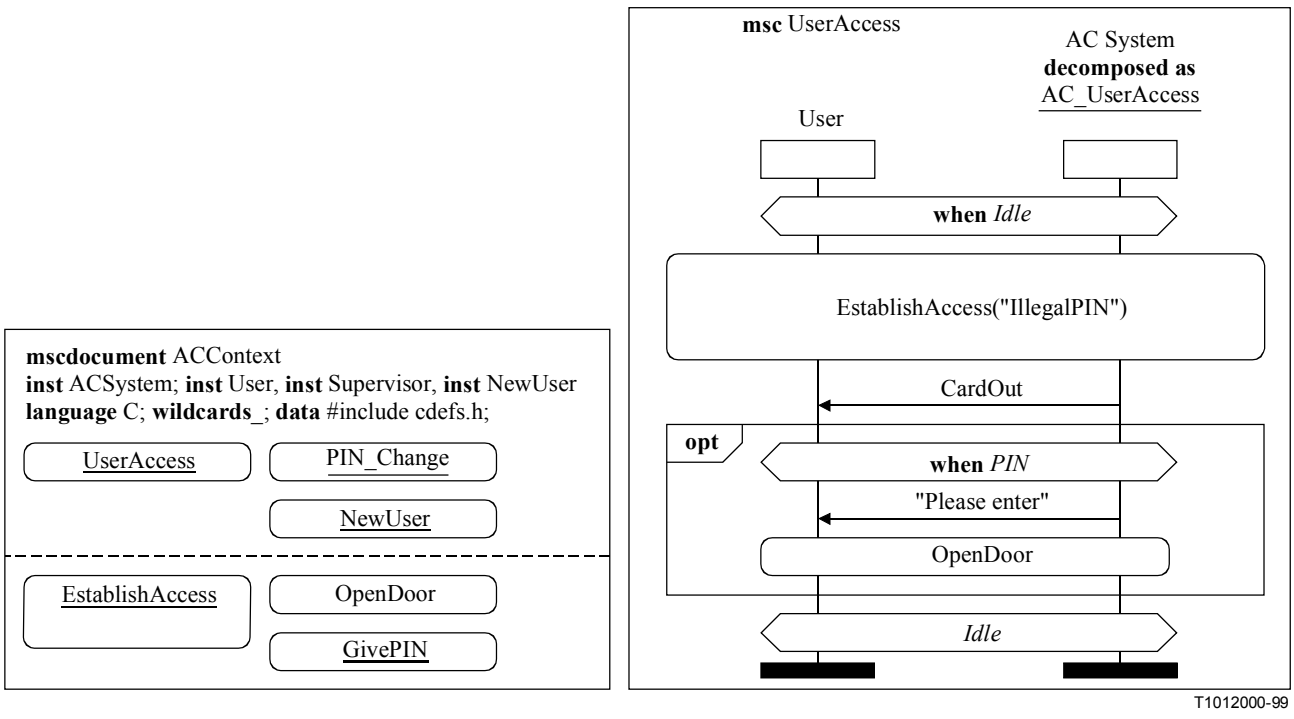


Figure 23/Z.120 – Instance ACContext d'un document MSC

Pour illustrer les exigences statiques de cohérence lors de la décomposition, l'on a également reproduit la séquence MSC *UserAccess* référencé par l'instance *ACContext* du document MSC.

8.2 Décomposition d'instance

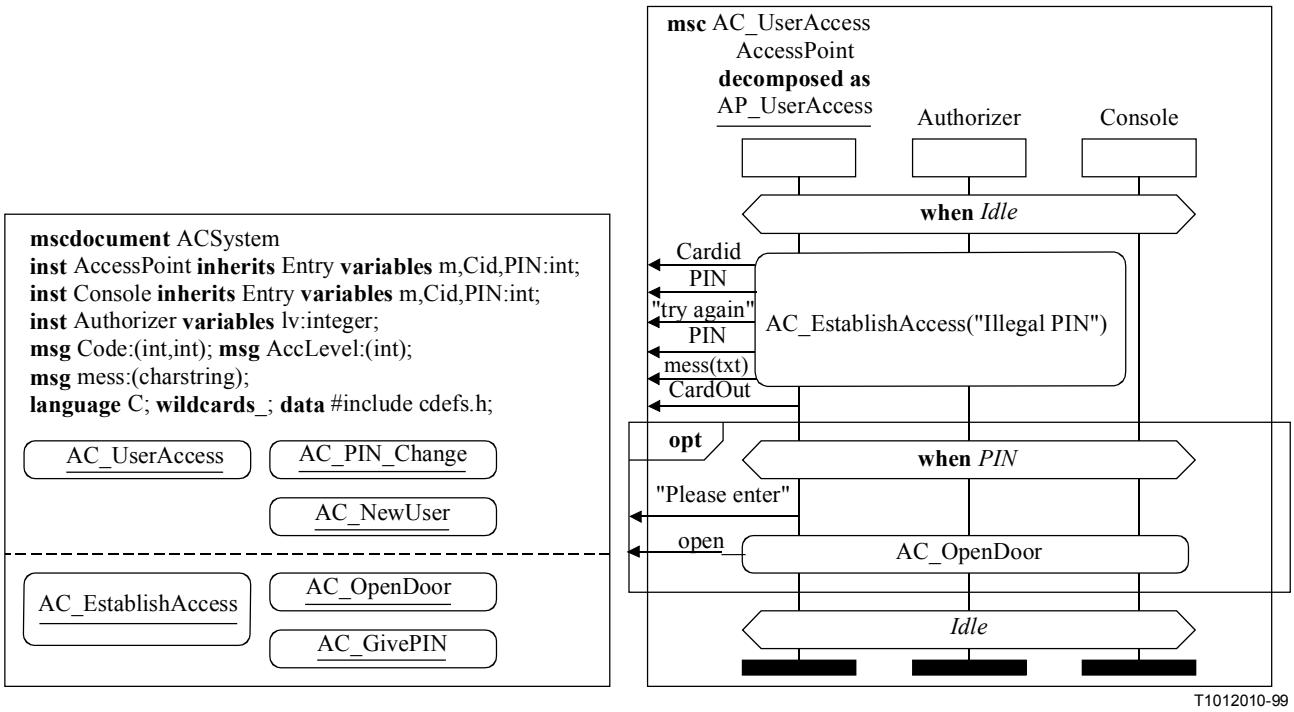


Figure 24/Z.120 – Document MSC au niveau immédiatement inférieur

Le document MSC *ACSystem* représenté sur la Figure 24 est donc la description de l'instance *ACSystem* indiquée dans la Figure 23. Cette description montre aussi l'héritage décrit pour les instances contenues. Dans la Figure 27 ci-dessous, l'on a inséré la définition de la sorte d'instance *Entry*.

L'on a également inséré des déclarations de variables dynamiques pour les instances et les messages qui possèdent des paramètres.

Les exigences statiques pour la décomposition et les références MSC peuvent être vérifiées d'après cet exemple. Dans l'instance *ACContext*, l'on trouve le diagramme MSC *UserAccess* dans lequel l'instance contenue *ACSystem* est décomposée par *AC_UserAccess* qui, à son tour, fait référence au diagramme *AC_EstablishAccess.*

D'autre part, la séquence *UserAccess* fait référence au diagramme *EstablishAccess* représenté sur la Figure 25.

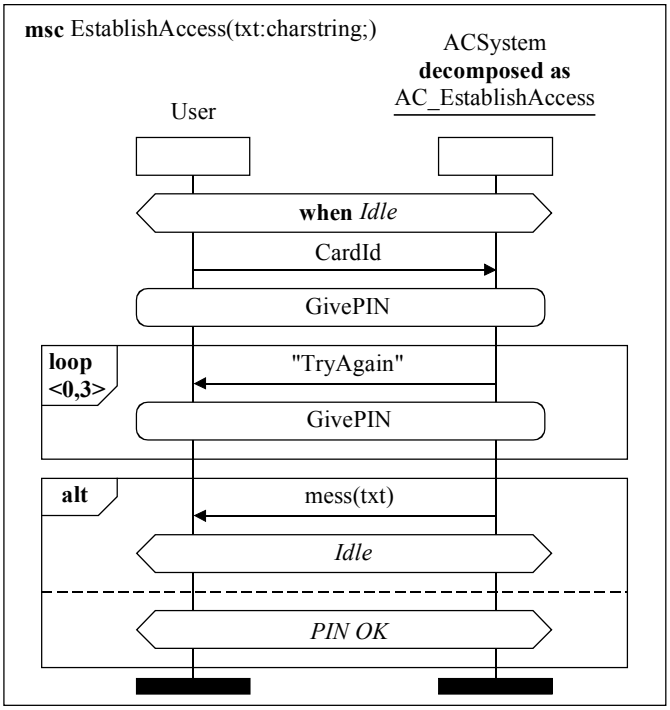


Figure 25/Z.120 – Diagramme MSC EstablishAccess

A l'intérieur du diagramme *EstablishAccess*, qui est un utilitaire de l'instance *ACContext*, l'instance *ACSystem* est décomposée en *AC_EstablishAccess*, qui représente donc les points de confluence de la référenciation et de la décomposition à partir du diagramme UserAccess de l'instance *ACContext*. En d'autres termes, que la décomposition ou la référenciation soit suivie la première, aucune différence ne devrait apparaître à la fin du processus.

Dans la séquence *EstablishAccess*, il y a une définition d'une variable statique *txt*. Le paramètre réel peut être vu sur la Figure 24.

Par souci d'exhaustivité, la séquence *AC_EstablishAccess* est représenté sur la Figure 26.

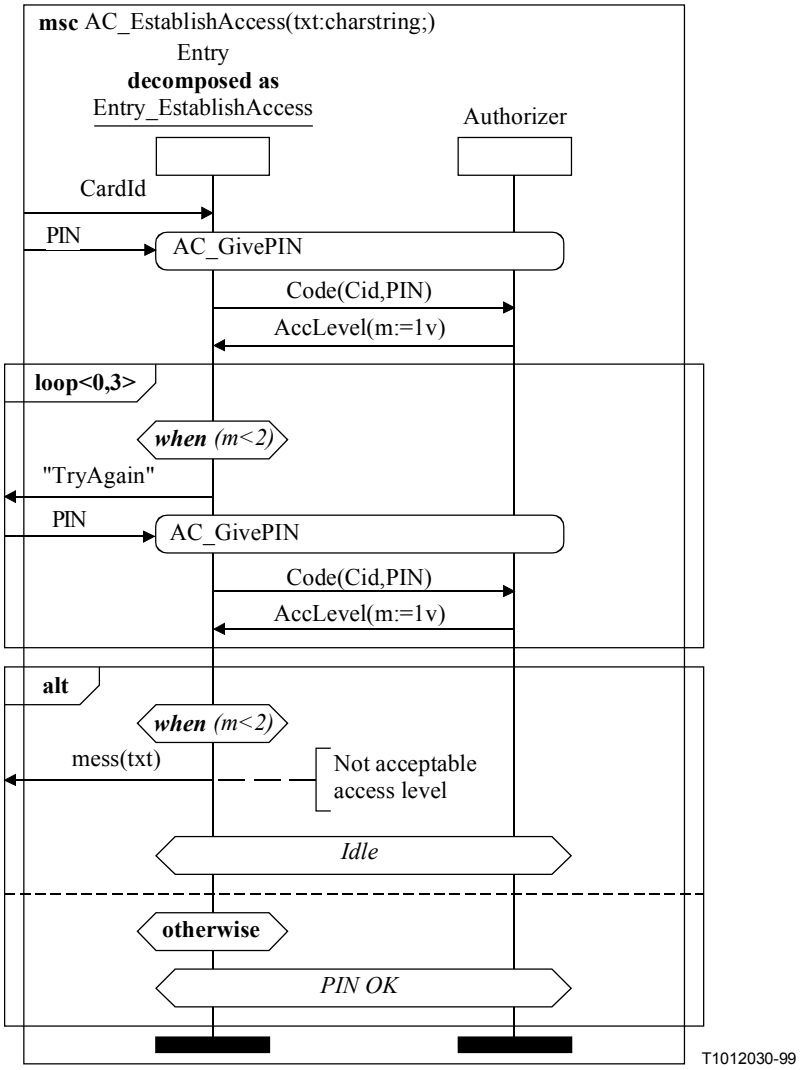


Figure 26/Z.120 – Diagramme MSC AC_EstablishAccess

A l'intérieur du diagramme *AC_EstablishAccess*, il y a à la fois des conditions de sorte état qui sont utilisées comme gardes et configuration, et des expressions booléennes qui sont utilisées comme gardes. Des associations sont représentées avec certains des messages.

8.3 Héritage d'instance

Voir les Figures 27 et 28.

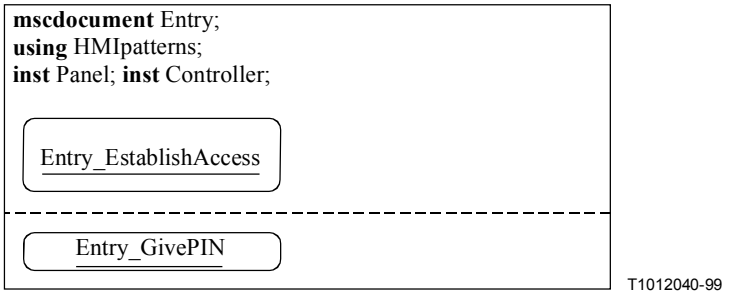


Figure 27/Z.120 – Entrée de document MSC

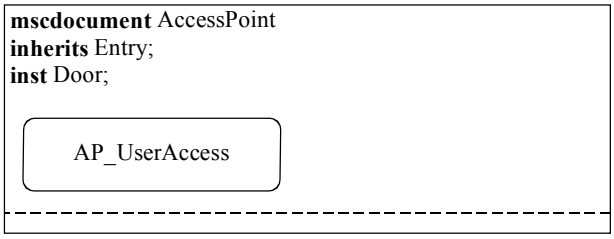


Figure 28/Z.120 – Document MSC AccessPoint héritant d'une entrée

Dans le document MSC *AccessPoint*, l'on voit qu'il n'est pas nécessaire de redéfinir les instances *Panel* et *Controller* qui sont déjà définies dans l'entrée héritée (*Entry*).

9 Diagrammes MSC simples

9.1 Diagramme MSC de base

Cet exemple illustre une connexion simplifiée réalisée dans un système de commutation. L'exemple montre les constructions MSC les plus basiques: instances (de processus), environnement, messages et conditions globales.

L'état de repos est un état initial. L'état de prise est intermédiaire et l'état de conversation est final.

Voir Figure 29.

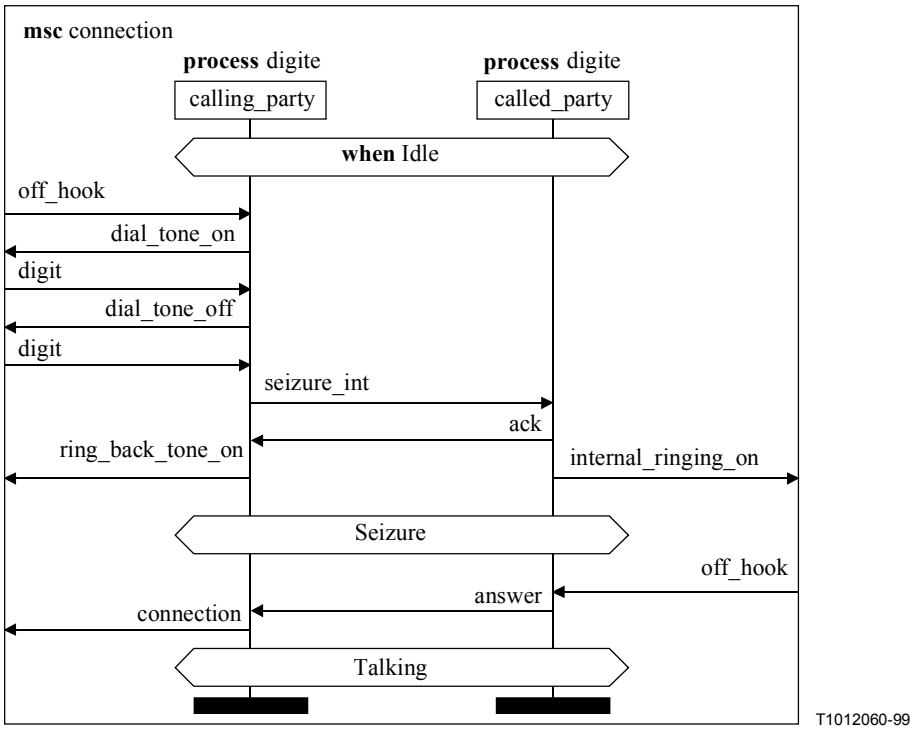


Figure 29/Z.120 – Connexion MSC

```
msc connection;
inst calling_party: process digite;
inst  called_party: process digite;
gate out off_hook to calling_party;
gate in dial_tone_on from calling_party;
```

```
gate out digit to calling_party;
gate in dial_tone_off from calling_party;
gate out digit to calling_party;
gate in ring_back_tone_on from calling_party;
gate in internal_ringing_on from called_party;
gate out off_hook to called_party;
gate in connection from calling_party;

        calling_party : instance process digite;
            condition when Idle shared all;
            in off_hook from env;
            out dial_tone_on to env;
            in digit from env;
            out dial_tone off to env;
            in digit from env;
            out seizure_int to called_party;
            in ack from called_party;
            out ring_back_tone on to env;
            condition Seizure shared all;
            in answer from called_party;
            out connection to env;
            condition Talking shared all;
        endinstance;
        called_party: instance process digite;
            condition when Idle shared all;
            in seizure_int from calling_party;
            out ack to calling_party;
            out internal_ringing_on to env;
            condition Seizure shared all;
            in off_hook from env;
            out answer to calling_party;
            condition Talking shared all;
        endinstance;
endmsc;
```

9.2 Dépassement de message

Cet exemple montre le dépassement de deux messages ayant le même nom de message 'message1'. Dans la représentation textuelle, les noms d'instance de message (a, b) sont employés pour une correspondance unique entre entrée et sortie de message. Dans la représentation graphique, les messages sont représentés soit par des flèches horizontales dont l'une est brisée pour indiquer un dépassement, soit par des flèches se croisant, inclinées vers le bas. Voir Figure 30.

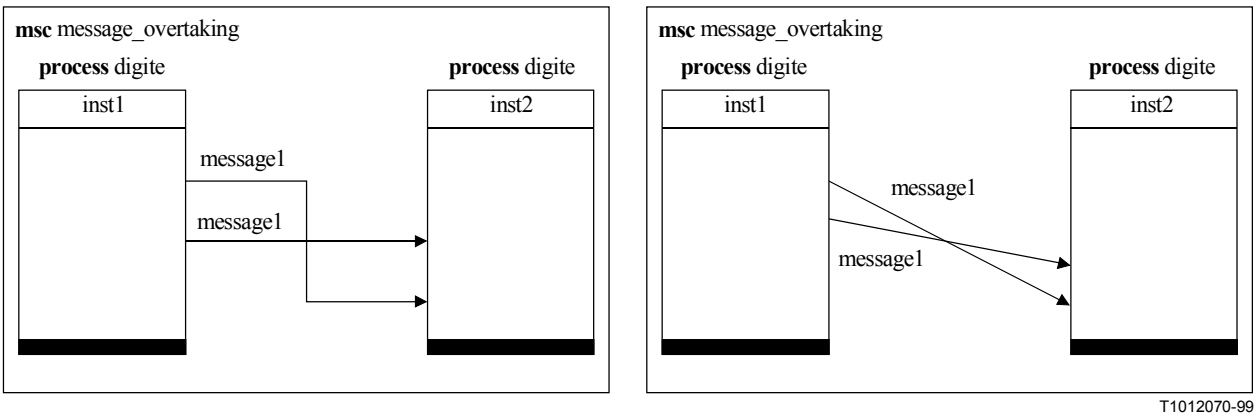


Figure 30/Z.120 – Dépassement de message

```
msc message_overtaking;
inst inst1;
inst inst2;
        inst1: instance process digite;
            out message1, a to inst2;
```

```
            out message1, b to inst2;
        endinstance;
        inst2: instance process digite;
            in message1, b from inst1;
            in message1, a from inst1;
        endinstance;
endmsc;
```

9.3 Concepts de base de diagramme MSC

Cet exemple contient les constructions MSC de base: instances, environnement, messages, conditions, actions et expiration de temporisateur. Dans la représentation graphique, les deux types de symboles d'instance sont utilisés: la forme ligne simple et la forme colonne. Voir Figure 31.

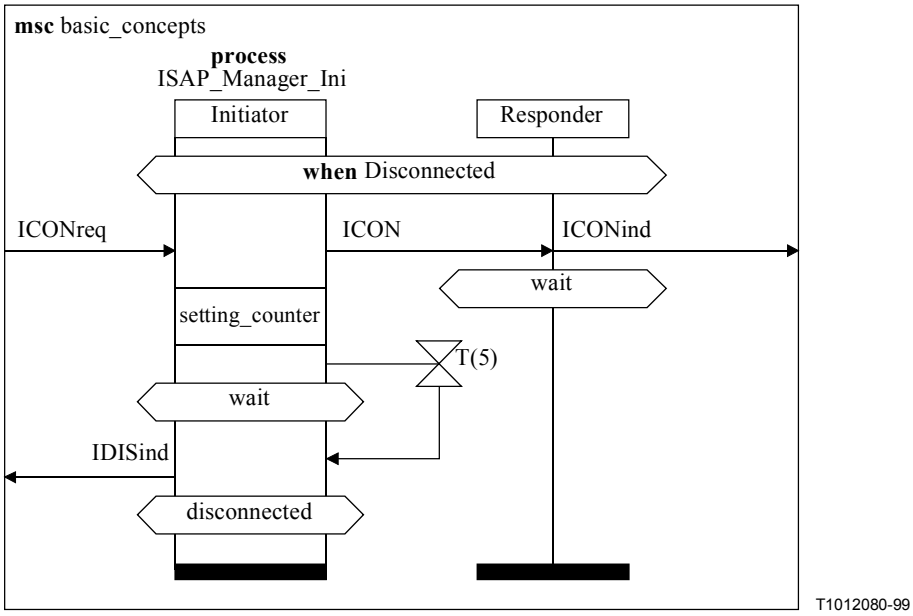


Figure 31/Z.120 – Concepts de base MSC

Syntaxe textuelle orientée instance

```
msc basic_concepts;
inst Initiator: process ISAP_Manager_Ini;
inst Responder;
gate out ICONreq to Initiator;
gate in ICONind from Responder;
gate in IDISind from Initiator;

        Initiator: instance process ISAP_Manager_Ini;
            condition when Disconnected shared all;
            in ICONreq from env;
            out ICON to Responder;
            action "setting_counter";
            starttimer T (5);
            condition wait shared;
            timeout T;
            out IDISind to env;
            condition disconnected shared;
        endinstance;

        Responder: instance;
            condition when Disconnected shared all;
            in ICON from Initiator;
            out ICONind to env;
            condition wait shared;
        endinstance;
endmsc;
```

Syntaxe textuelle orientée événement

```
msc basic_concepts;
inst Initiator: process ISAP_Manager_Ini;
inst Responder;
gate out ICONreq to Initiator;
gate in ICONind from Responder;
gate in IDISind from Initiator;

Initiator: instance process ISAP_Manager_Ini;
Responder:    instance;
all:          condition when Disconnected;
Initiator:    in ICONreq from env;
              out ICON to Responder;
Responder:    in ICON from Initiator;
              out ICONind to env;
              condition wait;
              endinstance;
Initiator:    action "setting_counter";
              starttimer T (5);
              condition wait;
              timeout T;
              out IDISind to env;
              condition disconnected;
              endinstance;
endmsc;
```

9.4 Composition de diagramme MSC par étiquetage de conditions

Cet exemple illustre la composition de diagrammes MSC au moyen de conditions globales. La condition globale finale 'Wait_For_Resp' du diagramme MSC 'connection_request' est identique à la condition globale initiale du diagramme MSC connection confirm. Par conséquent, les deux diagrammes MSC peuvent être composés pour donner le diagramme MSC résultant 'connection' (voir exemple au 9.5).

La composition est définie au moyen du diagramme HMSC 'con_setup' de la Figure 34.

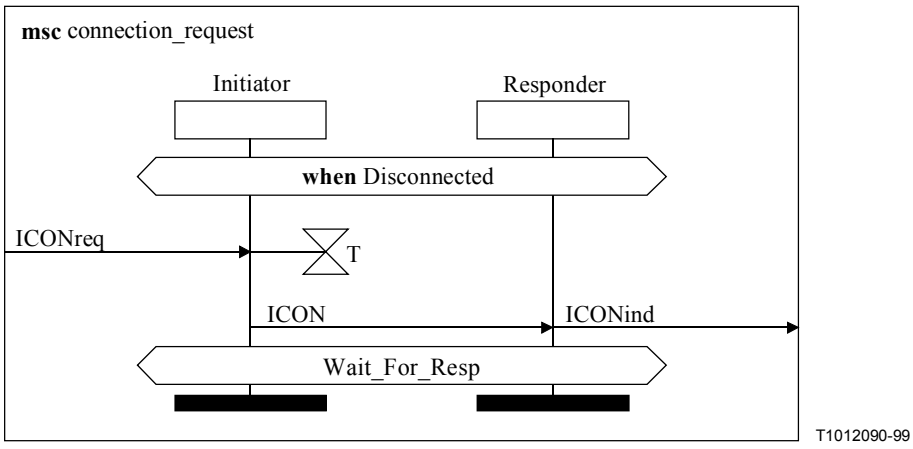


Figure 32/Z.120 – Diagramme MSC connection_request

```
msc connection_request;
inst Initiator;
inst Responder;
gate out ICONreq to Initiator;
gate in ICONind from Responder;

        instance Initiator;
              condition when Disconnected shared all;
              in ICONreq from env;
```

```
            starttimer T;
            out ICON to Responder;
            condition Wait_For_Resp shared all;
        endinstance;
        instance Responder;
            condition when Disconnected shared all;
            in ICON from Initiator;
            out ICONind to env;
            condition Wait_For_Resp shared all;
        endinstance;
endmsc;
```

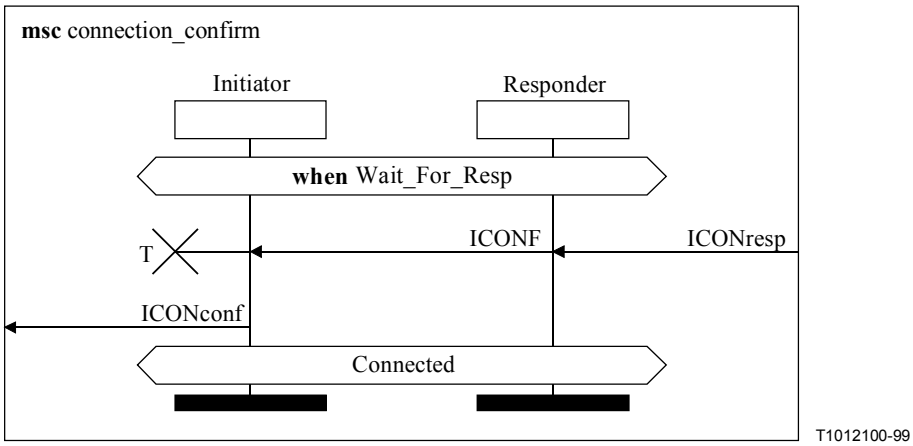


Figure 33/Z.120 – Diagramme MSC connection_confirm

```
msc connection_confirm;
inst Initiator;
inst Responder;
gate in ICONconf from Initiator;
gate out ICONresp to Responder;

        instance Initiator;
            condition when Wait_For_Resp shared all;
            in ICONF from Responder;
            stoptimer T;
            out ICONconf to env;
            condition Connected shared all;
        endinstance;
        instance Responder;
            condition when Wait_For_Resp shared all;
            in ICONresp from env;
            out ICONF to Initiator;
            condition Connected shared all;
        endinstance;
endmsc;
```

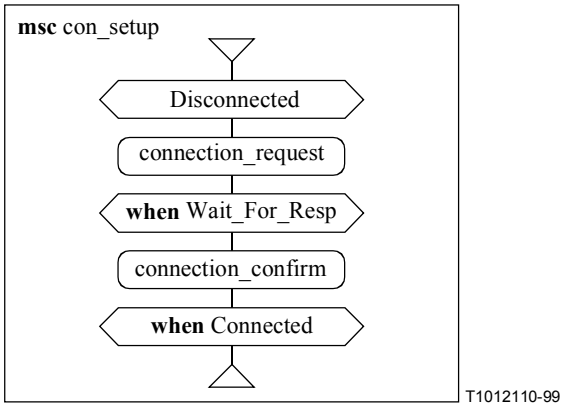


Figure 34/Z.120 – Diagramme MSC con_setup

```
msc con_setup;
expr L1;
        L1: condition Disconnected seq ( L2 );
        L2: (connection_request) seq (L3);
        L3: condition when Wait_For_Resp seq ( L4);
        L4: (connection_confirm) seq ( L5 );
        L5: condition when Connected seq ( L6 );
        L6: end;
endmsc;
```

9.5 Diagramme MSC avec supervision temporelle

Dans cet exemple, le diagramme MSC 'connection' contient un arrêt de temporisateur. Voir Figure 35.

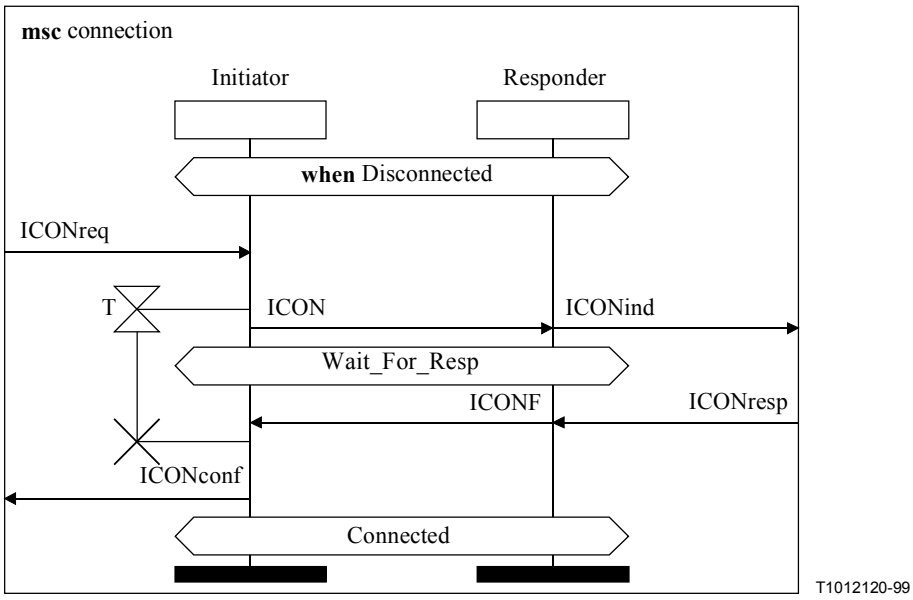


Figure 35/Z.120 – Diagramme MSC connection

```
msc connection;
inst Initiator;
inst Responder;
gate out ICONreq to Initiator;
gate in ICONind from Responder;
gate out ICONresp to Responder;
gate in ICONinf from Initiator;
```

```
        Initiator: instance;
            condition when Disconnected shared all;
            in ICONreq from env;
            starttimer T;
            out ICON to Responder;
            condition Wait_For_Resp shared all;
            in ICONF from Responder;
            stoptimer T;
            out ICONconf to env;
            condition Connected shared all;
        endinstance;
        Responder: instance;
            condition when Disconnected shared all;
            in ICON from Initiator;
            out ICONind to env;
            condition Wait_For_Resp shared all;
            in ICONresp from env;
            out ICONF to Initiator;
            condition Connected shared all;
        endinstance;
endmsc;
```

9.6 Diagramme MSC avec perte de message

Dans cet exemple, le diagramme MSC 'failure' contient une expiration de temporisateur due à la perte d'un message. Voir Figure 36.

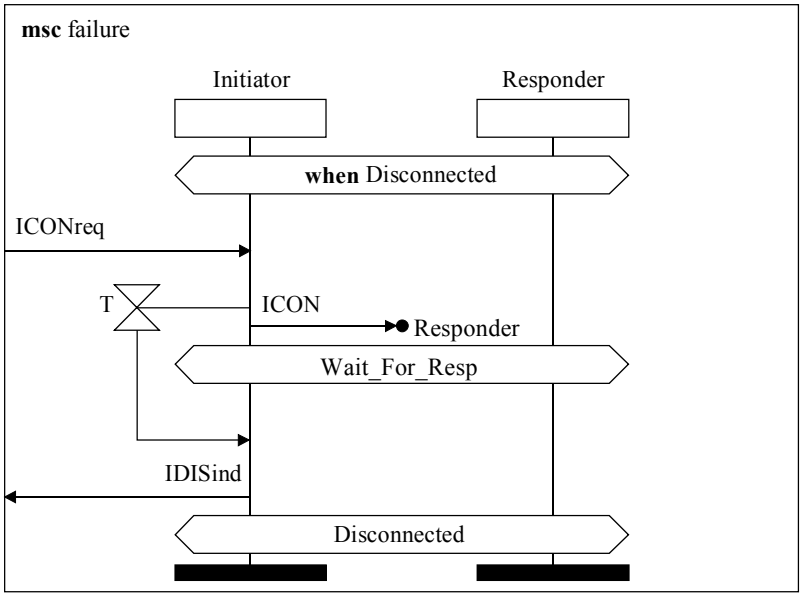


Figure 36/Z.120 – Diagramme MSC failure

```
msc failure;
inst Initiator;
inst Responder;
gate out ICONreq to Initiator;
gate in IDISind from Initiator;

        Initiator: instance;
            condition when Disconnected shared all;
            in ICONreq from env;
            starttimer T;
            out ICON to lost Responder;
            condition Wait_For_Resp shared all;
            timeout T;
            out IDISinf to env;
            condition Disconnected shared all;
```

```
        endinstance;
        Responder: instance;
            condition when Disconnected shared all;
            in ICON from Initiator;
            condition Wait_For_Resp shared all;
            condition Disconnected shared all;
        endinstance;
endmsc;
```

9.7 Conditions locales

Dans cet exemple, les conditions locales d'une instance sont utilisées pour indiquer les états locaux de cette instance. Voir Figure 37.

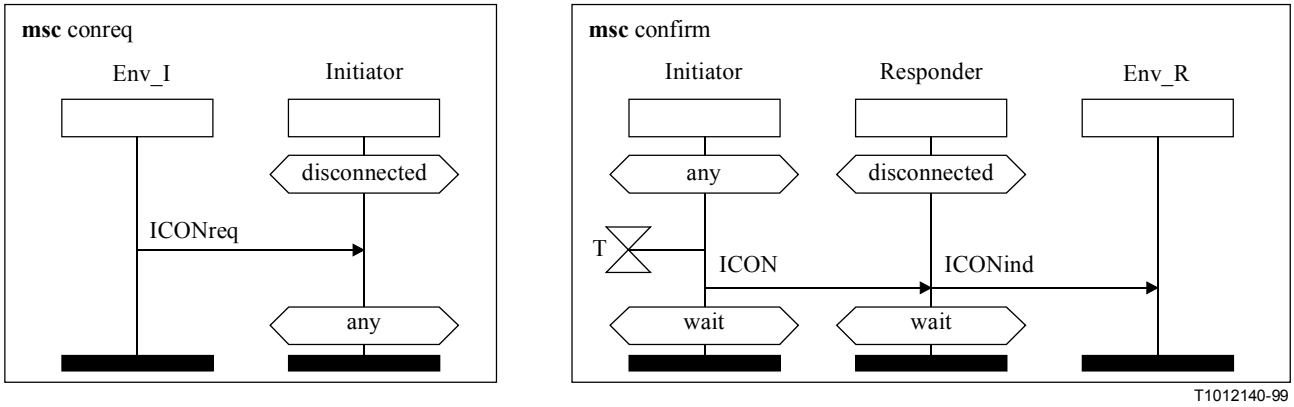


Figure 37/Z.120 – Conditions locales

```
msc conreq;
inst Env_I;
inst Initiator;
        Env_I: instance;
            out ICONreq to Initiator;
        endinstance;
        Initiator: instance;
            condition disconnected shared;
            in ICONreq from Env_I;
            condition any shared;
        endinstance;
endmsc;

msc confirm;
inst Initiator;
inst Responder, Env_R;
        Initiator: instance;
            condition any shared;
            starttimer T;
            out ICON to Responder;
            condition wait shared;
        endinstance;
        Responder: instance;
            condition disconnected shared;
            in ICON from Initiator;
            out ICONind to Env_R;
            condition wait shared;
        endinstance;
        Env_R: instance;
            in ICONind from Responder;
        endinstance;
endmsc;
```

10 Données

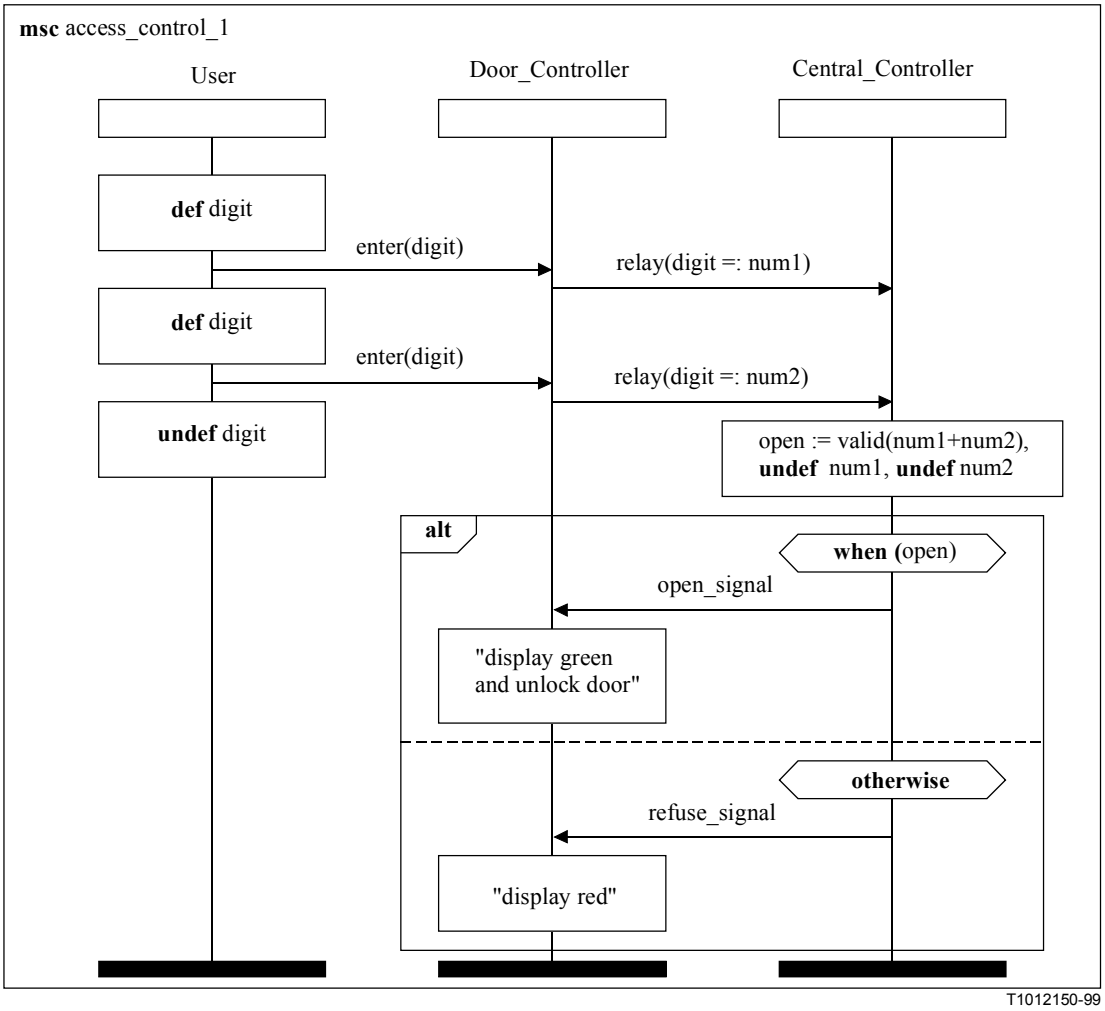


Figure 38/Z.120 – Données en cadre d'action, messages et conditions

Dans la Figure 38, l'instance *User* est censée posséder la variable *digit* et l'instance *Central_Controller* possède les variables *num1* et *num2*. La déclaration de ces variables interviendra dans le document MSC englobant qui est représenté dans la Figure 39 ci-après.

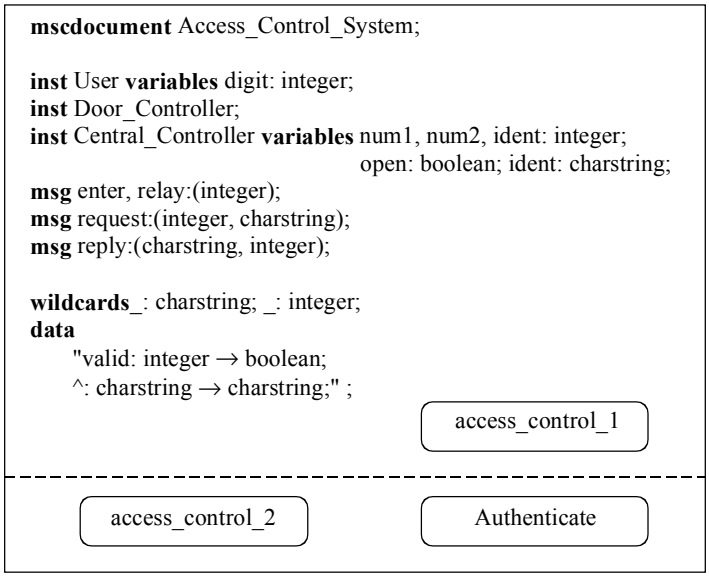


Figure 39/Z.120 – Document MSC pour Access_Control_System

Le premier cadre d'action pour l'instance *User* définit la variable *digit* dont l'effet est d'attribuer une valeur non spécifiée à cette variable. Appelons cette valeur *v1*. Cette valeur est ensuite transmise à l'instance *Door_Controller* par l'intermédiaire d'un seul paramètre du message *enter*. Le paramètre *digit* apparaît sous la forme d'une expression: ce n'est pas une association parce qu'il n'y a aucun symbole de lien et les exigences statiques interdisent qu'il soit une structure car *enter* est un message complété, c'est-à-dire que son origine et sa destination sont des instances. Cette valeur *v1* de la variable *digit* est donc disponible pour l'instance *Door_Controller* jusqu'à ce qu'une nouvelle valeur de la variable *digit* soit reçue, sans tenir compte de ce qui se passe pour la valeur de *digit* dans son instance possédante.

Dans le premier message *relay*, la valeur *v1* de la variable *digit* est transmise sous forme de paramètre au moyen de la partie expression de l'association *digit* =: *num1*. Cette association a pour conséquence que la valeur *v1* se lie à la valeur *num1* dans l'instance *Central_Controller*.

Le deuxième cadre d'action pour l'instance *User* redéfinit la valeur de la variable *digit* afin de passer à une autre valeur non spécifiée, par exemple *v2*. La sémantique permet que les valeurs *v1* et *v2* soient les mêmes. Cette deuxième valeur est envoyée à l'instance *Door_Controller* au moyen du deuxième message *enter*. Noter que les deux valeurs de la variable *digit* peuvent coexister dans une trace passant par le diagramme MSC bien qu'une seule valeur soit connue par une instance à un instant donné.

Dans le deuxième message *relay*, la valeur *v2* de la variable *digit* devient liée à la variable *num2* de l'instance *Central_Controller*. La valeur de *num1* est donc *v1* et celle de *num2* est *v2*.

L'action finale sur l'instance *User* supprime la définition de la variable *digit*, ce qui signifie que celle-ci devient une variable non liée de l'instance *User*. Toute éventuelle référence subséquente à la variable *digit* sera donc illégale dans cette instance, ce qui pourrait se produire si un autre diagramme MSC est placé en séquence après *access_control_1*. La suppression explicite de la définition d'une variable supprime celle-ci du domaine de visibilité jusqu'à ce qu'elle soit liée par une association ou par une déclaration de type def.

Le cadre d'action relatif à l'instance *Central_Controller* contient trois déclarations, qui sont évaluées en parallèle. Les expressions éventuelles sont d'abord évaluées au moyen de l'état précédent. Donc l'expression *valid(num1 + num2)* est évaluée au moyen de la valeur *v1* comme étant la valeur de *num1* et la valeur *v2* est évaluée comme étant la valeur de *num2*. Après l'évaluation d'éventuelles expressions, l'état est mis à jour conformément aux associations, (à la déclaration de définition s'il

est présent) et aux déclarations de suppression de définition. Dans ce cas, la variable *open* est liée au résultat de l'expression *valid(num1 + num2*) dans laquelle la fonction booléenne *valid* est déclarée/définie dans la section "données" du document MSC englobant (Figure 39), tandis que les variables *num1* et *num2* deviennent non liées, c'est-à-dire hors du domaine de visibilité. Comme les déclarations sont évaluées en parallèle, ils peuvent être indiqués dans un ordre quelconque sans altérer l'interprétation. Les déclarations de suppression de définition auraient donc pu être indiquées avant la déclaration liant la variable *open*.

La valeur de la variable *open* est utilisée comme garde dans l'expression d'alternative en ligne. La garde répond aux exigences statiques en ce sens que les variables de garde (ici seulement *open*) sont détenues par l'instance active, qui est *Central_Controller*. La variante choisie dépend de la valeur de la variable *open*: si celle-ci est "Vrai", c'est la première variante qui est choisie, sinon c'est la seconde.

Noter que les cadres d'action apparaissant dans l'expression en ligne contiennent un texte informel, comme cela est indiqué par les caractères de citation qui encadrent ce texte.

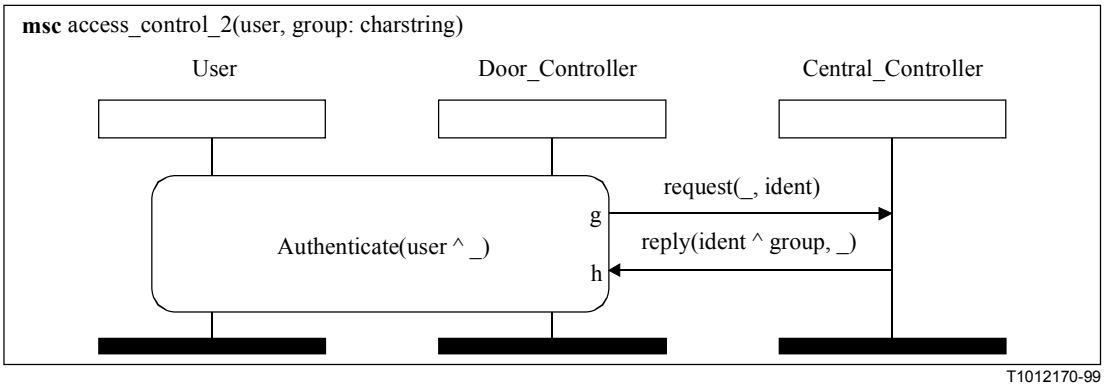


Figure 40/Z.120 – Données statiques et données de message incomplet

La Figure 40 montre le diagramme MSC *access_control_2* qui possède deux paramètres formels et statiques: *user* et *group*, l'un et l'autre du type *charstring*. Si le diagramme *access_control_2* est cité en référence dans un autre diagramme MSC, deux paramètres réels doivent être indiqués pour ces paramètres formels. La référence MSC *Authenticate* possède un seul paramètre réel, l'expression *user* ^ _, dans laquelle "^" représente une concaténation de chaîne (définie/déclarée dans la section "données" du document MSC englobant sur la Figure 39) et dans laquelle "_" représente une chaîne de caractères (*charstring*) générique (déclarée dans le document MSC englobant sur la Figure 39). Le diagramme MSC *access_control_2* utilise également une variable dynamique *ident*, qui est possédée par l'instance *Central_Controller*.

L'expression paramétrique réelle qui est attribuée à la référence *Authenticate* ne peut pas contenir de variables dynamiques mais peut contenir des caractères génériques et des variables statiques, comme c'est le cas ici. L'expression *user* ^ _ désigne toute chaîne qui commence par la valeur attribuée à la variable *user*.

Le message *request* est un message incomplet qui possède deux paramètres. Il est incomplet parce qu'il provient de la porte *g*. Les exigences statiques pour un message incomplet aboutissant à une instance nous indiquent que les paramètres ne doivent se composer que de structures. Le premier paramètre "_" est donc un caractère générique qui ne donne lieu à aucune association. Le deuxième paramètre est la structure *ident* qui finira par être liée dynamiquement à toute valeur envoyée par la référence MSC *Authenticate*. On notera que seules les variables dynamiques possédées par l'instance *Central_Controller* ou des caractères génériques peuvent apparaître en tant que structures dans les messages reçus par cette instance.

Le message *reply* est aussi un message incomplet qui possède deux paramètres. Mais cette fois il est incomplet parce qu'il about à une porte (ici *h*). Les exigences statiques nous indiquent que les paramètres doivent consister seulement en expressions, de sorte que le premier paramètre est l'expression *ident ^ group* et que le second est le caractère générique "_". Noter que les expressions paramétriques peuvent contenir des variables statiques (ici *ident*), des variables dynamiques (ici *group*) et des caractères génériques.

La valeur donnée à l'expression *ident ^ group* sera une chaîne dont la première partie consistera en tout ce qui a été envoyé dans le second paramètre du message *request* et dont la deuxième partie consistera en toute valeur fournie pour la variable statique *group* dans une référence au diagramme MSC *access_control_2*. Le deuxième paramètre du message *reply* peut être une chaîne quelconque. Les valeurs de ces deux paramètres sont envoyées avec le message *reply* et seront liées dynamiquement aux structures trouvées dans la liste correspondante de paramètres de réception de réponse se trouvant à l'intérieur de la référence MSC *Authenticate*.

11 Temps

Les exemples donnés dans le présent sous-paragraphe montrent les principales caractéristiques des concepts de temps dans les diagrammes MSC. Ces exemples sont extraits d'une spécification de test de qualité de fonctionnement pour un serveur de session d'accès en architecture TINA (*telecommunication information networking architecture*). Les diagrammes MSC comportent une composante de test (*TC, test component*) et un système sous test (*SUT, system under test*). Le test couvre trois étapes: l'obtention d'un accès nommé à un serveur de session d'accès, le réglage du contexte d'utilisateur pour un usager et la fourniture à celui-ci d'un service sélectionné. Ces étapes sont définies sous le nom de diagrammes MSC utilitaires (de service).

Les diagrammes MSC définisseurs couvrent la spécification comportementale y compris les contraintes temporelles (diagramme MSC *Black_Box_Behavior*), la spécification de mesurage (diagramme MSC *Blackballed_Measurement_Scenario*) et les traces d'exécution de test (diagramme MSC *Black_Box_Test_Execution*).

Les mesures relevées par la composante de test font appel aux variables temporelles *rel1*, ... , *abs2*, qui sont déclarées en tant que variables dynamiques de type temps de l'instance *TC*.

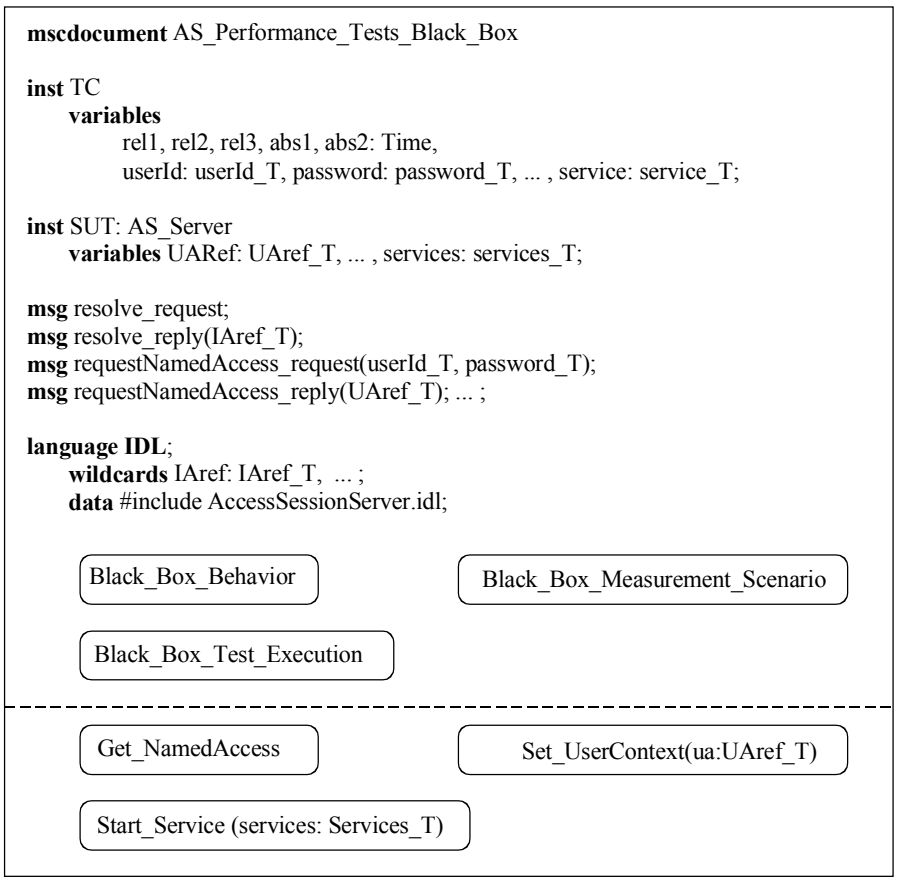


Figure 41/Z.120 – Document MSC et déclaration de variables temporelles

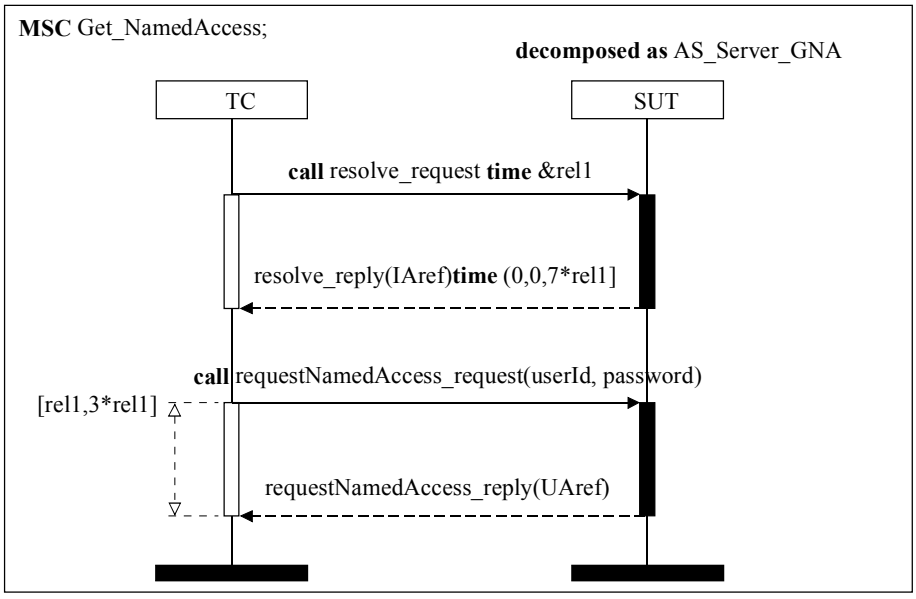


Figure 42/Z.120 – Contraintes temporelles et mesures

Le diagramme MSC utilitaire représenté sur la Figure 42 fait appel à une mesure de temps relatif pour observer la durée du message de la communication resolve_request. La variable de temps *rel1*

contiendra la durée écoulée entre la sortie de la communication par *TC* et son entrée par *SUT*. La variable *rel1* est liée lorsque l'entrée de la communication a lieu.

La mesure de la durée de la communication est utilisée par la suite pour limiter la durée de message pour *resolve_reply*. La contrainte de temps relatif (*0,0,7*rel1]* permet au message de prendre 70% au plus du temps qu'il a pris pour envoyer la communication de demande resolve_request de la composante *TC* au système *SUT*.

Par ailleurs, la mesure de la durée de la communication est utilisée pour contraindre l'exécution de l'instance *TC*: la contrainte de temps relatif (*rel1,3*rel1]* nécessite qu'après la sortie de la communication requestNamedAccess, il faille au moins le temps *rel1* et au plus le temps *3*3rel1* pour obtenir la réponse.

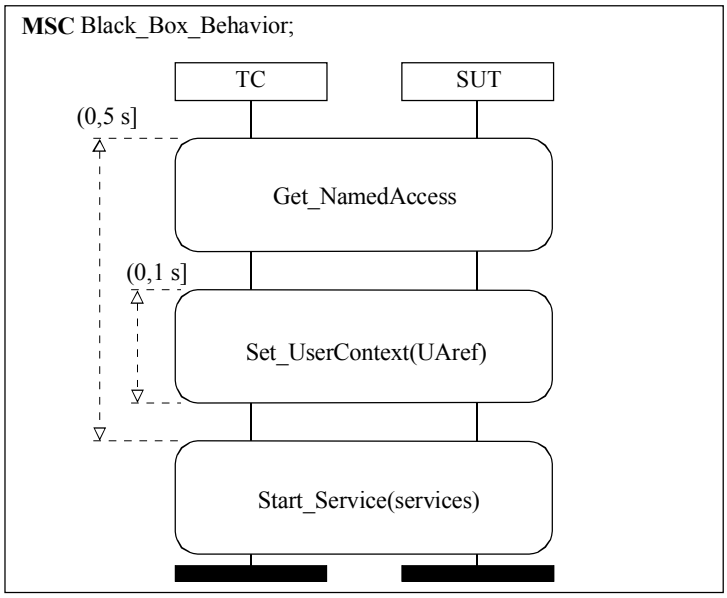


Figure 43/Z.120 – Contraintes temporelles pour références MSC

La Figure 43 donne un exemple d'utilisation des contraintes de temps pour références MSC: la contrainte de temps relatif *(0,1 s]* spécifie qu'il faut au plus *1 s* pour obtenir l'exécution du diagramme *Set_UserContext*, c'est-à-dire qu'il s'écoulera au plus *1 s* entre l'événement de début et l'événement de fin du diagramme *Set_UserContext.*

La contrainte de temps relatif *(0,5 s]* associe le début du diagramme *Get_NamedAccess* au début du diagramme *Start_Service*. Elle spécifie qu'il faut au plus *5 s* pour qu'il soit possible de lancer un service à partir du moment où la procédure d'obtention d'accès nommé a été lancée. Cette contrainte a également une incidence sur l'exécution du diagramme *Set_UserContext*.

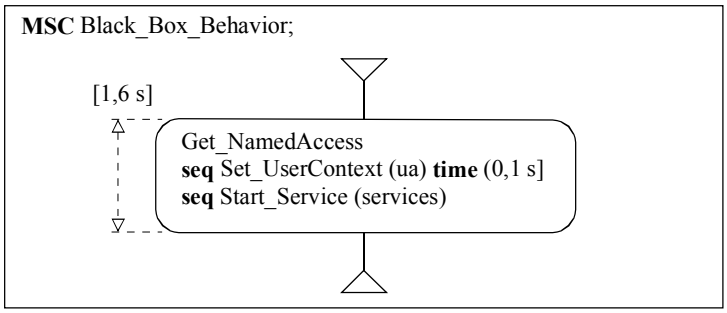


Figure 44/Z.120 – Contraintes temporelles pour diagrammes HMSC

La Figure 44 montre l'utilisation de contraintes temporelles pour les diagrammes HMSC. Elle est sémantiquement similaire à la Figure 43, sauf que dans cette figure la contrainte de temps relatif *[1,6 s]* va du début du diagramme *Get_NamedAccess* jusqu'à la fin du diagramme *Start_Service*, c'est-à-dire que le lancement d'un service nécessite au moins *1 s* et au plus *6 s.*

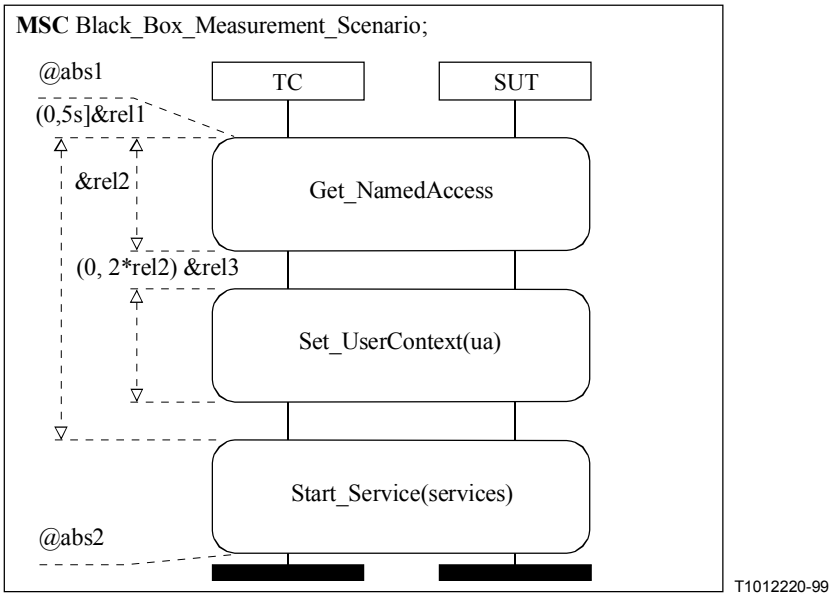


Figure 45/Z.120 – Observation du temps par les mesures

La séquence *Black_Box_Measurement_Scenario* de la Figure 45 utilise des mesures absolues (indiquées par le caractère @) et des mesures relatives (indiquées par le caractère &). Les mesures absolues suivent la valeur de l'horloge mondiale.

Les mesures absolues *@abs1* et *@abs2* indiquent les valeurs absolues du début et de la fin de l'essai décrit en tant que séquence des diagrammes *Get_NamedAccess, Set_UserContext* et *Start_Service*. En outre, les mesures relatives sont utilisées pour mesurer la distance temporelle entre des paires d'événements.

Par exemple, la mesure *&rel1* indique la distance entre l'événement de début de *Get_NamedAccess* et l'événement de début de *Start_Service*. Cette mesure est combinée avec une contrainte *(0,5 s]* indiquant que la durée entre le début de *Get_NamedAccess* et celui de *Start_Service* est contrainte. Le temps réellement nécessaire (dans les limites indiquées de la contrainte, est observé au moyen de la mesure relative.

*(0, 2*rel2) &rel3* est également une contrainte temporelle relative qui est combinée à une mesure relative. Cette contrainte se rapporte à une mesure déjà prise de la durée écoulée entre le début et la fin de *Get_NamedAccess*.

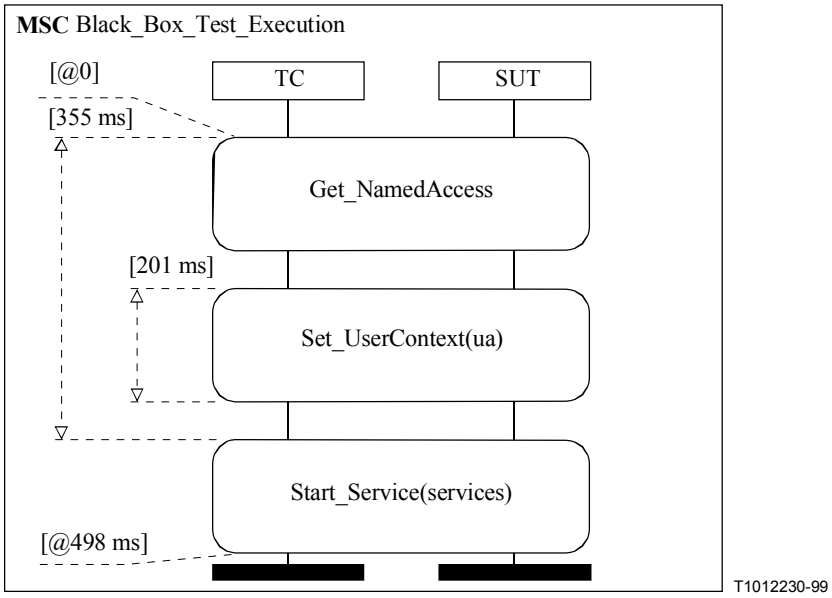


Figure 46/Z.120 – Temps singulier utilisé pour les traces de système

La Figure 46 montre la représentation des traces de système (extraites par exemple d'exécutions, de simulations ou d'essais dans le système) au moyen de points temporels absolus et relatifs. Les points temporels absolus sont indiqués par le caractère @. L'expression *[@0]* représente le fait que la séquence *Get_NamedAccess* a été lancée au temps d'horloge mondiale 0 (c'est-à-dire que l'horloge mondiale a été réarmée). L'exécution de la séquence *Get_NamedAccess, Set_UserContext* et *Start_Service* a pris *498 ms*. L'exécution de la séquence *Set_UserContext* a pris *201 ms*. L'événement de début de la séquence *Start_Service* s'est produit *355 ms* après l'événement de début de la séquence *Get_NamedAccess*.

12 Processus de création et de terminaison

Cet exemple montre la création dynamique de l'instance 'subscriber' due à une demande de connexion et la terminaison correspondante, due à une demande de déconnexion. Voir Figure 47.

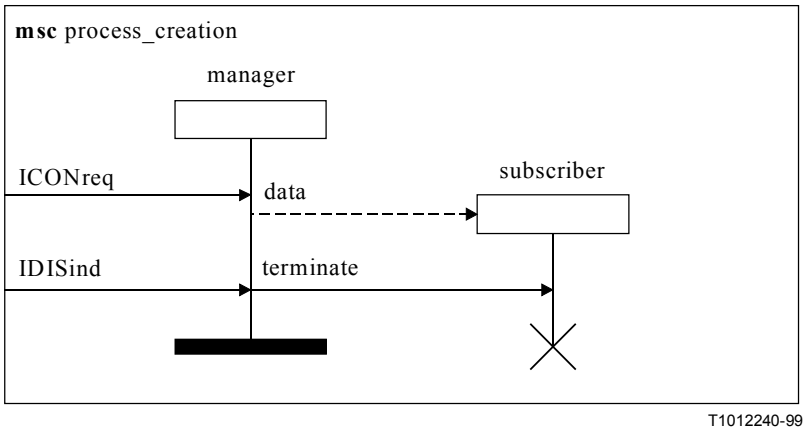


Figure 47/Z.120 – Diagramme MSC process_creation

```
msc process_creation;
inst manager;
inst subscriber;
gate out ICONreq to manager;
gate out IDISind to manager;

        manager: instance;
            in ICONreq from env;
            create subscriber(data);
            in IDISind from env;
            out terminate to subscriber;
        endinstance;
        subscriber: instance;
            in terminate from manager;
            stop;
        endinstance;
endmsc;
```

13 Corégion

Cet exemple montre une région concurrente qui doit indiquer que la consommation du message 'process_data1' et celle du message 'process_data2' ne sont pas ordonnées dans le temps, c'est-à-dire que le message 'process_data1' peut être consommé avant 'process_data2' ou inversement. Voir Figure 48.

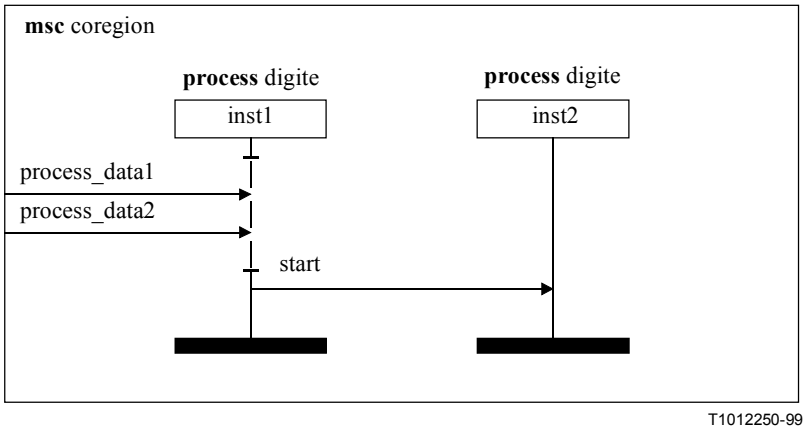


Figure 48/Z.120 – Corégion MSC

```
msc coregion;
inst inst1;
inst inst2;
gate out process_data1 to inst1;
gate out process_data2 to inst1;

        inst1: instance process digite;
            concurrent;
                in process_data1 from env;
                in process_data2 from env;
            endconcurrent;
            out start to inst2;
        endinstance;
        inst2: instance process digite;
            in start from inst1;
        endinstance;
endmsc;
```

14 Relation d'ordre général

14.1 Relation d'ordre généralisée dans une corégion

Cet exemple montre une relation d'ordre généralisée à l'intérieur d'une corégion au moyen de 'connexions', c'est-à-dire que l'ordre est décrit à l'aide de connexions reliant les événements dans la corégion. Dans le diagramme MSC 'connectivity' l'ordre suivant est défini: ICONreq < ICONind < ICONresp, ICONreq < IDISind.

Dans ce cas, IDISind n'est pas ordonné par rapport à ICONind et à ICONresp. Voir Figure 49.

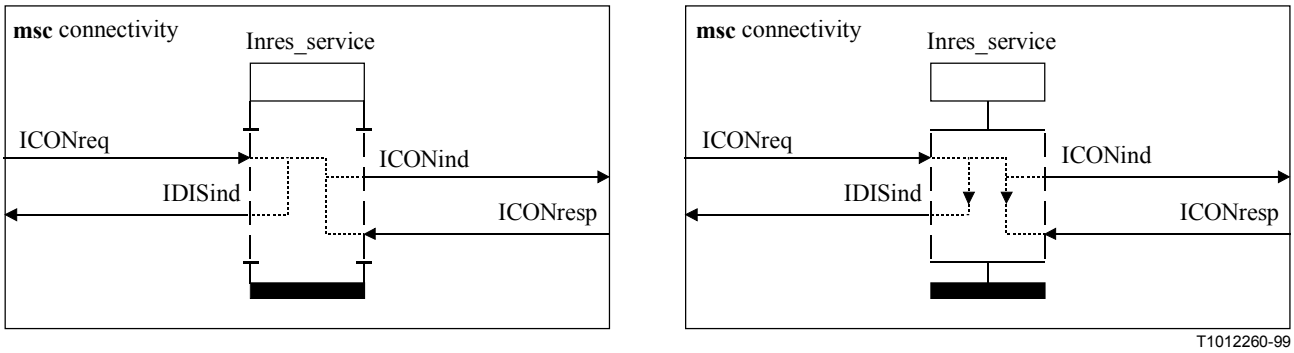


Figure 49/Z.120 – Connexité MSC dans deux variantes graphiques

```
msc connectivity;
inst Inres_service;
gate out ICONreq to Inres_service;
gate in ICONind from Inres_service;
gate in IDISind from Inres_service;
gate out ICONresp to Inres_service;

        Inres_service: instance;
            concurrent;
                in ICONreq from env before Label1, Label2;
                label Label1; out ICONind to env before Label3;
                label Label2; out IDISind to env;
                label Label3; in ICONresp from env;
            endconcurrent;
        endinstance;
endmsc;
```

14.2 Relation d'ordre généralisé entre différentes instances

Cet exemple montre l'utilisation de constructions de synchronisation issues de diagrammes de séquence du temps pour décrire l'ordre généralisé entre différentes instances. La ligne (brisée ou oblique) entre le message entrant 'ICONreq' et le message sortant 'ICONind' définit l'ordre suivant: ICONreq < ICONind. Voir Figure 50.

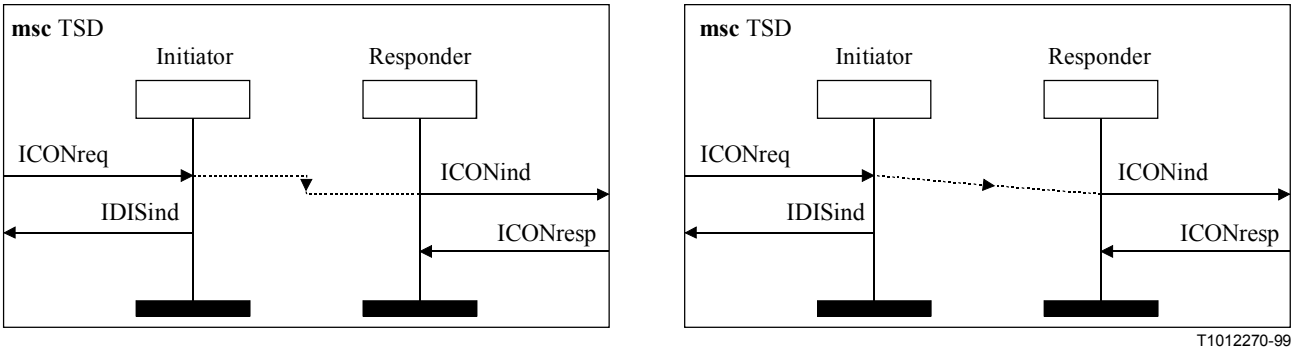


Figure 50/Z.120 – Diagrammes MSC de séquence temporelle dans deux variantes graphiques

```
msc TSD;
inst Initiator;
inst Responder;
gate out ICONreq to Initiator;
gate in ICONind from Responder;
gate in IDISind from Initiator;
gate out ICONresp to Responder;

        instance Initiator;
            in ICONreq from env before Label1;
            out IDISind to env;
        endinstance;
        instance Responder;
            label Label1; out ICONind to env;
            in ICONresp from env;
        endinstance;
endmsc;
```

15 Expressions en ligne

15.1 Expression en ligne avec composition en alternative

Dans cet exemple, le cas d'une connexion réussie est combiné avec celui d'un échec dans le même diagramme MSC à l'aide de l'expression en ligne utilisant l'opérateur de l'alternative (MSC 'alternative'). Dans le diagramme MSC 'exception' la même situation est décrite à l'aide de l'expression en ligne équivalente, utilisant l'opérateur de l'exception. Voir Figure 51.

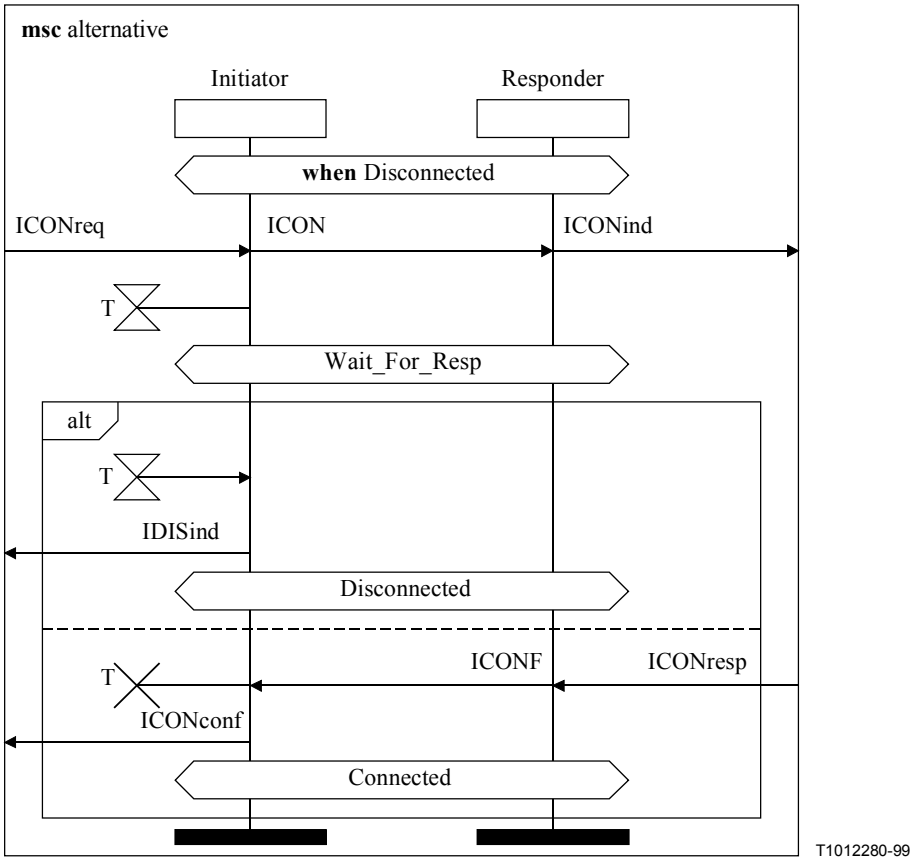


Figure 51/Z.120 – Expression en ligne avec alternative

```
msc alternative;
inst Initiator;
inst Responder;
gate out ICONreq to Initiator;
gate in ICONind from Responder;
gate in IDISind from Initiator;
gate out ICONresp to Responder;
gate in ICONconf from Initiator;

        Initiator:      instance;
        Responder:      instance;
        all:            condition when Disconnected;
        Initiator:      in ICONreq from env;
                        out ICON to Responder;
                        starttimer T;
        Responder:      in ICON from Initiator;
                        out ICONind to env;
        all:            condition Wait_for_Resp;
                        alt begin;
            Initiator:          timeout T;
                                out IDISind to env;
            all:                condition Disconnected;
                        alt;
            Responder:          in ICONresp from env;
                                out ICONF to Initiator;
            Initiator:          in ICONF from Responder;
                                stoptimer T;
                                out ICONconf to env;
            all:                condition Connected;
                        alt end;
        Initiator:      endinstance;
        Responder:      endinstance;
endmsc;
```

L'opérateur **exc** est assimilable à une alternative dans laquelle le second opérande est le reste entier du diagramme MSC comme le montre l'exemple suivant: (le diagramme MSC 'alternative' a exactement la même signification que le diagramme MSC 'exception'). Voir Figure 52.

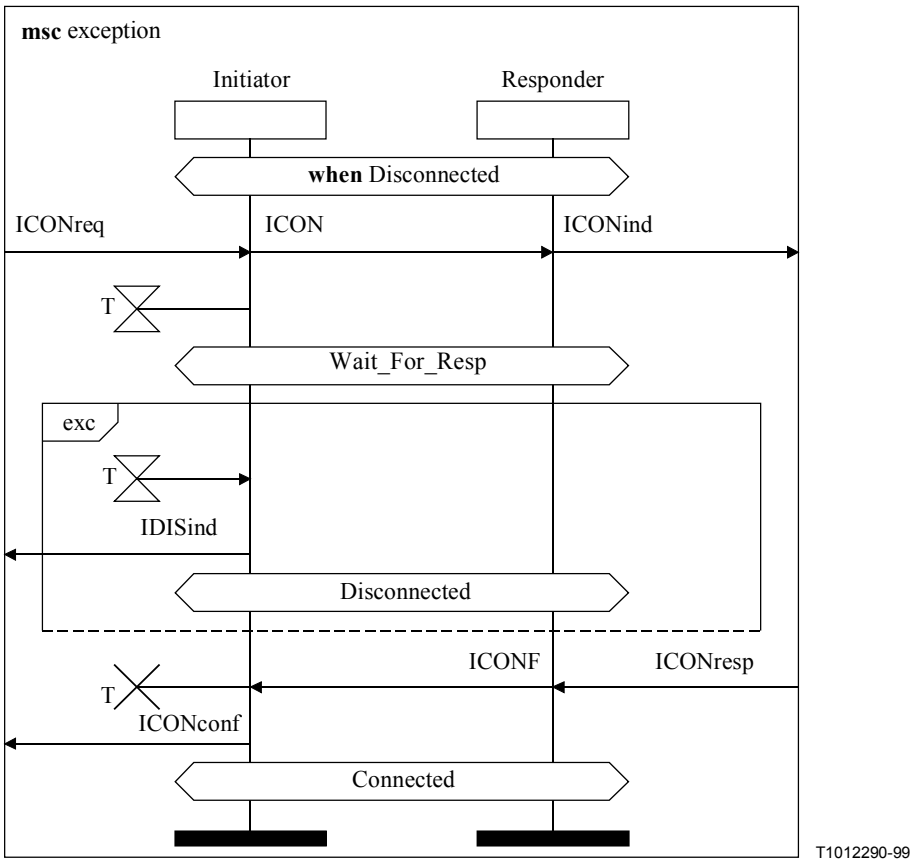


Figure 52/Z.120 – Expression en ligne avec exception

```
msc exception;
inst Initiator;
inst Responder;
gate out ICONreq to Initiator;
gate in ICONind from Responder;
gate in IDISind from Initiator;
gate out ICONresp to Responder;
gate in ICONconf from Initiator;

        Initiator:      instance;
        Responder:      instance;
        all:            condition when Disconnected;
        Initiator:      in ICONreq from env;
                        out ICON to Responder;
                        starttimer T;
        Responder:      in ICON from Initiator;
                        out ICONind to env;
        all:            condition Wait_for_Resp;
                        exc begin;
            Initiator:      timeout T;
                            out IDISind to env;
            all:            condition Disconnected;
                        exc end;
        Responder:      in ICONresp from env;
                        out ICONF to Initiator;
        Initiator:      in ICONF from Responder;
                        stoptimer T;
                        out ICONconf to env;
```

```
        all:            condition Connected;
        Initiator:      endinstance;
        Responder:      endinstance;
endmsc;
```

15.2 Expression en ligne avec portes

Cet exemple décrit le scénario de l'exemple du 15.1 de façon légèrement différente: le message 'ICONF' issu de l'instance 'Responder' est connecté par les portes à l'expression en ligne avec alternative attachée à l'instance 'Initiator'. Le message 'ICONF' issu de l'instance 'Responder' est propagé par la porte g1 (sur les deux branches de l'alternative) vers l'instance 'Initiator'. Cela décrit le fait que l'instance 'Initiator' est en attente d'une réponse de la part de l'instance 'Responder'. Deux cas sont combinés dans le diagramme MSC 'very_advanced':

a) l'instance 'Responder' ne répond pas à temps: le message entrant 'ICONF' est ignoré après expiration du temporisateur et déconnexion de l'instance 'Initiator';

b) l'instance 'Responder' répond à temps, ce qui conduit à une connexion réussie.

Voir Figure 53.

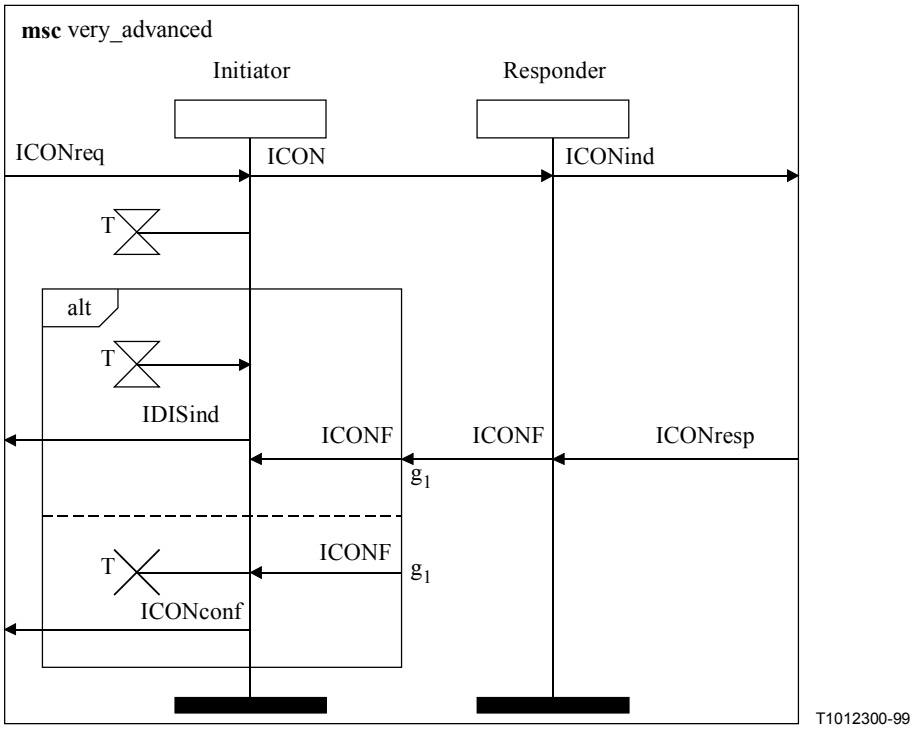


Figure 53/Z.120 – Expression en ligne avec portes

```
msc very_advanced;
inst Initiator;
inst Responder;

gate out ICONreq to Initiator;
gate in ICONind from Responder;
gate in IDISind from Initiator;
gate out ICONresp to Responder;
gate in ICONconf from Initiator;
        Initiator:      instance;
        Responder:      instance;
        Initiator:      in ICONreq from env;
                        out ICON to Responder;
                        starttimer T;
```

```
        Responder:      in ICON from Initiator;
                        out ICONind to env;
                        in ICONresp from env;
                        out ICONF to inline altref via g1;
        Initiator:      alt begin altref;
                            gate g1 out ICONF to Initiator
                            external in ICONF from Responder;
                            timeout T;
                            out IDISind to env;
                            in ICONF from env via g1;
                        alt;
                            gate g1 out ICONF to Initiator;
                            in ICONF from env via g1;
                            stoptimer T;
                            out ICONconf to env;
                        alt end;
        Initiator:      endinstance;
        Responder:      endinstance;
endmsc;
```

15.3 Expression en ligne avec composition en parallèle

Cet exemple illustre comment une expression en ligne parallèle décrit l'imbrication d'une demande de connexion de la part de l'instance 'Station_Res', c'est-à-dire 'MDAT(CR)' suivi de 'ICONind', avec une demande de déconnexion de la part de l'environnement, c'est-à-dire 'IDISreq' suivi de 'MDAT(DR)'. Voir Figure 54.

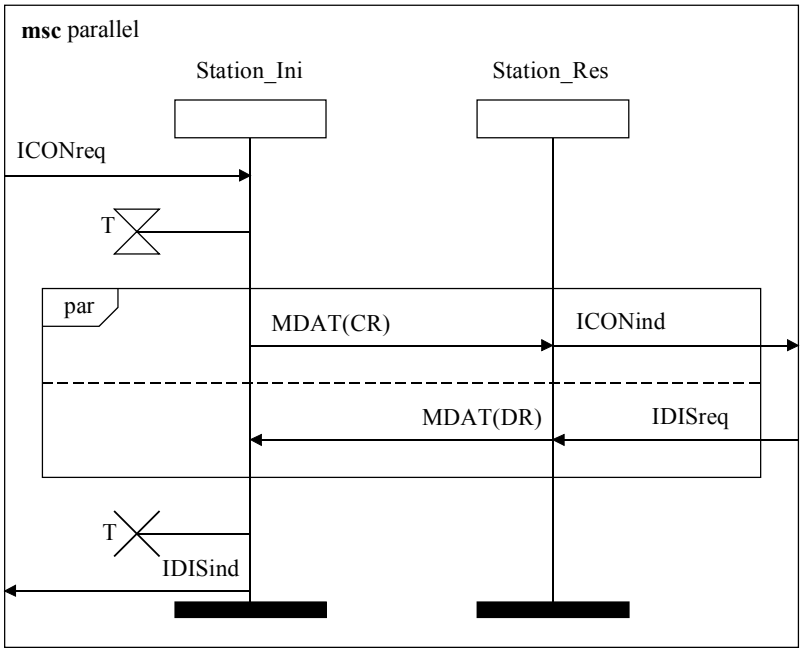


Figure 54/Z.120 – Expression en ligne avec composition en parallèle

```
msc parallel;
inst Station_Ini;
inst Station_Res;
gate out ICONreq to Station_Ini;
gate in ICONind from Station_Res;
gate out IDISreq to Station_Res;
gate in IDISind from Station_Ini;

        Station_Ini: instance;
                        in ICONreq from env;
                        starttimer T;
```

```
        Station_Res:        instance;
        all:                par begin;
            Station_Ini:        out MDAT(CR) to Station_Res;
            Station_Res: in MDAT(CR) from Station_Ini;
                                out ICONind to env;
                            par;
            Station_Res: in IDISreq from env;
                                out MDAT(DR) to Station_Ini;
            Station_Ini:        in MDAT(DR) from Station_Res;
                            par end;
        Station_Res:        endinstance;
        Station_Ini: stoptimer T;
                            out IDISind to env;
                            endinstance;
endmsc;
```

16 Références de diagramme MSC

16.1 Référence MSC

Dans cet exemple, la référence MSC 'data_transmission' est utilisée pour décrire deux transmissions de données réussies, faisant référence à la même définition MSC.

La référence *ref* de diagramme MSC possède des portes ambiguës car portant des noms implicites; mais cela n'a pas d'importance tant que cette référence n'est pas utilisée comme référence MSC dans un autre diagramme. Voir Figures 55 et 56.

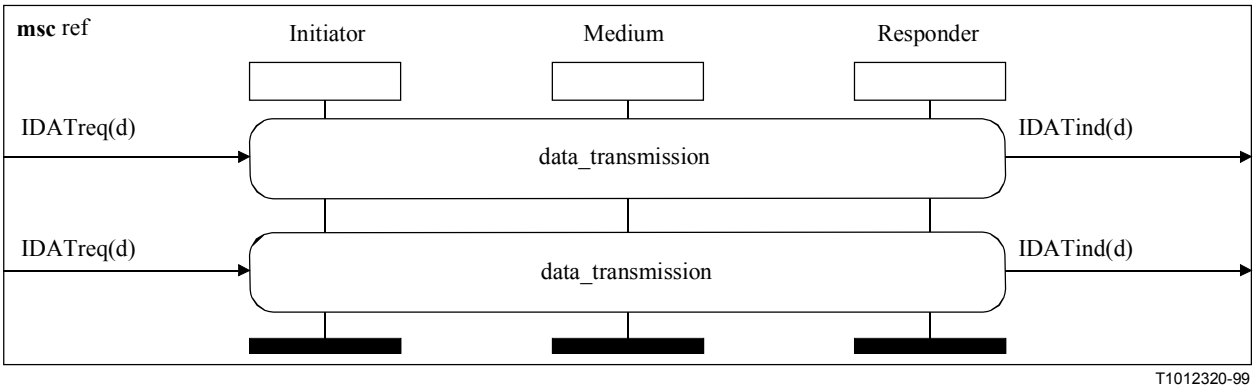


Figure 55/Z.120 – Diagramme MSC comportant des références à d'autres diagrammes MSC

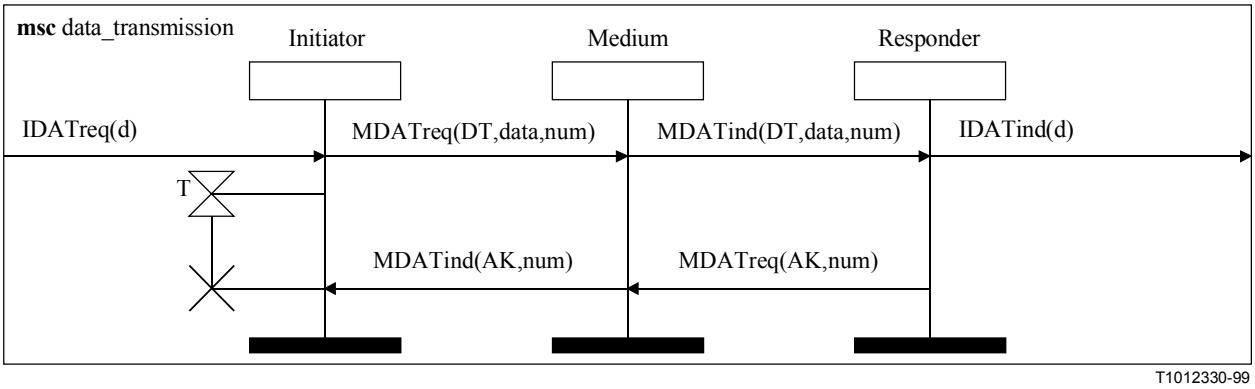


Figure 56/Z.120 – Diagramme MSC référencé

msc ref;
inst Initiator;
inst Medium;
inst Responder;
gate out L1 **to reference** data_trans1;
gate in L2 **from reference** data_trans1;
gate out L3 **to reference** data_trans2;
gate in L4 **from reference** data_trans2;

Initiator:	**instance**;
Medium:	**instance**;
Responder:	**instance**;
all:	**reference** data_trans1:data_transmission;
	gate in IDATreq**,** L1 **from env**;
	gate out IDATind, L2 **to env**
	reference data_trans2:data_transmission;
	gate in IDATreq, L3 **from env**
	gate out IDATind, L4 **to env**;
Initiator:	**endinstance**;
Medium:	**endinstance**;
Responder:	**endinstance**;

endmsc;

msc data_transmission;
inst Initiator;
inst Medium;
inst Responder;
gate out IDATreq(d) **to** Initiator;
gate in IDATind(d) **from** Responder;

Initiator:	**instance**;
Medium:	**instance**;
Responder:	**instance**;
Initiator:	**in** IDATreq(d) **from env**;
	out MDATreq(DT, data, num) **to Medium**;
	starttimer T;
Medium:	**in** MDATreq(DT, data, num) **from Initiator**;
	out MDATind(DT, data, num) **to Responder**;
Responder:	**in** MDATind(DT, data, num) **from Medium**;
	out IDATind(d) **to env**;
	out MDATreq(AK, num) **to Medium**;
Medium:	**in** MDATreq(AK, num) **from Responder**;
	out MDATind(AK, num) **to Initiator**;
Initiator:	**in** MDATind(AK, num) **from Medium**;
	stoptimer T;
Initiator:	**endinstance**;
Medium	**endinstance**;
Responder:	**endinstance**;

endmsc;

16.2 Référence MSC avec porte

Cet exemple montre une référence MSC reliée avec l'extérieur par une porte. Les diagrammes MSC référencés 'message_lost' et 'time_out' sont décrits dans l'exemple montré dans la Figure 61.

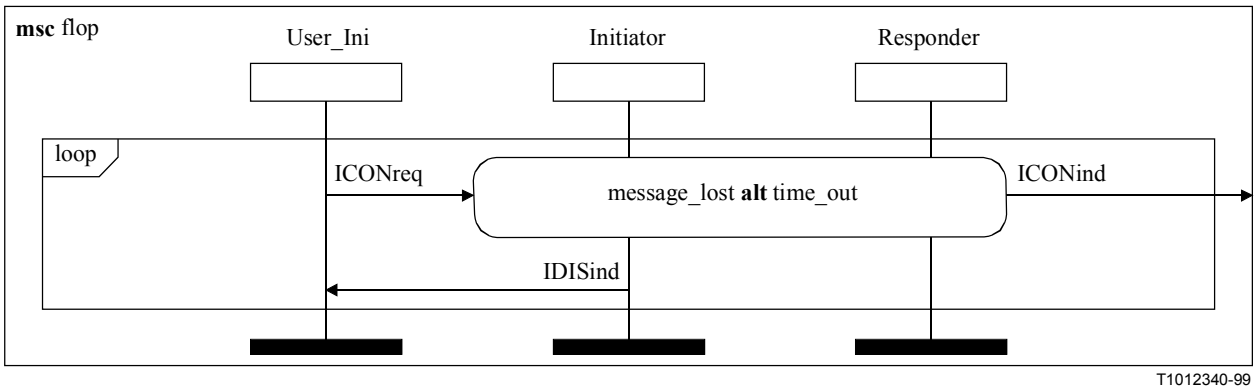


Figure 57/Z.120 – Référence à un diagramme MSC avec portes de messagerie

```
msc flop;
inst User_Ini;
inst Initiator;
inst Responder;
gate in ICONind from failed;

        User_Ini:       instance;
        Initiator:      instance;
        Responder:      instance;
        all:            loop begin;
            User_Ini:       out ICONreq to reference failed;
            Initiator,
            Responder:      reference failed: message_lost alt time_out;
                            gate in ICONreq from User_Ini;
                            gate out ICONind to env;


            Initiator:      out IDISind to User_Ini;
            User_Ini:       in IDISind from Initiator;
                        loop end;
        User_Ini:       endinstance;
        Initiator:      endinstance;
        Responder:      endinstance;
endmsc;
```

17 Diagramme MSC de haut niveau

17.1 MSC de haut niveau avec boucle libre

Dans cet exemple, le diagramme MSC 'setup_attach' illustre la modélisation des connexions à l'aide d'une boucle libre. Voir Figure 58.

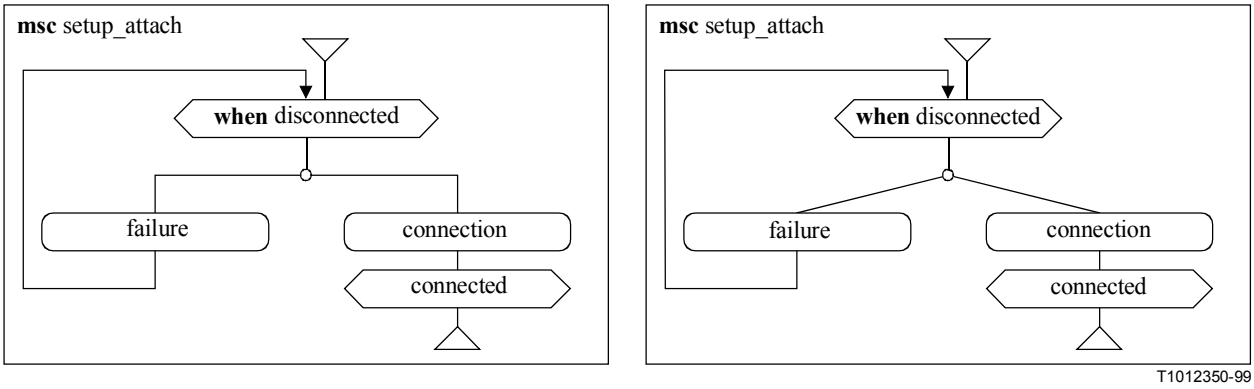


Figure 58/Z.120 – HMSC setup_attach (en deux variantes graphiques différentes)

```
msc setup_attach;
expr L1;
        L1: condition when disconnected seq ( L2 );
        L2: connect seq ( L3 alt L4 );
        L3: (failure); seq ( L1 );
        L4: (connection); seq ( L5 );
        L5: condition connected seq ( L6 );
        L6: end;
endmsc;
```

17.2 MSC de haut niveau avec boucle

Dans cet exemple, le diagramme MSC 'setup_loop' illustre le modèle des connexions à l'aide d'une boucle attachée à la référence MSC 'failure'. Voir Figure 59.

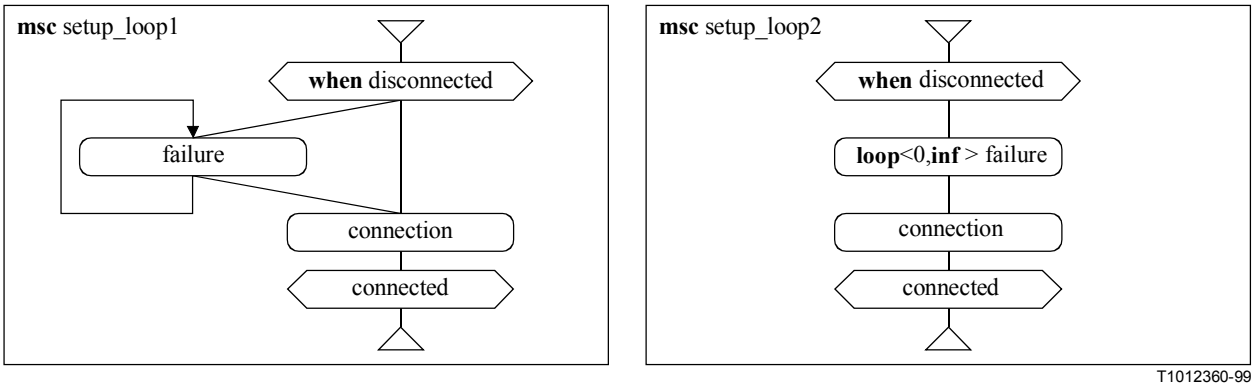


Figure 59/Z.120 – Diagramme setup_loop selon deux variantes différentes

```
msc setup_loop1;
expr L1;
        L1: condition when disconnected seq (L2 alt L3);
        L2: (failure) seq (L2 alt L3);
        L3: (connection) seq (L4);
        L4: condition connected seq (L5);
        L5: end;
endmsc;

msc setup_loop2;
expr L1;
        L1: condition when disconnected seq (L2);
        L2: (loop <0,inf> failure) seq (L3);
        L3: (connection) seq (L4);
        L4: condition connected seq (L5);
        L5: end;
endmsc;
```

17.3 MSC de haut niveau avec composition en alternative

Dans cet exemple le diagramme MSC 'alternative' montre une construction d'alternative contenant un bon 'parenthésage': la construction du branchement possède sa propre construction d'union.

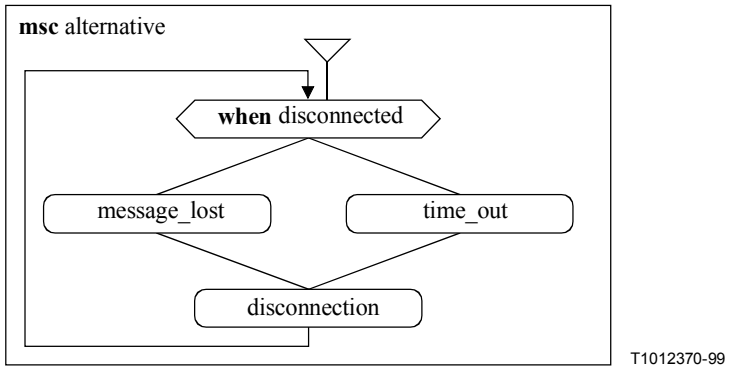


Figure 60/Z.120 – Diagramme HMSC avec expression en alternative

```
msc alternative;
expr L1;
        L1: condition disconnected seq ( L2 alt L3 );
        L2: (message_lost); seq ( L4 );
        L3: (time_out); seq ( L4 );
        L4: (disconnection); seq ( L1 );
endmsc;
```

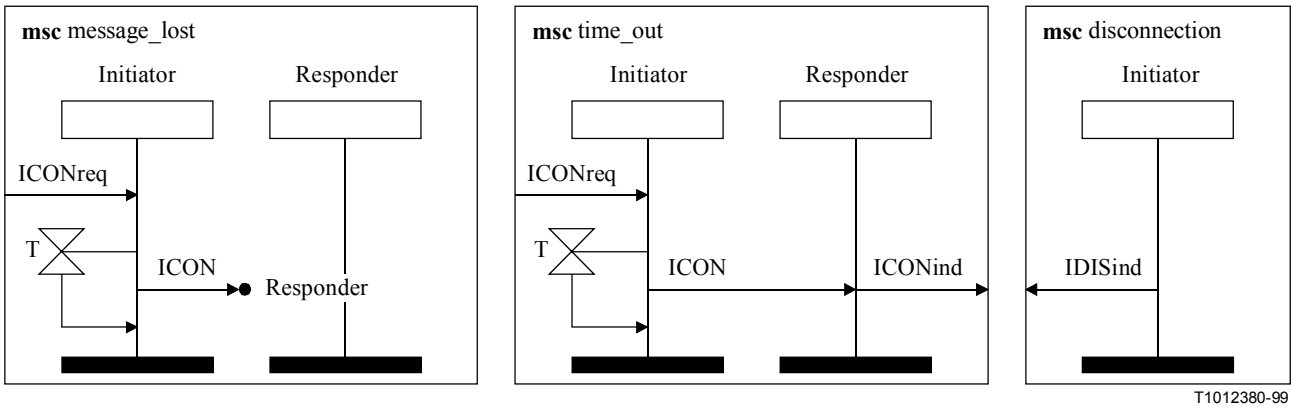


Figure 61/Z.120 – Simples diagrammes MSC référencés à partir du HMSC de la Figure 60

```
msc message_lost;
inst Initiator;
inst Responder;
gate out ICONreq to Initiator;

        Initiator: instance;
            in ICONreq from env;
            starttimer T;
            out ICON to lost Responder;
            timeout T;
        endinstance;
        Responder: instance;
            in ICON from Initiator;
        endinstance;
endmsc;
```

```
msc time_out;
inst Initiator;
inst Responder;
gate out ICONreq to Initiator;
gate in ICONind from Responder;

        Initiator: instance;
            in ICONreq from env;
            starttimer T;
            out ICON to Responder;
            timeout T;
        endinstance;
        Responder: instance;
            in ICON from Initiator;
            out ICONind to env;
        endinstance;
endmsc;

msc disconnection;
inst Initiator;
gate in IDISind from Initiator;

        Initiator: instance;
            out IDISind to env;
        endinstance;
endmsc;
```

17.4 MSC de haut niveau avec composition en parallèle

Dans cet exemple, le parallélisme dans le diagramme HMSC est utilisé pour exprimer le fait que la demande de connexion de la part de l'initiateur ('Initiator') et la demande de déconnexion de la part du répondeur ('Responder') sont exécutées en parallèle.

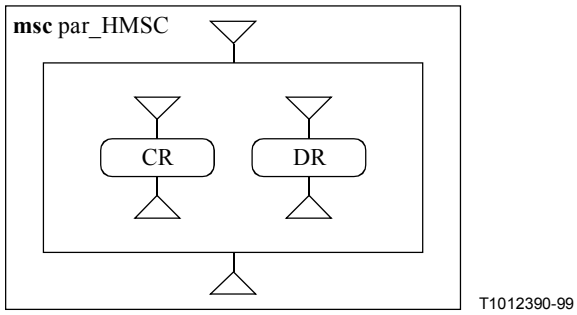


Figure 62/Z.120 – Diagramme HMSC avec expression en parallèle

```
msc par_HMSC;
expr L1;
        L1:  expr L2;
                L2: (CR); seq ( L3 );
                L3: end;
             endexpr
        par
             expr L4;
                L4: (DR); seq ( L5 );
                L5: end;
             endexpr
        seq ( L6 );
        L6: end;
endmsc;
```

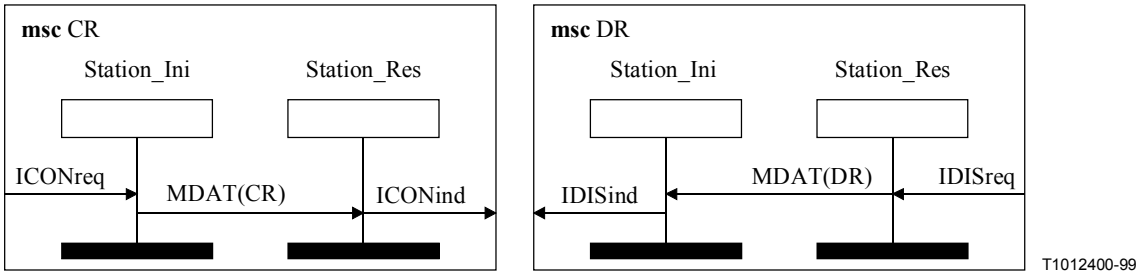


Figure 63/Z.120 – Diagrammes MSC référencés à partir du HMSC de la Figure 62

```
msc CR;
inst Station_Ini;
inst Station_Res;
gate out ICONreq to Station_Ini;
gate in ICONind from Station_Res;

        Station_Ini: instance;
                        in ICONreq from env;
                        out MDAT(CR) to Station_Res;
                        endinstance;
        Station_Res:    instance;
                        in MDAT(CR) from Station_Ini;
                        out ICONind to env;
                        endinstance;
endmsc;

msc DR;
inst Station_Ini;
inst Station_Res;
gate out IDISreq to Station_Res;
gate in IDISind from Station_Ini;

        Station_Res:    instance;
                        in IDISreq from env;
                        out MDAT(DR) to Station_Ini;
                        endinstance;
        Station_Ini: instance;
                        in MDAT(DR) from Station_Res;
                        out IDISind to env;
                        endinstance;
endmsc;
```

ANNEXE A

Index

Les entrées sont les mots clés <keyword> et les non terminaux extraits de la *grammaire textuelle concrète* et de la *grammaire graphique concrète*. Les numéros de page indiqués font référence à la version anglaise du texte.

<abs measurement>, 65, 67
<abs time area>, 66, 67
<abs time expr>, 67
<abs time mark>, 7, 64, 65, 66, 67
<abs time symbol>, 22, 25, 26, 29, 46, 47, 48, 49, 67, 71, 76, 86
<action area>, 20, 49, 68
<action statement>, 48, 49
<action symbol>, 42, 49, 66, 67
<action>, 18, 22, 48
<actual create in gate area>, 39, 40, 49
<actual create in gate>, 34, 76
<actual create out gate area>, 39, 40, 49
<actual create out gate>, 34, 76
<actual data parameter list>, 56
<actual data parameters>, 56, 75
<actual gate area>, 39, 76
<actual in call gate area>, 39, 40
<actual in call gate>, 35, 76
<actual in gate area>, 26, 29, 39, 40
<actual in gate>, 34, 76
<actual in reply gate area>, 39, 40
<actual in reply gate>, 36, 76
<actual instance parameter>, 75
<actual instance parameters>, 75
<actual instance parm list>, 75
<actual message list>, 75
<actual message parameters>, 75
<actual order in gate area>, 39, 41, 42
<actual order in gate>, 19, 34, 76
<actual order out gate area>, 39, 41, 42
<actual order out gate>, 34, 76
<actual out call gate area>, 39, 40
<actual out call gate>, 35, 76
<actual out gate area>, 26, 29, 39
<actual out gate>, 34, 76
<actual out reply gate area>, 39, 40
<actual out reply gate>, 36, 76
<actual parameters>, 75, 77
<actual timer list>, 75
<actual timer parameters>, 75
<alphanumeric>, 7, 8, 51
<alt area>, 71, 72
<alt expr>, 70, 71
<alt op area>, 85, 86, 87
<apostrophe>, 8, 51
<binding>, 58, 60, 61
<bounded time>, 66
<call in area>, 29, 30, 31
<call in symbol>, 30, 31, 67
<call in>, 27, 28, 32
<call out area>, 29, 30
<call out symbol>, 30, 67
<call out>, 28, 32
<character string>, 2, 6, 8, 12, 13, 48, 51
<close par>, 11, 51
<coevent area>, 68
<coevent layer>, 68
<comment area>, 13
<comment symbol>, 13
<comment>, 12
<composite special>, 6, 8
<concurrent area>, 20, 68
<condition area>, 20, 44, 86
<condition identification>, 43, 44, 85
<condition name list>, 44
<condition symbol>, 44, 86
<condition text>, 44, 86
<condition>, 19, 44
<connection point symbol>, 86
<connector layer>, 6, 20
<cont int symbol>, 66, 67
<cont interval>, 66
<containing area>, 16
<containing-clause>, 15, 16, 18, 19, 20
<coregion symbol>, 68, 69
<coregion symbol1>, 68, 69
<coregion symbol2>, 68, 69
<create area>, 20, 42, 49, 68
<create gate identification>, 34, 35, 39, 40
<create gate>, 18
<create source>, 35
<create target>, 34
<create>, 18, 32, 35, 49
<createline symbol>, 22, 37, 39, 40, 49, 50, 66
<data definition>, 15, 55
<data parameter decl>, 17, 56
<data statement list>, 48, 61
<data statement>, 61
<decimal digit>, 7, 8
<decomposition>, 15, 22, 23, 78
<def create in gate area>, 38, 39, 49
<def create in gate>, 18, 34, 35
<def create out gate area>, 38, 39, 49
<def create out gate>, 18, 34, 35
<def gate area>, 20, 37
<def in call gate area>, 37, 38
<def in call gate>, 18, 35
<def in gate area>, 26, 29, 37, 38
<def in gate>, 18, 25, 34, 35
<def in reply gate area>, 38
<def in reply gate>, 18, 35, 36
<def order in gate area>, 37, 39, 42
<def order in gate>, 18, 34
<def order out gate area>, 37, 39, 42
<def order out gate>, 18, 19, 34
<def out call gate area>, 37, 38
<def out call gate>, 18, 35, 36

<def out gate area>, 26, 29, 37, 38
<def out gate>, 18, 25, 34, 35
<def out reply gate area>, 38
<def out reply gate>, 18, 35, 36
<define statement>, 61
<defining msc reference area>, 16
<defining msc reference>, 15
<defining part area>, 16
<delim>, 11, 51
<document head>, 15, 16, 55, 59
<duration>, 46, 55
<durationlimit>, 46
<dynamic decl list>, 15, 55
<end coregion>, 19, 68, 69
<end method>, 19, 28, 32
<end suspension>, 19, 28, 32
<end>, 2, 12, 13, 15, 17, 18, 22, 28, 43, 44, 50, 51, 54, 55, 68, 70, 71, 75, 76, 77, 85
<equal par delim>, 51
<equal par>, 11
<escape decl>, 51
<escapechar>, 11, 51
<event area>, 20, 23, 27, 31, 42
<event definition>, 18, 22
<event layer>, 6, 20, 31, 72
<exc area>, 71, 72
<exc expr>, 70, 71
<exc inline expression symbol>, 72, 73
<expression>, 25, 27, 44, 46, 48, 58, 60, 64, 67, 71, 85
<extra-global>, 70
<found message area>, 2, 26, 27
<found message symbol>, 27, 30, 36, 37, 38, 39, 40
<found method call area>, 30, 31
<found reply area>, 30
<found reply symbol>, 30, 37, 38, 40
<frame symbol>, 16, 20, 66, 67, 86
<full stop>, 7, 8, 51
<gate def layer>, 6, 20
<gate identification>, 36, 38, 39, 40, 41
<general name area>, 12, 20, 71, 72
<general name symbol>, 12
<general order area>, 25, 26, 30, 31, 39, 41, 46, 49
<general order symbol>, 37, 41
<general order symbol1>, 41, 42
<general order symbol2>, 41, 42
<hmsc condition area>, 86
<hmsc diagram>, 19, 20
<hmsc end area>, 85, 86
<hmsc end symbol>, 86, 87
<hmsc line symbol>, 86, 87
<hmsc line symbol1>, 86
<hmsc line symbol2>, 86
<hmsc reference area>, 86
<hmsc start symbol>, 85, 86, 87
<hmsc>, 17, 19, 85
<identifier>, 2, 3, 15, 18, 61
<incomplete call in>, 28, 32
<incomplete call out>, 28
<incomplete message area>, 2, 20, 26, 42, 68
<incomplete message event>, 18, 24
<incomplete message input>, 24, 32
<incomplete message output>, 24
<incomplete method call area>, 20, 30, 42
<incomplete method call event>, 18, 28
<incomplete reply area>, 20, 30, 42
<incomplete reply in>, 28
<incomplete reply out>, 28, 32
<inf natural>, 71
<informal action>, 48
<inheritance>, 15
<inline create in gate area>, 36, 37
<inline create in gate>, 35, 71
<inline create out gate area>, 36, 37
<inline create out gate>, 35, 71
<inline expr identification>, 25, 70, 71
<inline expr>, 5, 19, 70
<inline expression area>, 12, 20, 71, 72
<inline expression symbol>, 36, 37, 66, 67, 71, 72, 73, 74
<inline gate area>, 26, 29, 36, 71, 72
<inline gate interface>, 70, 71, 74
<inline gate>, 71
<inline in call gate area>, 36
<inline in call gate>, 35, 71
<inline in gate area>, 36
<inline in gate>, 35, 71
<inline in reply gate area>, 36, 37
<inline in reply gate>, 35, 71
<inline order gate area>, 37, 42, 71, 72
<inline order in gate area>, 37
<inline order in gate>, 34, 35, 71
<inline order out gate area>, 37
<inline order out gate>, 19, 34, 35, 71
<inline out call gate area>, 36
<inline out call gate>, 35, 71
<inline out gate area>, 36
<inline out gate>, 35, 71
<inline out reply gate area>, 36, 37
<inline out reply gate>, 35, 71
<input address>, 24, 25, 28, 34
<input dest>, 34, 35, 36
<instance area>, 20, 22, 45
<instance axis symbol>, 5, 23, 25, 26, 29, 30, 31, 44, 46, 47, 48, 49, 68, 69, 71, 72, 76
<instance axis symbol1>, 23, 27
<instance axis symbol2>, 23, 27, 41, 69
<instance body area>, 22, 23
<instance end statement>, 19, 22
<instance end symbol>, 23
<instance event area>, 12, 20, 27
<instance event list>, 18, 32, 70
<instance event>, 18, 22, 32
<instance fragment area>, 22
<instance head area>, 14, 22, 49
<instance head statement>, 19, 22
<instance head symbol>, 22, 23, 49, 66
<instance heading>, 22, 23
<instance item>, 15
<instance kind>, 15, 17, 18, 22, 23, 39
<instance layer>, 6, 20
<instance name list>, 18, 19, 22
<instance parameter decl>, 2, 17
<instance parameter name>, 17
<instance parm decl list>, 2, 17
<int symbol 1>, 66, 67
<int symbol 2>, 66, 67

<int symbol>, 22, 25, 26, 29, 46, 47, 48, 49, 66, 67, 71, 76, 86
<int symbol1>, 67
<int symbol2>, 67
<interval area 2>, 66
<interval area>, 66
<interval label>, 66
<keyword>, 2, 6, 8, 120
<kind denominator>, 18
<label name list>, 85
<left angular bracket>, 8, 71
<left bind symbol>, 58
<left binding>, 58
<left closed>, 7, 66
<left curly bracket>, 7
<left delim>, 51
<left open>, 7, 66
<left square bracket>, 7, 46
<letter>, 7, 8
<lexical unit>, 6, 10
<loop area>, 71
<loop boundary>, 70, 71, 75
<loop expr>, 70
<lost message area>, 2, 26
<lost message symbol>, 26, 30, 36, 38, 39, 40
<lost method call area>, 30
<lost reply area>, 30, 31
<lost reply symbol>, 30, 37, 38, 40
<measurement>, 65, 66
<message area>, 2, 20, 26, 27, 42
<message decl clause>, 15
<message decl list>, 17, 54
<message decl>, 15, 54
<message end area>, 5, 26, 27
<message event area>, 20, 25, 42, 68
<message event>, 18, 24
<message in area>, 5, 25, 26
<message in symbol>, 5, 26, 66, 67
<message input>, 24, 25, 27, 32, 44
<message name list>, 54, 55
<message out area>, 25, 26
<message out symbol>, 25, 66, 67
<message output>, 24, 25, 44
<message parameter decl>, 17
<message parm decl list>, 17
<message sequence chart>, 2, 17
<message start area>, 26
<message symbol>, 5, 25, 26, 29, 30, 31, 36, 38, 39, 40
<method area>, 20, 29, 31
<method call area>, 20, 29, 30
<method call end area>, 29, 30
<method call event area>, 20, 30, 42
<method call event>, 18, 28
<method call gate>, 18
<method call start area>, 29, 30
<method end area>, 31
<method event area>, 31
<method event layer>, 31
<method identification>, 29, 30
<method invokation area>, 31
<method start area>, 31
<method symbol>, 27, 29, 31
<msc body area>, 6, 19, 20
<msc body>, 17, 18, 70, 71
<msc diagram>, 2, 19
<msc document area>, 16
<msc expression>, 85
<msc gate def>, 18
<msc gate interface>, 17, 18, 22, 76, 79
<msc head>, 17
<msc heading>, 14, 16, 19, 20
<msc inst interface>, 17, 18, 22
<msc parameter decl>, 17, 20
<msc ref expr>, 75, 76, 85, 86
<msc ref ident expr>, 75
<msc ref par expr>, 75
<msc ref seq expr>, 75
<msc reference area>, 20, 76
<msc reference identification>, 25, 75, 76
<msc reference symbol>, 16, 39, 40, 41, 66, 67, 76, 78, 86, 87
<msc reference>, 19, 75
<msc statement>, 18
<msc symbol>, 19, 20, 38, 71, 73
<msc>, 17, 19
<mscexpr area>, 20, 85, 86
<msg gate>, 18
<msg identification>, 24, 25, 26, 27, 28, 29, 30, 34, 35, 36
<multi instance event list>, 18, 19, 44
<multi instance event>, 19, 22
<name>, 2, 8, 10, 12, 15, 16, 17, 18, 19, 20, 22, 23, 24, 25, 26, 27, 30, 34, 35, 36, 39, 41, 44, 46, 47, 48, 49, 55, 66, 71, 75, 76, 78, 85
<national>, 7, 10
<nestable par pair>, 51
<nestable par>, 11
<node area>, 86, 87
<node expression area>, 85, 86, 87
<node expression>, 85
<node>, 85
<non-nestable par pair>, 51
<non-nestable par>, 11
<non-orderable event>, 18, 19
<non-parenthesis>, 10, 11
<non-par-non-escape>, 11
<note>, 6, 8, 10
<open par>, 11, 51
<operand area>, 12, 71, 72
<opt area>, 71
<opt expr>, 70
<order dest list>, 18, 34, 35
<order dest>, 18, 19, 34, 35
<order gate>, 18
<orderable event>, 18, 19, 69
<ordered event area>, 41, 42
<other character>, 8, 51
<output address>, 24, 25, 28, 34
<output dest>, 34, 35, 36
<overline>, 7
<par area>, 71, 72
<par decl list>, 51
<par expr area>, 86
<par expr>, 70, 71, 75
<par expression>, 85
<par frame symbol>, 66, 67, 86

<parameter defn>, 60
<parameter list>, 24, 25, 27, 46, 47, 48, 49, 56, 60
<parent>, 75, 78
<parenthesis declaration>, 15, 51
<parenthesis>, 10, 11
<pattern>, 25, 27, 46, 48, 58, 59, 60, 65
<pure data string>, 10, 11
<qualifier left>, 8, 15
<qualifier right>, 8, 15
<qualifier>, 2, 3, 15
<ref gate>, 76
<reference gate interface>, 75, 76, 78
<reference identification>, 19, 24, 25, 34, 35
<rel measurement>, 65
<rel time mark>, 7, 65
<reply area>, 20, 29
<reply end area>, 29, 30
<reply event area>, 20, 31, 42
<reply gate>, 18, 71
<reply in area>, 29, 31
<reply in symbol>, 31
<reply in>, 28, 32
<reply out area>, 29, 31
<reply out symbol>, 31
<reply out>, 28, 32
<reply start area>, 29, 30
<reply symbol>, 29, 37, 38, 40, 66, 67
<restart symbol>, 46, 47, 48, 66
<right angular bracket>, 8, 71
<right bind symbol>, 58
<right binding>, 58
<right closed>, 7, 66
<right curly bracket>, 7
<right delim>, 51
<right open>, 7, 66
<right square bracket>, 7, 46
<sdl reference>, 15
<separator area>, 16, 72
<separator symbol>, 66, 67, 72
<shared alt expr>, 70
<shared condition>, 19, 22, 43, 44
<shared event area>, 20
<shared exc expr>, 70
<shared inline expr>, 19, 70
<shared instance list>, 44
<shared loop expr>, 70
<shared msc reference>, 19, 75
<shared opt expr>, 70
<shared par expr>, 70
<shared>, 43, 44, 45, 70, 75
<simple msc diagram>, 19
<singular time>, 66
<space>, 8, 10
<special>, 6, 8, 51
<start area>, 85
<start coregion>, 19, 68, 69
<start method>, 19, 28, 32
<start suspension>, 19, 28, 32
<start symbol1>, 47
<start symbol2>, 47
<start>, 85
<starttimer>, 46
<stop symbol>, 23, 50, 66, 67
<stop>, 7, 19, 50
<stoptimer>, 46
<string>, 10, 11, 46, 50, 51, 52, 53, 54, 55, 56, 58, 59
<substructure reference>, 78
<suspension area>, 20, 31
<suspension event layer>, 31
<suspension symbol>, 27, 29, 31, 32
<text area>, 6, 13, 20
<text definition>, 13, 18, 85
<text layer>, 6, 20, 85
<text symbol>, 13
<text>, 8, 11, 13, 15
<textual defining part>, 15
<textual msc document>, 15, 19
<textual utility part>, 15
<time dest list>, 18, 19, 70, 75
<time dest>, 19
<time interval area>, 66, 71, 72
<time interval>, 19, 26, 29, 66, 70, 71, 75, 76, 85, 86
<time offset>, 17, 20, 64
<time point>, 64, 66
<timeout area>, 46, 47, 48
<timeout area1>, 47
<timeout area2>, 47, 48
<timeout symbol>, 47, 48, 66, 67
<timeout symbol1>, 48
<timeout symbol2>, 48
<timeout symbol3>, 46, 47, 48
<timeout>, 46
<timer area>, 20, 42, 46, 68
<timer decl clause>, 15
<timer decl list>, 18, 55
<timer decl>, 15, 55
<timer name list>, 55
<timer parameter decl>, 17
<timer parm decl list>, 17, 18
<timer start area>, 46, 48
<timer start area1>, 46
<timer start area2>, 46
<timer start symbol>, 46, 47, 48, 66, 67
<timer statement>, 18, 46
<timer stop area>, 46, 47
<timer stop area1>, 47
<timer stop area2>, 47
<timer stop symbol>, 47, 66, 67
<timer stop symbol1>, 47
<timer stop symbol2>, 46, 47
<type ref list>, 54, 55
<undefine statement>, 61
<underline>, 7, 8, 10, 51
<unmatched string>, 11
<upward arrow head>, 7
<using clause>, 15
<utility part area>, 16
<variable decl item>, 55
<variable decl list>, 55, 56
<variable list>, 55
<vertical line>, 7
<virtuality>, 15, 16, 17
<void symbol>, 5, 25, 26, 30, 31, 36, 37, 38, 39, 40, 41
<wildcard decl>, 55
<wildcard>, 59
<word>, 6, 7

SÉRIES DES RECOMMANDATIONS UIT-T

Série A	Organisation du travail de l'UIT-T
Série B	Moyens d'expression: définitions, symboles, classification
Série C	Statistiques générales des télécommunications
Série D	Principes généraux de tarification
Série E	Exploitation générale du réseau, service téléphonique, exploitation des services et facteurs humains
Série F	Services de télécommunication non téléphoniques
Série G	Systèmes et supports de transmission, systèmes et réseaux numériques
Série H	Systèmes audiovisuels et multimédias
Série I	Réseau numérique à intégration de services
Série J	Réseaux câblés et transmission des signaux radiophoniques, télévisuels et autres signaux multimédias
Série K	Protection contre les perturbations
Série L	Construction, installation et protection des câbles et autres éléments des installations extérieures
Série M	RGT et maintenance des réseaux: systèmes de transmission, circuits téléphoniques, télégraphie, télécopie et circuits loués internationaux
Série N	Maintenance: circuits internationaux de transmission radiophonique et télévisuelle
Série O	Spécifications des appareils de mesure
Série P	Qualité de transmission téléphonique, installations téléphoniques et réseaux locaux
Série Q	Commutation et signalisation
Série R	Transmission télégraphique
Série S	Equipements terminaux de télégraphie
Série T	Terminaux des services télématiques
Série U	Commutation télégraphique
Série V	Communications de données sur le réseau téléphonique
Série X	Réseaux de données et communication entre systèmes ouverts
Série Y	Infrastructure mondiale de l'information et protocole Internet
Série Z	**Langages et aspects généraux logiciels des systèmes de télécommunication**

Genève, 2001