5i’

## D.3.10 *Traitement des donn´ees*

### D.3.10.1 *D´eclarations de variables*

Les variables sont locales à une instance de processus, ce qui signifie que toutes les variables ont une et une seule instance de processus. Seule l’instance de processus propri´etaire peut modifier la valeur des variables.

Les variables d´eclar´ees dans la figure D-3.10.1 sont locales à chaque instance de processus P; elles ne peuvent dont | tre atteintes et modifi´ees que par chaque instance de processus P (chaque instance de processus peut avoir acc`es à sa propre copie de variables ou modifier celle-ci).

**Figure D-3.10.1 [T24.100] (a traiter comme tableau MEP), p. 1**

Une variable peut | tre initialis´ee imm´ediatement apr`es la d´eclaration, comme indiqu´e dans la figure D-3.10.2.

**Figure D-3.10.2 [T25.100] (a traiter comme tableau MEP), p. 2**

Le LDS permet deux mani`eres d’initialiser des variables. Il est possible de d´eclarer une valeur initiale pour toutes les variables d’une certaine sorte en utilisant l’instruction DEFAULT dans la d´efinition de type de donn´ees (voir le § D.6.4.5). Dans ce cas, l’initialisation est valable pour toutes les variables de cette sorte.

Par ailleurs, il est possible d’initialiser chaque variable d’une certaine sorte par une valeur comme indiqu´e dans la figure D-3.10.2. S’il existe à la fois un ´enonc´e DEFAULT et une initialisation lors de la d´eclaration, c’est cette derni`ere qui pr´evaut. Si une variable n’est pas initialis´ee, sa valeur initiale est consid´er´ee comme <<ind´efinie>> dans le syst`eme.

Naturellement, la notation abrégée de l’initialisation de variables indiquée dans la figure D-3.10.2 n’est utilisable que pour des variables simples ou pour des types de données permettant une notation concrète compacte pour les variables (on trouvera un autre exemple dans la figure D-3.10.3).

**Figure D-3.10.3 [T26.100] (à traiter comme tableau MEP), p. 3**

La figure D-3.10.3 montre que la sorte Struct a une notation concrète abrégée indiquant une valeur de structure. Dans le cas d’un générateur de tableaux, il est suggéré d’initialiser explicitement les variables en simulant une expression <<While construct>> dans la chaîne de transition initiale du processus (voir la figure D-3.10.4).

**Figure D-3.10.4 [T27.100] (à traiter comme tableau MEP), p. 4**

D.3.10.2 *Variables révélées/visualisées*

Deux processus peuvent échanger des informations par d'autres moyens que des signaux. Un processus peut avoir accès à la valeur d'une variable possédée par un autre processus grâce à l'opération VIEW (visualiser). Il existe cependant plusieurs règles à appliquer:

— les deux processus doivent appartenir au même bloc;

— le processus exécutant l'opération VIEW doit spécifier l'identificateur de la variable visualisée dans la définition de visibilité;

— le processus révélant la variable doit la déclarer avec l'attribut REVEALED (révélé);

— l'identificateur de sorte (ou identificateur de syntype) de la déclaration de variables et de la définition de visibilité doivent être les mêmes.

La valeur que le processus de visualisation obtient par l'opération VIEW est la même que celle qu'obtient le processus de révélation par un accès ordinaire.

Etant donné que le processus de visualisation ne possède pas la variable visualisée, il ne peut en modifier la valeur. Naturellement, la valeur visualisée peut être affectée à une variable que possède lui-même le processus effectuant les visualisations.

L'utilisateur du LDS peut trouver que la définition des variables révélées/visualisées offre une manière facile de spécifier la communication entre deux processus. Cependant, plusieurs problèmes peuvent se poser dans la mise en oeuvre des systèmes ainsi spécifiés et la présente section a pout but d'aider les utilisateurs à éviter ou à surmonter de tels problèmes. Décrire des systèmes du LDS qui ont mis en oeuvre une valeur révélée/visualisée est moins difficile, car les problèmes auront été surmontés dans la mise en oeuvre de sorte et qu'il sera possible de mettre en concordance la solution choisie avec le LDS.

Dans le reste de la présente section, on admet que le processus R (Révélateur) possède et révèle des variables et que le processus V (Visualisateur) se rapporte à ses variables dans sa définition de visualisation.

Une tentative de visualiser des variables du processus V avant la création du processus R aboutit en LDS à une erreur. L'utilisateur peut éviter ce problème de deux manières:

— soit en s'assurant que l'instance de procesus révélatrice R est créée et a initialisé les variables pertinentes avant l'instance de processus de visualisation V;

— soit en s'assurant que V ne passe pas dans des transitions utilisant des variables révélées/visualisées avant que R ait été créé et ait initialisé les variables pertinentes.

Dans le premier cas, une manière simple d'y parvenir consiste à faire de R l'ascendant (ou ancêtre) de V (comme dans l'exemple de la figure D-3.10.5), ou d'admettre que R soit créé en même temps que le système (création implicite). Dans le second cas, on peut obtenir que la transition pertinente en V ne puisse être déclenchée que par un signal de R.

Les variables de R ne peuvent être visualisées après l'arrêt de R. Toute tentative de visualiser des données serait alors une erreur en LDS. L'utilisateur peut éviter ce problème de deux manières:

— soit en n'utilisant en aucun cas l'arrêt de R;

— soit en s'assurant que V n'ignore pas que R doit s'arrêter et ne fait pas d'autres tentatives de visualiser les données pertinentes.

La première solution présente l'inconvénient pour l'auteur de la mise en oeuvre que R n'a pas libéré les données stockées qu'elle utilisait.

On trouvera dans la figure D-3.10.5 un exemple de variables révélées/visualisées.

D.3.10.3 *Valeurs export´ees/import´ees*

Un processus peut d´eclarer <<exportables>> une ou plusieurs de ses variables, ce qui a pour effet que tous les autres processus <<quel que soit le bloc auquel ils appartiennent>> peuvent importer une copie de la valeur de la variable sur demande. Le processus importateur doit d´eclarer la variable dans sa d´efinition d'importation.

Lorsque le processus exportateur effectue une exportation, la valeur de la variable est copi´ee dans une variable implicite. Un processus d'importation obtient, au moyen de l'expression d'importation, la valeur de cette copie. Ainsi, la valeur obtenue par l'expression d'importation peut diff´erer de la valeur obtenue gr | ce à l'acc`es ordinaire par le propri´etaire, m | me si ces deux op´erations sont effectu´ees au m | me moment.

On trouvera un exemple dans la figure D-3.10.5.

**Figure D-3.10.5, p. 5**

Dans la figure D-3.10.5, le processus P1 est censé être l'ascendant des processus P2 et P3; en conséquence, l'identificateur d'instance de processus des expressions IMPORT et VIEW est indiqué par l'attribut PARENT.

D.3.10.4 *Expressions*

Des expressions d'un processus en LDS peuvent servir de texte formel pour des décisions, options, sélections, t | ches, signaux continus, conditions de validation et construction d'initialisation. Des expressions sont également utilisées comme paramètres réels de la sortie, de l'appel de procédure, de la construction de création. Des expressions PId sont utilisées dans la partie TO de la construction de sortie. Les expressions des constructions d'options et de sélections (évaluées statistiquement) devraient | tre des sortes de données prédéfinies. Dans une expression, il peut y avoir des termes fondamentaux (c'est-à-dire des termes ne contenant que des représentations de valeurs constantes) et des termes comportant des variables.

Le LDS a un ensemble prédéfini d'opérateurs infixes. Ces opérateurs peuvent | tre utilisés pour n'importe quel type de données et ils sont les seuls opérateurs infixes autorisés. Pour ces opérateurs, les règles de priorité sont définies et ne peuvent | tre modifiées. Les opérateurs infixes prédéfinis sont les suivants:

=>, OR, XOR, AND, IN, /=, =, >, <, <=, >=, +, —, //, *, /, MOD, REM

Le LDS offre en outre les opérateurs prédéfinis:

—, NOT

qui sont des opérateurs prédéfinis unaires.

Lorsque l'on utilise des opérateurs prédéfinis, il faut veiller à appliquer les règles de priorité car ces opérateurs, étant prédéfinis indépendamment du domaine d'application, pourraient donner lieu à des confusions.

A titre d'exemple, admettons le newtype et l'expression de la figure D-3.10.6. Compte tenu des règles de priorité de multiplication et d'addition, on évalue l'expression comme A = >((B*C) + D). Mais cela est différent de la priorité de AND et OR et un tel changement peut | tre une cause de confusion. De plus, ce type de changement peut aboutir à des axiomes incohérents et rendre non valable la spécification du LDS.

**Figure D-3.10.6 [T28.100] (à traiter comme tableau MEP), p. 6**

Tous les autres opérateurs définis par l'utilisateur sont des fonctions qui doivent | tre utilisées dans la notation préfixe.

Les deux opérateurs VIEW et IMPORT ont une sémantique particulière, expliquée dans les paragraphes précédents. En tout cas, ils renvoient une valeur d'une certaine sorte, qui peut | tre un autre opérande dans une expression.

D.3.11 *Expresssion du temps en LDS*

La nécessité de mesurer le temps et de demander une temporisation dans un système est satisfaite par des temporisateurs et un ensemble d'opérations exécutées sur ceux-ci.

Dans le modèle LDS, les temporisateurs (<<Timers>>) sont des métaprocessus capables d'émettre des signaux vers le processus sur demande. L'utilisation de temporisateurs doit | tre déclarée dans leur définition, dans le cadre des définitions de processus. Les opérations <<SET>> et <<RESET>> servent à activer les temporisateurs. L'opération SET implique qu’une temporisation doit se produire à un moment sp´ecifi´e et l’op´eration RESET annule la temporisation sp´ecifi´ee. (A noter qu’une op´eration SET comporte implicitement une op´eration RESET pour toute temporisation provenant de ce temporisateur qui n’aurait pas expir´e.)

La construction <<set>> contient l’expression temporelle du d´elai requis, le nom du temporisateur dont il s’agit et, en option, une liste d’expressions. Cette liste sp´ecifie des valeurs qui seront comprises dans le signal du temporisateur de m | me ordre.

Les listes d’expressions peuvent être spécifiées dans la construction <<reset>> pour permettre de réinitialiser une instance particulière de l’instance de temporisateur ayant les mêmes valeurs.

La définition du temporisateur doit comprendre la liste des identificateurs de référence de sortes correspondant aux sortes utilisées dans les expressions de set/reset.

On trouvera un exemple d’instruction <<set>> dans la figure D-3.11.1.

Dans la construction set, il faut spécifier un temps absolu. Le temps relatif est transformé en valeur de temps absolue par l’adjonction de la fonction primitive <<NOW>>, qui représente le moment actuel. L’expression du délai demandé devrait être une expression temporelle. Le temps est une sorte prédéfinie, <<héritée>> de la sorte <<réel>>.

La possibilité de recevoir une temporisation est spécifiée à l’aide du nom de temporisateur dans une entrée, comme indiqué dans la figure D-3.11.1.

**Figure D-3.11.1, p. 7**

Il est possible de définir des synonymes pour indiquer les durées souhaitées. Après avoir choisi les unités de temps et de durée, l’utilisateur peut définir les synonymes nécessaires pour représenter les durées, comme indiqué dans la figure D-3.11.2.

**Figure D-3.11.2 [T29.100] (à traiter comme tableau MEP), p. 8**

Dans la Recommandation, il est indiqué que l'armement d'un temporisateur sur un moment déjà <<écoulé>> est autorisé. Cette décision a été prise afin de faciliter la simulation de systèmes; toutefois, une opération de ce genre pourrait donner une spécification manquant de clarté et devrait donc être évitée.

## D.3.12 *Emploi de qualificatifs*

En LDS, des qualificatifs sont utilisés pour faire référence à des points d'une spécification, lorsque le nom ne détermine pas uniquement un point donné. Naturellement, pour les définitions d'un point, seul le nom doit être spécifié mais lorsque l'on se réfère à ce point en dehors de l'occurrence qui le définit, il peut ête nécessaire de disposer d'un identificateur composé d'un qualificatif et de ce nom.

Cela est valable également pour l'utilisation de définitions différées: l'occurrence définissante utilise le nom pour repérer la définition; la définition différée utilise un nom qualifié pour spécifier son contexte.

Dans la figure D-3.12.1, on trouvera un exemple d'emploi de qualificatifs pour un processus qui peut recevoir deux signaux différents ayant même nom mais des identificateurs différents. La première entrée se rapporte à un type de signal défini au niveau du bloc; la seconde entrée se réfère à un type de signal défini dans la définition de processus. Dans la seconde entrée, la qualification pourrait être omise car lorsque des identificateurs ne sont pas qualifiés, il faut se référer à la définition la plus interne.

**Figure D-3.12.1 [T30.100] (à traiter comme tableau MEP), p. 9**

## D.3.13 *Syntaxe de noms*

En LDS, des noms peuvent consister soit en un mot unique soit en une liste de mots séparés par des délimiteurs (espaces ou caractères de contrôle). Cette seconde possibilité permet d'obtenir une spécification plus lisible lorsque l'on utilise des noms d'une certaine longueur, particulièrement en LDS/GR, car les symboles graphiques ont une taille limitée. On trouvera dans la figure D-3.13.1 certains exemples de noms composés de plusieurs mots.

**Figure D-3.13.1, p. 10**

Chaque fois que l'emploi de plusieurs mots dans un nom donne lieu à une ambigu"it´e (voir les exemples de la figure D.3.13.2), les d´elimiteurs entre deux mots doivent | tre remplac´es par un caract`ere de soulignement (<< _>>). La cha | ne de caract`eres obtenue par la transformation de d´elimiteurs en soulignements d´esigne toujours le m | me nom; ainsi, par exemple, BUSY SUB d´esigne le m | me nom que BUSY _SUB. On trouvera dans la figure D-3.13.3 certains exemples d'emploi sans ambigu"it´e de noms.

**Figure D-3.13.2 [T31.100] (a traiter comme tableau MEP), p. 11**

**Figure D-3.13.3 [T32.100] (à traiter comme tableau MEP), p. 12**

Il importe de noter que le caractère de soulignement peut aussi être employé dans des noms comme caractère de continuation, pour permettre de répartir des noms sur plus d'une ligne. Dans ce cas, on peut aussi désigner un nom contenant un caractère de soulignement suivi par un ou plusieurs délimiteurs en éliminant le soulignement et les délimiteurs.

On trouvera dans la figure D-3.13.4 des exemples de désignations différentes pour le même nom.

**Figure D-3.13.4 [T33.100] (à traiter comme tableau MEP), p. 13**

## D.4 *Structuration et affinage de systèmes LDS*

### D.4.1 *Considérations générales*

Le présent chapitre traite de certaines techniques et de constructions du LDS qui permettent une spécification descendante des systèmes de grande taille. En général, les termes <<subdivision>> et <<affinage>> sont appliqués à ces techniques avec les significations suivantes:

— subdivision: il s'agit de diviser une partie du système en parties plus petites dont le comportement global est équivalent à la partie non subdivisée. Ce terme peut s'appliquer à des blocs (structurés en nouveaux sous-blocs, canaux et sous-canaux), à des canaux (structurés en blocs, nouveaux canaux et sous-canaux) et à des processus (structurés en service);

— affinage: adjonction de nouveaux détails aux fonctions d'un système. Considérée à partir de l'environnement, l'affinage d'un système a pour résultat un enrichissement de son comportement car plusieurs types de signaux et d'informations peuvent être traités.

On notera que la structure interne d'une partie d'un système renseigne plus précisément sur la structure, mais pas nécessairement sur le comportement du système. Il est théoriquement possible de distinguer l'aspect d'une représentation plus détaillée du comportement (par exemple, le traitement d'un nouveau signal) de l'aspect d'une structure plus détaillée (couvrant, par exemple, à la fois la structure des parties et celle du comportement du système), mais dans la pratique ces deux aspects sont généralement confondus: ainsi, des détails supplémentaires sur la structure du système en éclairent également le comportement.

La structure minimale d'un système en LDS est décrite dans le chapitre 2 de la Recommandation: un système consiste en un ensemble de blocs reliés par des canaux et ces blocs contiennent des processus.

Les concepts applicables à la subdivision en plusieurs niveaux de détails et d'affinage signaux sont traités dans le chapitre 3 de la Recommandation. Ces concepts ne sont pas nécessaires dans le cas de systèmes qui n'ont pas à être affinés.

### D.4.2 *Critères de subdivision*

On nomme subdivision la technique qui consiste à fragmenter une représentation d'un système d'un niveau de détail élevé en éléments d'un maniement facile. Le processus de subdivision ajoute une structure à un système.

Parmi les critères qui entraînent la subdivision de représentation d'un système, on peut citer les suivants:

a) définir des blocs ou des processus qui, par leurs dimensions, facilitent la compréhension du système;

b) établir une certaine correspondance avec les divisions effectives du logiciel et/ou du matériel;

c) utiliser les subdivisions fonctionnelles naturelles;

d) réduire au minimum l'interaction entre les blocs;

e) réutiliser des représentations préexistantes (par exemple un système de signalisation).

Les critères qui seront effectivement adoptés dépendront de plusieurs facteurs et notamment du degré de précision requis.

Etant donné que la relation entre les niveaux dépend des critères de subdivision choisis, il importe de spécifier clairement ces critères afin de faciliter la compréhension de la représentation. L'usager décide des critères de subdivision mais certaines restrictions s'appliquent afin de garantir une représentation correcte en LDS. Les paragraphes qui suivent traitent de ces restrictions.

### D.4.3 *Subdivision des blocs*

On peut subdiviser un bloc en un ensemble de blocs et de canaux pratiquement de la même façon qu'on subdivise un système en blocs et en canaux. En LDS/GR, cela est représenté à l'aide d'un diagramme de sous-structure de bloc. On en trouvera dans la figure D-4.3.1. Dans le premier diagramme de cette figure, le diagramme du bloc B1 contient une référence à sa sous-structure. Dans le second diagramme, il existe un diagramme de sous-structure pour le bloc B1. Le symbole de bloc rattaché au canal C2 par une ligne en taits discontinus, dans le premier diagramme, représente une référence à la sous-structure de canal C2 (voir le § D.4.5).

En LDS/PR, la subdivision d'un bloc est représentée par un ensemble de définitions comprises entre les mots-clés SUBSTRUCTURE et ENDSUBSTRUCTURE à l'intérieur de la définition du bloc. Les définitions d'une sous-structure de bloc sont les mêmes que celles de la définition de système; de plus, la spécification des connexions entre canaux et sous-canaux doit être effectuée comme indiqué dans la figure D-4.3.2.

**Figure D-4.3.1, p. 14**

**Figure D-4.3.2 [T34.100] (a traiter comme tableau MEP), p. 15**

La sp´ecification des processus et celle d'une sous-structure de bloc peuvent coexister à l'int´erieur d'une d´efinition de bloc. Dans ce cas, les processus du bloc repr´esentent le comportement de ce bloc pour un niveau de pr´ecision donn´e; d'autres processus internes des d´efinitions des sous-blocs repr´esenteront d'une mani`ere plus d´etaill´ee le m | me comportement. On trouvera au § D.4.7 (figure D-4.7.1) un exemple de bloc d´ecrit tant en fonction de processus que de sous-structures.

S'il n'existe pas de processus à l'int´erieur d'un bloc mais seulement une sous-structure de bloc et que celle-ci ne fait pas l'objet d'un rep´erage, il existe en LDS/GR une notation abr´eg´ee permettant de simplifier le diagramme. Cette notation permet d'embo | ter des blocs en consid´erant le cadre du bloc ext´erieur comme impliquant le cadre de sous-structure. Gr | ce à cette notation, l'exemple de la figure D-4.3.1 peut | tre trac´e comme dans la figure D-4.3.3.

**Figure D-4.3.3, p. 16**

Chaque bloc provenant de la subdivision d’un bloc peut | tre lui-m | me subdivis´e, ce qui donne une structure arborescente hi´erarchique de blocs et de leurs sous-blocs. Un diagramme auxiliaire appel´e <<diagramme d’arbre de bloc>> repr´esentant cette structure g´en´erale est d´ecrit au § D.4.4.

A moins qu’il ne s’agisse de l’affinage de signaux, les r`egles suivantes doivent | tre observ´eesdans le processus de subdivision.

1) les listes de signaux des sous-canaux reli´es à un canal d’entr´ee ne doivent pas comporter de nouveaux signaux; ces listes de signaux doivent comprendre tous les signaux de la liste du canal initial (pour les canaux bidirectionnels, il faut tenir compte de la liste des trajets entrants). Ainsi, dans l’exemple de la figure D-4.3.1, C2.1 et C2.2 transportent sur les trajets d’entr´ee tous les signaux de L2. En outre, aucun des signaux transport´es par C2.1 ne peut appara | tre sur le trajet d’entr´ee de C2.2;

2) les listes de signaux des sous-canaux (par exemple C1.1 et C1.2) reli´es à une voie de sortie (par exemple C1) ne peuvent comporter de nouveaux noms de signaux; ces listes de signaux doivent comprendre tous les noms de signaux du canal initial. Ainsi, L1.1 et L1.2 contiennent tous les noms de signaux de L1. Les listes L1.1 et L1.2 peuvent contenir les m | mes identificateurs de signaux;

3) si le bloc original contient des processus, il existe deux options. Tout d’abord, une copie de chaque processus peut | tre red´efinie dans l’un ou l’autre des nouveaux sous-blocs. Deuxi`emement, de nouveaux processus peuvent | tre d´efinis dans les sous-blocs de telle mani`ere que l’interface reste sans changement;

4) des d´efinitions de donn´ees du bloc ascendant se trouvent dans ses sous-blocs, de sorte que chacun de ceux-ci peut utiliser un type de donn´ees d´efini dans le bloc ascendant sans avoir à le red´efinir;

5) si un type de donn´ees d´efini dans le bloc ascendant est red´efini avec le m | me nom dans un sous-bloc, la nouvelle d´efinition s’applique au sous-bloc d´efinissant tandis que l’ancienne d´efinition reste valable pour les autres sous-blocs. Il faut ´eviter une red´efinition servant uniquement à caract´eriser un sous-bloc car un lecteur peut la n´egliger, supposant que l’ancienne d´efinition est valable. Dans certains cas cependant, il convient de proc´eder à une telle red´efinition, notamment lorsqu’il s’agit d’un affinage du comportement. Il faut veiller à souligner cette red´efinition par une annotation appropri´ee.

### D.4.4 *Diagramme d’arbre de blocs*

Le diagramme d’arbre de blocs repr´esente la structure d’un syst`eme en fonction de ces blocs et sous-blocs. Ce diagramme vise à donner au lecteur un aper,cu de la structure totale d’un sous-syst`eme.

Le diagramme est un arbre hi´erarchique de symboles de blocs et de lignes de subdivision comme indiqu´e dans la figure D-4.4.1.

**Figure D-4.4.1, p. 17**

Il est préférable que le diagramme soit tracé de telle manière que tous les symboles de blocs aient la même dimension. Ainsi, les blocs du même niveau de subdivision apparaissent comme à un niveau uniforme dans le diagramme. Si le diagramme est trop grand pour tenir sur une seule page, il doit être divisé en diagrammes d'arbre de blocs <<partiels>> comme indiqué dans la figure D-4.4.2.

Il est souvent utile de subdiviser un diagramme d'arbre de blocs en diagrammes partiels.

La division en plusieurs diagrammes partiels s'effectue de telle sorte que le premier diagramme, ayant pour racine le système, soit coupé de manière que les blocs encore subdivisés apparaissent comme non subdivisés. Les blocs, là où le diagramme original a été coupé, apparaissent comme des racines dans les diagrammes indiquant la subdivision.

**Figure D-4.4.2, p. 18**

Si des diagrammes partiels sont utilisés et qu'il n'est pas évident de savoir si un bloc est subdivisé ailleurs et de trouver où continuent les diagrammes, il convient d'insérer des références en utilisant le symbole commentaire.

## D.4.5 *Subdivision des canaux*

Un canal peut être subdivisé indépendamment des blocs qu'il relie. Cela permet la représentation du comportement du canal en question lorsqu'il achemine des signaux. Il est parfois nécessaire de représenter le comportement du canal afin d'obtenir une représentation exacte du parcours d'un signal.

Pour ce faire, on examine le canal en tant qu'élément indépendant dont l'environnement se compose des deux blocs qu'il relie. Considérant le canal de cette manière, on peut exprimer sa structure au moyen de blocs, de canaux et de processus.

En LDS/GR, la subdivision de canaux est représentée à l'aide de diagrammes de sous-structure de canaux, comme indiqué dans la figure D-4.5.1. (l'exemple représente la sous-structure du canal C2 de la figure D-4.3.1).

Un diagramme de sous-structure de canaux montre comment un canal est subdivisé en sous-composants. Ce diagramme ressemble au diagramme de système (sauf en ce qui concerne la connexion des blocs). Toutes les directives données au § D.4.3 sont valables pour le diagramme de sous-structure de canaux.

Dans le diagramme d'interaction de blocs oὺ apparaissent les canaux subdivis´es, il faut faire r´ef´erence au diagramme de sous-structure de canaux d´ecrivant la subdivision.

**Figure D-4.5.1, p. 19**

En LDS/PR, la forme est semblable à celle de la d´efinition de sous-structure de blocs; la seule diff´erence est que, dans l'instruction CONNECT, les sous-canaux d'extr´emit´e sont raccord´es à des blocs ext´erieurs (B1 et B2 dans la figure D-4.5.2) et non à des canaux ext´erieurs.

L'import/export des valeurs est autoris´e entre un bloc et une sous-structure de canaux. Cela permet une repr´esentation directe du mod`ele OSI dans laquelle les communications de couches homologues sont mod´elis´ees par l'´echange de signaux et les communications de couches contigu"es par des valeurs partag´ees.

D.4.6 *Repr´esentation du syst`eme en cas de subdivision*

Lorsqu'un syst`eme est repr´esent´e sous la forme d'un ensemble de blocs interconnect´es par des canaux et que l'on exprime le comportement de chaque bloc au moyen d'un ou de plusieur processus, nous nous trouvons en pr´esence d'une repr´esentation sur un seul niveau. Ceci revient à dire que nous pouvons voir tous les ´el´ements de la repr´esentation au m | me niveau. Nous

introduisons, avec la subdivision une relation hi´erarchique entre les diff´erents documents. Le document qui en r´esulte repr´esente la structure du syst`eme lorsque le syst`eme se compose de n blocs.

Un autre document peut pr´esenter le syst`eme compos´e d'un ensemble diff´erent de blocs, dont certains proviennent des blocs du document pr´ec´edent (certains blocs du document pr´ec´edent ont ´et´e remplac´es par les sous-blocs r´esultant de la subdivision des blocs). Ce nouveau document doit | tre rapport´e au pr´ec´edent document.

Mais pour obtenir une repr´esentation compl`ete du syst`eme, il ne suffit pas d'´etablir des rapports entre les documents, il faut ´egalement les organiser de telle sorte qu'il soit possible d'acc´eder à la repr´esentation du syst`eme par niveaux, en commen,cant par une vue d'ensemble g´en´erale pour passer ensuite à des repr´esentations de plus en plus d´etaill´ees. Ceci exige le regroupement des diff´erents documents afin de repr´esenter le syst`eme à diff´erents niveaux.

On ne devrait pas trouver les m | mes ´el´ements à tous les niveaux. La repr´esentation du syst`eme au premier niveau peut se composer de repr´esentations des blocs et des canaux, à l'exclusion des processus qui d´ecrivent le comportement de chaque bloc. A un niveau inf´erieur, on peut souhaiter inclure la représentation du comportement de certains blocs, mais pas de celui d'autres blocs. Le niveau de représentation le plus bas (c'est-à-dire la représentation la plus détaillée) devrait représenter intégralement les comportements de tous les blocs, c'est-à-dire l'ensemble complet des processus qui expriment ce comportement.

**Figure D-4.5.2 [T35.100] (à traiter comme tableau MEP), p. 20**

L’observation de l’arborescence de blocs soulève plusieurs réflexions.

En premier lieu, l’arborescence a toujours une et seulement une racine, qui est le système. Il en est ainsi de même lorsque le système représenté se compose de plusieurs blocs depuis le début. Dans l’arborescence, ces blocs sont représentés au niveau 1. Le nom du système peut suffire pour constituer le bloc qui sert de racine. On peut appliquer les définitions des canaux aux blocs bien que cela ne soit pas exigé, sauf si les blocs ont des processus associés.

L’observation des feuilles de cette arborescence inspire une seconde remarque: elles ne sont pas toutes au même niveau. Ceci peut être dû à un nombre inégal de subdivisions sur les blocs de l’arborescence. Le nombre de subdivisions dépend de plusieurs facteurs, dont la plupart relèvent de l’appréciation subjective de la personne chargée d’élaborer les spécifications, et du concepteur.

Pour que des blocs feuille ne soient pas subdivisés de nouveau, le LDS n’exige qu’une chose, à savoir que leur comportement soit entièrement spécifié (c’est-à-dire que son attitude puisse être représentée par les définitions des processus). Par conséquent, un bloc feuille doit contenir au moins un processus associé.

Lorsqu'un syst`eme est repr´esent´e à plusieurs niveaux d'abstraction, nous pouvons repr´esenter le syst`eme au niveau de notre choix. Le choix d'un niveau donn´e exige que les blocs soient examin´es à ce m | me niveau, que leurs processus associ´es et tous les blocs feuille soient examin´es à des niveaux sup´erieurs avec leurs processus associ´es (voir la figure D-4.6.1).

Il est souvent pratique de choisir diff´erents niveaux à diff´erentes fins, par exemple un niveau <<d'aper,cu>> pour la pr´esentation et un niveau plus d´etaill´e pour la mise en oeuvre.

**Figure D-4.6.1, p. 21**

La repr´esentation d'un syst`eme à un certain niveau peut | tre incompl`ete dans la mesure oǜ certains des blocs de ce niveau n'ont pas de processus associ´es.

On parle de niveaux de repr´esentation et de s´election d'un certain niveau de repr´esentation pour les raisons suivantes:

— il se peut que notre repr´esentation ait atteint un certain <<niveau>> de pr´ecision, repr´esent´e par ce niveau de repr´esentation (dans ce cas, les blocs de ce niveau sont des blocs feuille; ils ne sont pas complets car ils demandent un suppl´ement de travail!);

— nous voulons consid´erer la repr´esentation du syst`eme à un certain niveau de pr´ecision; nous choisissons donc le niveau de repr´esentation qui correspond au mieux aux abstractions que nous cherchons. On notera que dans certains cas un niveau donn´e de repr´esentation peut | tre constitu´e de documents à diff´erents niveaux d'abstraction. La repr´esentation d'une partie du syst`eme au niveau 2 peut | tre tr`es d´etaill´ee, tandis que celle d'une autre partie au niveau 4 peut demeurer abstraite. Cela signifie qu'en choisissant une repr´esentation au niveau 3, l'on peut obtenir une repr´esentation tr`es d´etaill´ee de certaines parties et une simple vue d'ensemble d'autres parties;

— la m´ethodologie de repr´esentation et conception peut permettre à chaque niveau d'avoir une signification pr´ecise. Ainsi le niveau 1 correspond à la sp´ecification, le niveau 2 fournit la structure g´en´erale du syst`eme, le niveau 3 la structure des modules (<<racks>>, fonctions de logiciel), le niveau 4 donne la structure d´etaill´ee (planches imprim´ees, proc´edures, modules de logiciel). Dans ce cas, le lecteur choisit un niveau donn´e en fonction de son propre besoin, et l'on notera que la m´ethode permet d'´eviter les diff´erences entre les niveaux de pr´ecision des parties qui constituent un niveau donn´e de repr´esentation.

D.4.6.1 *Sous-ensemble de subdivision cohérent*

En plus de la spécification totale du système et de celle qui est donnée par niveaux, le LDS comporte une notion de <<sous-ensemble de système cohérent>>. Celle-ci peut être considérée comme une spécification de niveau unique dans laquelle tous les blocs peuvent être pris d'un niveau quelconque de la structure d'un système, étant entendu:

— qu'un bloc peut être choisi pour faire partie de la représentation cohérente du système s'il peut être considéré comme un bloc feuille (ou bien il est un bloc feuille ou bien tous les processus nécessaires à la représentation de son comportement lui sont associés);

— que, si un bloc est choisi, tous les blocs obtenus par la subdivision de son bloc ascendant doivent être compris, soit directement soit par l'inclusion de leurs descendants;

— que tous les documents définissant les signaux passant dans le canal qui relie un bloc de la représentation doivent être fournis. Selon la stratégie de subdivision choisie, cela peut impliquer qu'une fois qu'un bloc a été pris, son ascendant doit aussi l'être, au moins en ce qui concerne les parties définissant les données et les signaux.

Dans les cas où certains canaux ont été subdivisés,chacun étant considéré comme un système, il faut fournir la spécificationde chacun de ces <<systèmes>>. Leur spécification consiste en documents du même type que ceux de la représentation de systèmes usuels. Il convient d'ajouter des notations référant ces systèmes à la spécification du système principal auquel ils appartiennent. Ainsi, en un sens, on peut considérer ces canaux subdivisés

comme des systèmes internes. Chacun de ces systèmes peut avoir plusieurs niveaux de représentation et comportera en outre des systèmes internes si certains des canaux contenus sont subdivisés plus avant.

D.4.7 *Affinage*

Le mécanisme d'affinage a été introduit dans le LDS pour <<cacher>> les signaux de niveau inférieur au niveau supérieur d'abstraction et permettre la spécification descendante du comportement du système.

L'affinage permet à l'utilisateur de subdiviser des signaux en sous-signaux, ce qui donne une structure hiérarchique comme dans le cas des blocs et des sous-blocs. Cela signifie qu'à l'intérieur d'une définition de signal, il est possible de définir un ensemble de nouveaux signaux qui sont appelés signaux affinés, ou sous-signaux du signal défini. De même que l'arbre de bloc pour une définition de bloc, on peut tracer, pour plus de clarté, un <<arbre>> de signaux pour une définition de signal.

L'affinage est étroitement lié à la subdivision de bloc car seuls les signaux transportés par un canal relié à un bloc subdivisé peuvent être affinés. Cela signifie qu'un signal figurant dans la liste d'un canal peut
être remplacé par ses sous-signaux lors de la subdivision du bloc qui y est connecté. Les sous-canaux correspondants générés par la subdivision du bloc devront spécifier des sous-signaux dans leurs listes de signaux.

Lorsqu'un signal est défini comme devant être acheminé par un canal, le canal transportera automatiquement tous les sous-signaux de ce signal, même si certains des sous-signaux se dirigent dans le sens opposé (dans ce cas, le canal est considéré comme implicitement bidirectionnel). On trouvera dans la figure D-4.7.1 un exemple d'affinage. Cet exemple représente un système dans lequel un processus d'un bloc émet des fichiers de textes vers un processus d'un autre bloc. Au niveau d'affinage le plus élevé, on obtient ceci par l'émission de signaux qui représentent chacun un fichier de textes (signal sf). Au niveau d'affinage inférieur, on peut spécifier que le fichier de textes se compose d'un certain nombre d'enregistrements émis par un signal (signal sr) et que le récepteur doit répondre (signal nr) après absorption de chaque enregistrement. L'émetteur enverra un signal fin de fichier à la fin de l'opération. Dans cet exemple, au niveau d'affinage le plus élevé, les processus de B1 et de B2 communiquent en utilisant le signal sf; au niveau immédiatement inférieur, les processus de B11 et de B21 communiquent en utilisant les signaux sr, nr et eof.

Voici certains cas réels dans lesquels la notion d'affichage peut s'appliquer:

— transformation de nom: un signal affiné devient un autre signal portant un nom différent. C'est une transformation d'élément à élément dans laquelle seul le nom du signal change. Cette possibilité est généralement impliquée lorsque l'on veut que chaque niveau soit entièrement compréhensible en soi-même. (Il peut être commode d'adapter le nom d'un signal à son contexte.);

— transformation par division: il s’agit d’une transformation d’un signal en plusieurs autres, dans laquelle un signal est divisé en plusieurs signaux par souci de précision. A titre d’exemple, un signal générique <<Alarme>> est divisé en <<Register _Alarm>>, <<Central _Processor _Alarm>> et <<Subscriber _Alarm>>;

— transformation avec algorithme: le signal originel est transformé en un ensemble de signaux qui déclenchent un algorithme afin de fournir l’information originelle.

**Figure D-4.7.1, p. 22**

D.4.7.1 *Sous-ensemble d'affinage coh´erent*

Lorsque l'affinage est appliqu´e à une sp´ecification de syst`eme, la notion de sous-ensemble de subdivision coh´erent est limit´ee de mani`ere à ´eviter des communications avec diff´erents signaux entre diff´erents niveaux d'affinage. Dans ce cas, on dit que la d´efinition de syst`eme contient plusieurs sous-ensembles d'affinage coh´erents. Si un sous-ensemble d'affinage coh´erent contient un processus communiquant avec des sous-signaux, ce processus ne peut communiquer avec le signal ascendant et l'autre extr´emit´e de la liaison de communication doit communiquer avec les m | mes sous-signaux.

D.4.7.2 *Transformation entre signaux et sous-signaux*

L'utilisateur peut avoir besoin de d´ecrire la transformation entre diff´erents niveaux d'affichage à des fins de simulation ou pour contr | ler le comportement du syst`eme indiff´erent au niveau de pr´ecision. Il peut le faire de mani`ere informelle à l'aide de processus LDS suppl´ementaire d´ecrivant la transformation dynamique d'un signal en ses sous-signaux ou vice versa.

Dans la figure D-4.7.2, deux processus sont introduits pour décrire la transformation appliquée dans la figure D-4.7.1. Un processus d'affinage définit la manière dont le signal de niveau élevé est <<affiné>> en un ensemble de signaux au niveau inférieur suivant. Un processus d'extraction décrit la transformation inverse.

Figure D-4.7.2, p. 23

## D.5 *Concepts supplémentaires*

### D.5.1 *Macros*

La construction macro permet de traiter les répétitions et/ou de structurer une description. Elle comprend une définition de macro contenant une partie de spécification LDS qui peut être référencée (appel de macro) en d'autres endroits d'une spécification de système.

La définition de macro peut être donnée en tous les endroits où des définitions de données sont autorisées. Cependant, le nom de macro n'a pas de portée. Ainsi, une macro définie dans un bloc peut être référencée en dehors de celui-ci.

En LDS/PR, il est possible d'employer une définition de macro pour remplacer toute séquence d'unités lexicales. Cette possibilité diffère des définitions de macro en LDS/GR qui ne peuvent remplacer que des collections d'unités syntaxiques.

Pour mettre en concordance des documents en LDS/GR et en LDS/PR contenant des macros, il faut appliquer les restrictions ci-après à l'utilisation de macros LDS/PR:

1) Une macro ne peut remplacer qu'une ou plusieurs des constructions syntaxiques suivantes (qui correspondent à des symboles LDS/GR):

départ

état

entrée

condition de validation

signal continu

mise en réserve

instruction d'action

instruction de terminaison

2) Des paramètres formels de macros ne peuvent pas être utilisés en des endroits déterminant le type de la construction en LDS/PR. Cela signifie qu'ils ne peuvent être utilisés là où sont employés les mots clés du LDS/PR. De même, des paramètres réels ne devraient pas contenir de mot clé correspondant à des symboles du LDS/GR: START, STATE, PROCEDURE, INPUT, TASK, OUTPUT, DECISION, CREATE, STOP, PROVIDED, CALL, COMMENT, JOIN, RETURN, SAVE ou OPTION.

3) Chaque énoncé de la définition de macro doit pouvoir être atteint à partir d'au moins un accès d'entrée de macro.

En LDS/PR, la macro a toujours au plus un accès d'entrée et un accès de sortie, de sorte qu'il est nécessaire d'employer des étiquettes et des liaisons pour représenter en LDS/PR une macro LDS/GR ayant plus d'un accès d'entrée ou de sortie. Si la macro doit être appelée en plus d'un endroit, les étiquettes devront être communiquées sous la forme de paramètres.

En LDS/GR, il y a deux moyens différents de représenter les accès d'entrée et de sortie d'une définition de macro. L'utilisateur peut soit tracer un cadre pour la macro et relier à celui-ci les accès d'entrée et de sortie, soit utiliser explicitement des symboles d'accès d'entrée et d'accès de sortie (le cadre de macro étant facultatif).

L'exemple de la figure D-5.1.1 illustre en LDS/GR et en LDS/PR une macro ayant deux accès d'entrée et deux accès de sortie. (Dans cet exemple, les accès d'entrée et de sortie sont reliés au cadre de macro.)

Cela signifie que dans la spécification principale en LDS/PR (figure D-5.1.2), existent les branchements et les étiquettes correspondants. A noter que l'étiquette sera probablement différente pour chaque appel de macro.

La figure D-5.1.3 illustre deux définitions de macro qui définissent un mécanisme de synchronisation. A noter l'emploi du paramètre pseudo-formel MACROID, qui sert à rendre uniques des noms d'état. Le même exemple est repris dans la figure D-5.1.4 en ce qui concerne l'emploi de symboles d'accès d'entrée et d'accès de sortie.

Dans le cas de macros emboîtées, c'est-à-dire lorsqu'il existe un ou plusieurs appels de macros à l'intérieur d'une définition de macro, il convient de noter que l'expansion des macros extérieures n'affecte pas celle des macros intérieures. Plus précisément, l'expansion de macros emboîtées doit être considérée comme si l'expansion d'une macro devrait être terminée avant le début de l'expansion d'appels éventuels de macros intérieures.

**Figure D-5.1.1, p. 24**

**Figure D-5.1.2, p. 25**

**Figure D-5.1.3, p. 26**

**Figure D-5.1.4, p. 27**

## D.5.2 *Systèmes génériques*

En LDS, il est possible de définir différents systèmes dans une seule spécification à l'aide des paramètres de systèmes. Ceux-ci ont une valeur indéfinie qui peut être donnée extérieurement pour obtenir une définition de système spécifique correspondant aux besoins des utilisateurs.

En fait, les paramètres de système sont des synonymes externes et peuvent être utilisés à tout endroit où un synonyme peut l'être. Naturellement, avant d'interpréter un système, il faut affecter une valeur à tous les synonymes externes. La figure D-5.2.1 donne quelques exemples valables de l'utilisation de synonymes externes.

**Figure D-5.2.1 [T36.100] (a traiter comme tableau MEP), p. 28**

Le LDS offre deux constructions complémentaires qui permettent des choix plus puissants conditionnés par des synonymes externes:

— construction SELECT: sélection conditionnelle d'une partie de spécification. La condition est spécifiée par une expression booléenne que l'on doit pouvoir évaluer statiquement, avant l'interprétation du système. Elle est choisie lorsque l'expression est VRAIE;

— construction ALTERNATIVE: permet la sélection conditionnelle d'une partie de spécification, entre deux options ou davantage. Elle ne peut être utilisée que pour choisir différentes transitions dans le corps de processus, de procédures ou de services.

## D.5.3 *Services*

### D.5.3.1 *Considérations générales*

La notion de service vise à offrir la possibilité de subdiviser une définition de processus sans introduire de parallélisme. Chaque service peut être considéré comme une <<fonction>> offerte par le processus. C'est une définition de processus partielle représentant un <<sous-comportement>> du processus. Ce <<sous-comportement>> constitue un élément du comportement du processus global. En conséquence, l'emploi du concept de service est une manière de structurer un processus.

Il y a de nombreuses raisons de structurer un processus, par exemple en vue de gérer la complexité et d'améliorer la lisibilité. Mais la subdivision constitue aussi un moyen d'isoler certaines parties d'un processus et de les décrire séparément. Ces parties peuvent être des «sous-parties» d'une fonction offerte par le système. Ainsi, grâce au concept de service, on peut isoler une fonction de système en décrivant un ensemble de services subordonnés dans un ou plusieurs processus.

**Figure D-5.3.1, p. 29**

A noter que le langage LDS ne possède pas de facilités permettant de composer une fonction de système en choisissant des services de plusieurs processus. C'est à l'utilisateur qu'il appartient de procéder à cette composition, entièrement en dehors du langage.

Un service, étant isolé d'autres services et décrit comme une machine à état fini dans son propre espace d'état, peut être développé et modifié sans que cela n'affecte d'autres services. Toutefois, il convient de relever que les services d'un processus ont souvent des données communes, ce qui signifie qu'ils peuvent influer les uns sur les autres par suite de manipulations de données.

Le comportement d'un processus n'était pas modifié lorsqu'il est subdivisé en services, les services représentent seulement une autre description du même comportement. On peut naturellement déciderau moment de la conception, de modéliser le comportement en plusieurs processus au lieu d'un seul. Toutefois, cela donnerait un comportement différent car plusieurs processus fonctionnent en parallèle et ne peuvent partager (lire et écrire) des données sans un échange de signaux compliqué.

Il convient de noter que la structuration d'un processus en services ne signifie pas que ce processus disparaîtra entièrement. Cela signifie seulement que le corps de processus, décrivant le comportement du processus, a été entièrement remplacé par plusieurs corps de service. Il restera au niveau du processus les paramètres formels, des déclarations et des définitions formelles. En plus de ceux-ci, certaines définitions et déclarations locales peuvent être reprises dans les définitions de service.

Une définition de service se compose des éléments suivants, dont certains sont facultatifs:

— nom de service;

— ensemble de signaux d'entrée valides: liste d'identificateurs de signaux définissant des signaux qui peuvent être reçus par le service;

— définitions de procédure: spécificationdes procédures qui sont locales au service. On peut aussi utiliser une référence de procédure;

— définitions de données: spécification de newtypes, syntypes et générateurs définis par l'utilisateur et locaux au service;

— définitions variables: déclaration de variables, locales au service. Ces variabls ne peuvent être révélées ou exportées vers d'autres processus (les mots clés REVEALED et EXPORTED ne sont pas admis). Pour chaque variable déclarée, il faut spécifier l'identificateur de sa sorte. Une valeur initiale peut être spécifiée en option;

— d´efinitions de visibilit´e: d´eclaration des identificateurs de variables qui peuvent servir à obtenir les valeurs des variables appartenant à d’autres processus. Pour chaque identificateur de variables, la sorte de variable doit | tre sp´ecifi´ee;

— d´efinitions d’import: sp´ecification d’identificateurs de variables appartenant à d’autres processus, que le service d´esire importer. Pour chaque identificateur, la sorte de variable doit | tre sp´ecifi´ee;

— définitions de temporisateur: voir le § D.3.11;

— définitions de macro: voir le § D.5.1;

— corps de service: spécification du comportement réel du service en ce qui concerne les états, les entrées, ls sorties, les tâches, etc. Comparé à un corps de processus, un corps de service peut aussi contenir l'émission et la réception de signaux prioritaires (§ D.5.3.2).

En LDS/GR, une définition de service est représentée à l'aide d'un diagramme de service. Celui-ci comprend les éléments suivants:

— un symbole de cadre facultatif: symbole de forme rectangulaire qui contient tous les autres symboles;

— l'en-tête de service: le mot clé SERVICE suivi par l'identificateur de service. L'en-tête de service est placé dans l'angle supérieur gauche du cadre;

— une numérotation de page facultative (placée dans l'angle supérieur droit);

— des symboles de texte: un symbole de texte est utilisé pour contenir des définitions de données, de variables, de visibilité, d'import et de temporisateur qui sont locales au service;

— références de procédure: symbole de procédure contenant un nom de procédure représentant une procédure, locale au service et définie séparément;

— diagramme de macro: voir les directives du § D.5.1;

— graphe de service: spécification du comportement réel du service en ce qui concerne les états, les entrées, les sorties, les tâches, etc. Comparé au graphe de processus, un graphe de service peut aussi contenir l'émission et la réception de signaux prioritaires (§ D.5.3.2).

Un exemple de concept de service est donné dans la figure D-5.3.2 pour le processus simple <<temporisateur>> (Timer). Ce processus est structuré en deux services qui sont repérés et définis en deux diagrammes de service (voir la figure D-5.3.3).

**Figure D-5.3.2, p. 30**

L'acheminement du signal <<IR6>> transportant un signal prioritaire (§ D.5.3.2) constitue une interface interne entre les deux services.

**Figure D-5.3.3, p. 31**

Il convient de noter que, les actions (sorties) étant comrpises dans la transition de départ dans le premier service, aucune action n'est autorisée dans la transition de départ du second service.

On trouvera un autre exemple du concept de service au § D.10.

### D.5.3.2 *Signaux prioritaires*

Un signal prioritaire est employ´e pour la communication entre deux transitions dans des services diff´erents d'un processus lorsqu'aucun signal ext´erieur (provenant d'autres processus) ne peut | tre absorb´e dans le laps de temps entre les transitions. Ainsi, les transitions sont <<concat´en´ees>>.

La figure D-5.3.4 est une illustration de la concat´enation des transitions lorsque des signaux prioritaires sont utilis´es.

**Figure D-5.3.4, p. 32**

La construction permettant d'obtenir des signaux prioritaires utilisant la file d'atente d'entr´ee ordinaire est d´ecrite dans la Recommandation. Les signaux prioritaires, ´emis vers un ´etat pr´eliminaire, peuvent | tre trait´es comme des signaux ordinaires et provoquent des transitions vers l'´etat principal (voir aussi le § D.5.3.3.1).

L’exemple suivant est une illustration des cons´equences de l’emploi de signaux prioritaires. L’exemple est expliqu´e sans l’emploi du mod`ele <<´etat pr´eliminaire principal>>.

Le diagramme de s´equencement (voir la figure D-5.3.5) est obtenu de l’interaction entre les services du processus <<SUBSCRIBER _LINE>> d´ecrit au § D.10.2.

**Figure D-5.3.5 et l´egende, p. 33**

Le tableau contient à la fois les signaux en provenance ou à destination de l’environnement du processus et les signaux prioritaires entre les services. Les fl`eches de signaux (de haut en bas) indique comment les signaux sont plac´es dans la file d’attente. Les lignes verticales en traits ´epais indiquent comment l’ex´ecution des transitions s’ordonne dans le temps.

La s´equence d´ebute avec le signal <<CONGESTION>> en provenance du registre, ce qui signifie que l’appel doit | tre rejet´e. Cela signifie aussi le rejet imm´ediat de l’appel suivant.

La figure montre qu’une transition, activ´ee par un signal prioritaire, par exemple <<RESERVE _FOR _MEASUREMENT>>, commence avant l’activation d’une transition par un signal ordinaire, par exemple <<A _ON _HOOK>>, malgr´e l’ordre inverse de celui de sa mise en file d’attente. La transition activ´ee par le signal <<RESERVE _FOR _MEASUREMENT>> d´econnecte la ligne d’abonn´e, ce qui signifie que l’appel suivant (signal <<A _OFF _HOOK>>) sera imm´ediatement rejet´e.

Dans le diagramme de séquencement ci-après, nous avons la même situation mais le diagramme n'utilise pas de signaux prioritaires. Il y a interfonctionnement entre les services et les signaux ordinaires.

**Figure D-5.3.6 et légende, p. 34**

Nous pouvons voir que l'ordre des transitions a été modifié par rapport à la figure D-5.3.5. L'appel suivant arrive avant la déconnexion de la ligne d'abonné, ce qui signifie que l'appel ne sera pas immédiatement rejeté.

### D.5.3.3 *Transformation*

Le service et le signal prioritaire sont des notions complémentaires car ils sont composés (modélisés) à partir de notions plus fondamentales du LDS, comme le processus et le signal. Pour pouvoir interpréter sémantiquement le service et le signal de priorité, ils doivent se transformer à nouveau en ces notions de base. Normalement, cette transformation ne doit pas être exécutée par l'utilisateur, tout au moins manuellement, mais celui-ci doit naturellement en être conscient. Dans un outil de contrôle sémantique du service, la transformation pourrait s'effectuer automatiquement.

Après la transformation, les services ont disparu et sont remplacés par une définition de processus étendue qui peut être interprétée sémantiquement. La transformation a défini le comportement.

D.5.3.3.1 *Transformation d’´etats*

La Recommandation donne des r`egles sp´ecifiques pour la transformation d’´etats. Le r´esultat de ces r`egles est que le nombre d’´etats du processus est le produit du nombre d’´etats qui se trouvent dans les services.

En outre, l’emploi de signaux prioritaires doublera ce produit car chaque ´etat transform´e est divis´e en un ´etat pr´eliminaire et un ´etat principal.

Ce doublement du nombre des ´etats est d | aux signaux prioritaires qui ne sont pas trait´es dans l’exemple suivant de transformation d’´etats.

Dans bien des cas, de nombreux ´etats du processus <<transform´e>> ne sont pas pertinents et l’espace d’´etat peut | tre r´eduit. Cela est trait´e dans l’exemple suivant, tir´e du § D.10.2.

Dans le processus <<SUBSCRIBER _LINE>>, nous avons 4 services: <<A _subscriber _actions>>, <<B _subscriber _actions>>, <<Connection.Disconnection>> et <<Congestion _supervision>>. Le nombre d’´etats est de 10, 5, 3 et 2, c’est-`a-dire 20 au total.

Appliquant à ces services les r`egles pour la transformation d’´etats, nous avons 300 ´etats (10*5*3*2) dans le processus, ce qui signifie 300 multiplets de noms. La dimension du multiplet est de 4 (4 services). Les positions du multiplet sont arrang´ees selon la figure D-5.3.7.

**Figure D-5.3.7, p. 35**

Tous les noms d’´etat possibles dans les diff´erentes positions donnent 300 combinaisons.

Ex.<A _IDLE, B _IDLE, CONNECTED, NO _ALARM>

<AWAIT _CONN, B _IDLE, CONNECTED, NO _ALARM>

<AWAIT _FIRST _DIGIT, B _IDLE, CONNECTED, NO _ALARM>

. | | |

<AWAIT _A _ON _HOOK _2, B _IDLE, CONNECTED, NO _ALARM>

<A _IDLE, B _RINGING, CONNECTED, NO _ALARM>

<A _IDLE, B _CONVERSATION, CONNECTED, NO _ALARM>

. | | |

<A _IDLE, AWAIT _B _ON _HOOK, CONNECTED, NO _ALARM>

. | | |

. | | |

<AWAIT _A _ON _HOOK _2, AWAIT _B _ON _HOOK, SEIZED, ALARM>

Beaucoup de ces combinaisons ne sont pas pertinentes car une ligne d'abonn´e ne pourra jamais | tre utilis´ee à la fois pour l'abonn´e A et l'abonn´e B au cours du m | me appel. Cela signifie que les 10 ´etats des ‘A _subscriber actions‘ ne peuvent | tre combin´es qu'avec un seul ´etat des ‘B _subscriber actions‘ c'est-`a-dire ‘B _IDLE‘. Cela signifie en outre que les 5 ´etats de ‘B _subscriber actions‘ ne peuvent | tre combin´es qu'avec un seul ´etat de ‘A _subscriber actions‘ (A _IDLE). Le nombre d'´etats pertinents est donc de (10*1*3*2) + (1*5*3*2) = 90.

Une r´eduction de l'espace d'´etat, comme dans l'exemple ci-dessus, d´epend du cas dont il s'agit et ne peut faire l'objet de r`egles g´en´erales formelles, applicables en vue d'une transformation automatique. Cependant, si la transformation est ex´ecut´ee manuellement, il est probable que l'espace d'´etat peut | tre r´eduit. Ainsi, lorsque l'on compare la complexit´e d'un processus con,cu sans services à celle du m | me processus subdivis´e en services, il convient d'utiliser le processus à espace d'´etat r´eduit pour obtenir une bonne base de comparaison. Dans la plupart des cas, le recours aux services permettra de r´eduire consid´erablement le nombre des ´etats.

D.5.3.3.2 *Transformation de transitions*

La Recommandation contient des r`egles sp´ecifiques applicables à la transformation de transitions. L'exemple qui suit illustre les modalit´es de transformation. Le processus de cet exemple est une sp´ecification d'un protocole simple. Le processus est subdivis´e en deux services, ´emission (send) et r´eception (receive).

**Figure D-5.3.8 et l´egende, p. 36**

La figure D-5.3.9 est un diagramme synoptique d’´etat pour les deux services. Les transitions sont num´erot´ees de 1 à 7.

**Figure D-5.3.9, p. 37**

Si l’on applique les r`egles formelles de transformation d’´etats et de transitions, on obtient pour le processus le diagramme synoptique d’´etat ci-apr`es. Aucun signal prioritaire n’est utilis´e, de sorte qu’il n’est pas n´ecessaire de diviser plus avant les ´etats en ´etats pr´eliminaires et ´etats principaux et cela n’est donc pas indiqu´e.

**Figure D-5.3.10, p. 38**

Le nombre d’´etats ne peut | tre r´eduit et le graphe de processus est repr´esent´e dans la figure D-5.3.11. Les noms d’´etats du graphe de processus correspondent aux multiplets de noms de la figure D-5.3.10.

*Exemple* — IS.IR correspond à IDLE _S.IDLE _R.

**Figure D-5.3.11 et l´egende, p.**