ITU - Telecommunication Standardization Sector **Temporary Document 9xx-E**

STUDY GROUP 10 **Original: English**

Geneva, 98.03.24 – 98.04.01

Question: 9

SOURCE: RAPPORTEUR

TITLE: **Title: Master List of Corrections for Z.120 (MSC-96)**

Contact: Øystein Haugen, Ericsson AS, etooha@eto.ericsson.se

To Do items

identified by the MSC group (ITU meeting, Geneva April 1996) as work program for the study period 1997 - 2000:

1. Overall structure of Z.120

- Improvement of the graphical grammar based on the study of existing graph grammars formalisms

2. Abstraction, structuring and modularisation concepts

-- remote procedure corresponding to SDL
- synchronous communication construct
- hierarchical structuring, e.g. grouping of instances
- inclusion of object-oriented concepts as inheritance, virtuality, etc.

3. Composition of MSCs

- strong global condition concept and other variants of MSC conditions
- formal data descriptions for use in conditions
- extnsion of composition mechanisms based on process algebra, e.g. disruption, interruption
- further investigation of the relation between composition mechanisms based on conditions and those based on process algebra operators

4. Data concepts

- formal data descriptions (ASN.1 or ADT) for messages, context parameters, conditions and actions
- declaration area for conditions and instances

5. Using MSC for test case specification

- Test case specifications by means of total ordering of events

6. Real-Time application of MSCs

- Inclusion of non-functional properties concerning performance, error rates etc, within MSCs

7. Exchange format for MSCs

- Definition of CIF corresponding to SDL

8. Object oriented modelling with MSCs as event traces

- Formalizing use cases by means of MSCs
- special language constructs concerning object communication, e.g. sequential (synchronous) object communication and communication caused by the encapsulation relation

Correction List for Z.120

The following classification is used for each request:

1: Error
2: Textual correction
3: Open Item
4: Deficiency
5: Clarification
6: Extension
7: Modification

Terminology

1. An *error* is an internal inconsistency within Z.120
2. A *textual correction* is a change to text or diagrams of Z.120 which corrects clerical or typographical errors.
3. An *open item* is a concern identified but not resolved. An open item may be identified either by a Change Request, or by agreement of the Working Party.
4. A *deficiency* is an issue identified where the semantics of MSC are not (clearly) defined by Z.120.
5. A *clarification* is a change to the text or diagrams of Z.120 which clarifies previous text or diagrams which could be ambiguously understood without the clarification. The clarification should attempt to make Z.120 correspond to the semantics of MSC as understood by the Working Party.
6. A *modification* is a change to the text or diagrams of Z.120 which changes the semantics of MSC.
7. An *extension* is a new feature, which must not change the semantics of features defined in Z.120.
8. A *closed item* is an issue which has been discussed and rejected. The item is retained in the document for historical and explanatory purposes.

Errors

Section 4:
Grammar of orderable event: Section 4.1 Message Sequence Chart, p.16:
<orderable event> ::=
[<event name>]
{ <message event> | <incomplete message event> | <create>

| <timer statement> | <action> }
[**before** <event name list>]

The optional <event name> is used when the event is generally ordered.

Change:
<event name list> ::=
{ <event name> | <gate name> } [, <event name list>]

to

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

Section 5:
Section 5.3 Inline expression p.39:
In <alt expr> an [<inline gate interface>] is missing before the first <msc body>

The correct grammar rule is:
<alt expr>::=
alt begin [<inline expr identification>] <end>
[<inline gate interface>] <msc body>
{alt <end> [<inline gate interface>] <msc body> }*
alt end

Grammar of loop: Section 5.4 MSC reference, p. 43:
The rule <msc ref loop expr> allows a loop boundary without the loop keyword:

Change:
<msc ref loop expr> ::= [loop] [<loop boundary>]{empty ...}

to:

<msc ref loop expr> ::= [loop [<loop boundary>]]{empty ...}

Textual corrections:

Section 4:
Partial Order**:** Section 4.1 Message Sequence Chart, p. 18:
In section 4.1 an ordering "< " of events is introduced. a < b means that event a must be completed before event b can begin. But why is < said to be reflexive then?

Change:
A binary relation which is transitive, antisymmetric and reflexive is called partial order.
to

A binary relation which is transitive, antisymmetric and irreflexive is called partial order.

Misspelling: Section 4.1 page 18, the 2nd line of the 2nd paragraph over Figure 4.1
Change
figure 1a
to
Figure 4.1(a)

Misspelling: Section 4.1 page 18 the 2nd line of the 1st paragraph over Figure 4.1
Change
(figure 1b)
to
(Figure 4.1(b))

Misspelling Section 4.3 Message page 20, the 2nd line of the 1st paragraph
Change
(a gate)
to
(through a gate)

Misspelling 4.3 Message page 20, the 2nd line of the 3rd paragraph
Change
figure 1a
to
Figure 4.1(a)

Semantics description for conditions Section 4.6 Condition p.31:
In MSC-96 no decomposition wrt. conditions is defined. There is only composition by means of HMSCs. Conditions in general have the state of a comment. Initial and final global conditions have a purpose expressed in form of static semantics rules. Intermediate conditions have no other purpose than documentation. Global conditions refer to all instances contained in the respective MSC. MSCs need not contain the same set of instances for composition.

Change:
Initial, intermediate and final global conditions are not introduced merely for documentation purposes in the sense of comments or illustrations. Global conditions indicate possible continuations of MSCs containing the same set of instances by means of condition name identification.

to

Initial and final global conditions are not introduced merely for documentation purposes in the sense of comments or illustrations. Global conditions indicate possible continuations of MSCs by means of condition name identification.

Section 5
Missing line break in production, Section 5.1 page 36
Insert in production for <coregion> before <coevent> a line break.

Missing line break in production, Section 5.1 page 37
Insert in production for <coregion symbol> before <coevent layer> a line break.

Misspelling Section 5.3 page 40
The production of <inf natural> should not allow multiple naturals, change to:
<inf natural> ::=
inf | <natural name>

Misspelling Section 5.5 page 48, the 3rd line of the 3rd paragraph
Change
<condition symbols>
to
<condition symbol>s

Section 6, Message Sequence Chart examples:
Section 6.1 Standard message flow diagram, p. 50:
MSC connection/textual grammar:

First instance-head:
calling party -> calling_party

Second instance-head:
called party -> called_party

Section 6.2 Message overtaking, p. 51:
MSC message_overtaking/textual grammar:
Second instance-head:
inst 2 -> inst2

Section 6.3 MSC basic concepts, p. 52, 53:
textual grammar
action statement in instance oriented and event oriented syntax:
action setting_counter; -> action 'setting_counter';

Misspelling Section 6.4 MSC-composition / MSC-decomposition, page 53,
The first paragraph should read:
In this example the composition of MSCs by means of global conditions is demonstrated. The final global condition 'Wait_For_Resp' of MSC 'connection_request' is identical with the initial global condition of MSC connection confirm. Therefore both MSCs may be composed to the resulting MSC 'connection' (example 6.5).

Section 6.4 MSC-composition/MSC-decomposition, p. 54, 55:
MSC connection_confirm/textual grammar:

First instance-head:
Inititator -> Initiator

MSC con_setup:
L2: seq L3 -> seq (L3)

Section 6.7 Local condition, p. 58:
MSC confirm/textual grammar
instance Responder:
condition disconnected; -> condition disconnected shared;

Section 6.14 Inline expression with alternative composition, p. 64, 65:
MSC alternative and MSC exception/graphical grammar:
Wait for Resp -> Wait_for_Resp

MSC alternative and MSC exception/textual grammar:
after second all: Wait for Resp -> Wait_for_Resp

Section 6.17 MSC reference, p. 68, 69:
MSC ref/textual grammar:
twice: Medium -> Medium:

MSC data_transmission / textual grammar:
Medium -> Medium:
third out-statement: outIDATind(d) -> out IDATind(d)

Section 6.21 High-Level MSC with alternative composition, p.71, 72:
MSC alternative/textual grammar:
L2: message lost -> message_lost
L3: time out -> time_out

MSC message_lost
instance Responder:
in ICON Initiator; -> in ICON from Initiator;

Misspelling Section 6.21, page72, the 4th line from the bottom
The last MSC description should read in extenso:

msc disconnection; **inst** Initiator;
instance Initiator;
out IDISind **to env**;
endinstance;
endmsc;

Section 6.22 High-Level MSC with parallel composition, p.73:
MSC par_HMSC / textual grammar:
msc par_HMSC -> msc par_HMSC;
twice: endexpr; -> endexpr

MSC CR / textual grammar:
first endinstance-statement:
endinstance -> endinstance;

Clarifications:

Section 4:
Grammar of shared: Section 4.6 Condition, p. 31:
Semantics description is missing for:

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

That denotes that either the keyword 'ALL' or an optional list is to follow the keyword 'SHARED'. Since the list is optional, that means that 'SHARED' may as well stand alone.
Add some sentences explaining the semantics:
E.g., according to the possibilities of an instance oriented and an event oriented syntax description (see 4.1) there are two syntax forms for conditions: within the instance oriented description the <shared condition> form is used whereas in the event oriented description a multi instance form is used employing a <multi instance event list> followed by a colon and the non-terminal <condition>. (see 4.1). In the <shared condition> form the keyword **shared** is used consistently also for local conditions, therefore within the non-terminal <shared> the <shared instance list> is optional.

Section 5:
Timers and decomposition. Section 5.2 Instance decomposition p 38
It was recognized that decomposed instances could have timer events and that the required interface consistency of the decomposition is not properly described.

Following the sentence "An analoguous correspondence must hold for incoming messages." the following paragraph shall be inserted:

Timer events of the decomposed instance should also appear in the decomposition diagram. Timer set, timeout and reset of the same timer must appear on the same instance within the decomposition. Note that not all timer events of a decomposition need to appear on the decomposed instance, while all timer events of the decomposed instance has to appear in the decomposition.

Stop and decomposition. Section 5.2 Instance decomposition p 38
Following the paragraph on timers and decomposition the following paragraph should be inserted:

When the decomposed instance is stopped, the decomposition should include stops on all its contained instances.

Semantics of loop: Section 5.3 Inline expression, p.41:
Some clarification is needed which explains that “The passes of a loop are connected by means of the (weak) sequential composition.” (to be appended to the paragraph explaining the semantics of **loop** at the very end of the page).

Exception expression. Section 5.3 Inline expressions p 41
To clarify the scope of an exception expression, insert after the sentence “All exception expressions must be shared by all instances in the MSC.” the sentence:

The exception expression is a shorthand for an alternative expression where the rest of the enclosing frame is the second operand.

Extensions:

Section 5
Substitution of message parameters. Section 5.4 p 43
To extend substitution also to cover substitution of message parameters, replace the production:

<replace message> ::=
[**msg**] <message name> **by** <message name>

by

<replace message> ::=
[**msg**] <msg identification> **by** <msg identification>

Substitution of Timers and Conditions Section 5.4 p 43
To extend substituion also to cover substitution of timers and conditions replace
<substitution> ::=
<replace message> | <replace instance> | <replace msc>

by

<substitution> ::=
<replace message> | <replace instance> | <replace msc> |
<replace timer> | <replace condition>

Add the following productions:
<replace timer> :=
[**tim**] <timer identification> **by** <timer identification>
<timer identification> ::=
<timer name> [, < timer instance name] [(<duration name>)]
<replace condition> ::=
[**cond**] <condition name list> **by** <condition name list>

Inclusion of text definition / text area in HMSC. Section 5.5 p 45-46
As appears obvious, free text definitions / text areas should be allowed also in HMSC.
Graphical grammar.
Change the definition of <mscexpr area> to
<mscexpr area> ::= {<text layer> <start area> <node expression area>* <hmsc end area>}
Textual grammar.
Change the definition of <msc expression> to
<msc expression> := <start> {<node expression> | <text definition>}*