



UNION INTERNATIONALE DES TÉLÉCOMMUNICATIONS

**CCITT**

COMITÉ CONSULTATIF  
INTERNATIONAL  
TÉLÉGRAPHIQUE ET TÉLÉPHONIQUE

**Z.100 Annexe D**

(11/1988)

ANNEXE D À LA RECOMMANDATION Z.100  
DIRECTIVES POUR LES USAGERS DU LDS

---

**DIRECTIVES POUR LES USAGERS DU LDS**

Réédition de la Recommandation du CCITT Z.100,  
Annexe D publiée dans le Livre Bleu, Fascicule X.2 (1988)

---

## NOTES

1 La Recommandation Z.100, Annexe D du CCITT a été publiée dans le fascicule X.2 du Livre Bleu. Ce fichier est un extrait du Livre Bleu. La présentation peut en être légèrement différente, mais le contenu est identique à celui du Livre Bleu et les conditions en matière de droits d'auteur restent inchangées (voir plus loin).

2 Dans la présente Recommandation, le terme «Administration» désigne indifféremment une administration de télécommunication ou une exploitation reconnue.

© UIT 1988, 2008

Tous droits réservés. Aucune partie de cette publication ne peut être reproduite, par quelque procédé que ce soit, sans l'accord écrit préalable de l'UIT.

ANNEXE D À LA RECOMMANDATION Z.100

**DIRECTIVES POUR LES USAGERS DU LDS**

**TABLE DES MATIÈRES**

	Page
D.1 Préface .....	4
D.2 Introduction .....	4
D.2.1 Aperçu général du LDS .....	4
D.2.1.1 Le LDS fondé sur un modèle de machine à états finis étendue .....	5
D.2.2 Forme syntaxique du LDS .....	5
D.2.3 Applicabilité du LDS .....	7
D.3 Concepts de base du LDS .....	8
D.3.1 Système .....	8
D.3.2 Blocs .....	10
D.3.3 Canaux .....	10
D.3.4 Signaux .....	12
D.3.5 Acheminement des signaux .....	13
D.3.6 Diagrammes de système et de bloc .....	13
D.3.7 Commentaires et extension de texte .....	17
D.3.7.1 Commentaires .....	17
D.3.7.2 Extension de texte .....	19
D.3.8 Processus .....	20
D.3.8.1 Création de processus .....	23
D.3.8.2 Etats .....	25
D.3.8.3 Entrées .....	34
D.3.8.4 Mises en réserve .....	42
D.3.8.5 Condition de validation et signaux continus .....	46
D.3.8.6 Sorties .....	49
D.3.8.7 Tâche .....	51
D.3.8.8 Décisions .....	52
D.3.8.9 Branchements et connecteurs .....	55
D.3.9 Procédures .....	56
D.3.9.1 Corps de procédure .....	57
D.3.9.2 Appel de procédure .....	60
D.3.10 Traitement des données .....	60
D.3.10.1 Déclarations de variables .....	60
D.3.10.2 Variables révélées/visualisées .....	62
D.3.10.3 Variables exportées/importées .....	63
D.3.10.4 Expressions .....	64
D.3.11 Expression du temps en LDS .....	65

	Page
D.3.12	Emploi de qualificatifs..... 66
D.3.13	Syntaxe de noms ..... 67
D.4	Structuration et affinage des systèmes en LDS..... 69
D.4.1	Considérations générales ..... 69
D.4.2	Critères de subdivision ..... 69
D.4.3	Subdivision des blocs ..... 69
D.4.4	Diagramme d'arbre de blocs ..... 73
D.4.5	Division des canaux..... 74
D.4.6	Représentation du système en cas de subdivision..... 75
	D.4.6.1 Sous-ensemble de subdivision cohérent ..... 77
D.4.7	Affinage..... 78
	D.4.7.1 Sous-ensemble d'affinage cohérent..... 79
	D.4.7.2 Transformation entre signaux et sous-signaux ..... 79
D.5	Concepts supplémentaires..... 80
D.5.1	Macros ..... 80
D.5.2	Systèmes génériques..... 83
D.5.3	Services..... 84
	D.5.3.1 Considérations générales ..... 84
	D.5.3.2 Signaux prioritaires..... 87
	D.5.3.3 Transformation..... 90
D.5.4	Directives applicables à la représentation en fonction des états et aux éléments graphiques 94
	D.5.4.1 Observations d'ordre général sur la représentation en fonction des états ..... 95
	D.5.4.2 Illustration d'état et élément graphique ..... 95
D.5.5	Diagrammes auxiliaires ..... 100
	D.5.5.1 Diagramme synoptique d'état..... 100
	D.5.5.2 Matrice état/signal..... 101
	D.5.5.3 Diagramme de séquençement ..... 102
D.6	Définition de données en LDS..... 103
D.6.1	Directives applicables aux données en LDS..... 103
	D.6.1.1 Introduction générale ..... 103
	D.6.1.2 Sortes ..... 104
	D.6.1.3 Opérateurs, littéraux et termes ..... 104
	D.6.1.4 Equations et axiomes ..... 106
	D.6.1.5 Informations complémentaires concernant les équations et les axiomes ..... 108
D.6.2	Générateurs et héritage ..... 111
	D.6.2.1 Générateurs ..... 111
	D.6.2.2 Héritage..... 113
D.6.3	Observations relatives aux équations..... 115
	D.6.3.1 Considérations générales ..... 115
	D.6.3.2 Application de fonctions aux constructeurs..... 116

	Page
D.6.3.3	Spécification d'ensemble d'essai ..... 117
D.6.4	Caractéristiques..... 118
D.6.4.1	Opérateurs cachés ..... 118
D.6.4.2	Relation d'ordre..... 118
D.6.4.3	Sortes avec champs..... 119
D.6.4.4	Sortes indexées ..... 120
D.6.4.5	Valeur par défaut de variables ..... 120
D.6.4.6	Opérateurs actifs ..... 121
D.7	Directives supplémentaires pour le dessin et l'écriture ..... 121
D.7.1	Directives pour le LDS/GR ..... 121
D.7.1.1	Considérations générales ..... 121
D.7.1.2	Points d'entrée et de sortie ..... 121
D.7.1.3	Symboles..... 122
D.7.1.4	Gabarit ..... 122
D.7.2	Directives applicables au LDS/PR..... 124
D.8	Documentation..... 124
D.8.1	Introduction ..... 124
D.8.2	Types de représentation de systèmes ..... 124
D.8.3	Structure de documents..... 125
D.8.4	Mécanisme de référence ..... 130
D.8.5	Classification des documents..... 130
D.8.6	Combinaison de LDS/GR et de LDS/PR..... 132
D.9	Mises en correspondance ..... 133
D.9.1	Mise en correspondance du LDS et du CHILL ..... 133
D.9.2	Mise en correspondance du LDS/GR et du LDS/PR..... 137
D.10	Exemples d'application ..... 137
D.10.1	Introduction ..... 137
D.10.2	Le concept de service..... 137
D.11	Outil pour le LDS ..... 153
D.11.1	Introduction ..... 153
D.11.2	Catégories d'outils..... 153
D.11.3	Entrée des documents ..... 153
D.11.4	Vérification des documents ..... 154
D.11.5	Reproduction des documents..... 154
D.11.6	Production des documents ..... 155
D.11.7	Modélisation et analyse du système..... 155
D.11.8	Génération de code ..... 156
D.11.9	Formation..... 156

## D.1 *Préface*

Le langage de spécification et de description du CCITT (LDS) a tout d'abord fait l'objet des Recommandations Z.101 à Z.103 (Tome VI.4 du Livre orange, 1976) puis, sous une forme développée, des Recommandations Z.101 à Z.104 (Livre jaune, 1980) qui ont été complétées et regroupées en 1984 dans les Recommandations Z.100 à Z.104 (Livre rouge). Au cours de la période d'études 1985-1988, le langage a été encore développé et harmonisé; les Recommandations existantes ont été fondues en une seule et une définition mathématique a été ajoutée.

Des directives sont indispensables aux utilisateurs pour faciliter l'utilisation du LDS dans ses applications à une large gamme de systèmes de télécommunication. Ces directives ont pour but d'aider les utilisateurs à comprendre la Recommandation concernant le LDS et son application à différents secteurs.

L'emploi du LDS est largement répandu au sein du CCITT et des organisations qui en sont membres; en outre, la gamme des applications de ce langage ne cesse de se développer. Les présentes directives sont établies à l'intention de ceux qui envisagent d'utiliser ou utilisent déjà le LDS; elles complètent la Recommandation sur le LDS en y ajoutant des conseils judicieux et des exemples utiles. Il y aura certes quelques chevauchements entre les directives et la Recommandation; cela semble d'ailleurs souhaitable, si l'on veut que les directives soient autonomes et faciles à consulter. C'est néanmoins la Recommandation qui constitue le document de base.

## D.2 *Introduction*

### D.2.1 *Aperçu général du LDS*

Le LDS peut servir à spécifier le fonctionnement que l'on attend d'un système et à décrire le fonctionnement effectif d'un système. Il a été conçu pour spécifier et décrire le comportement des systèmes de commutation qui interviennent dans les télécommunications, mais peut également être utilisé dans une gamme d'applications plus large. De fait, le LDS convient particulièrement bien à tous les systèmes où il est possible de représenter correctement un comportement à l'aide de machines à états finis étendues (§ D.2.1.1) et où l'on s'intéresse spécialement aux phénomènes d'interaction.

Le LDS peut également servir de point de départ à des méthodes de documentation permettant de représenter intégralement la spécification ou la description d'un système. Dans ce contexte, la signification de la spécification et de la description est liée à leur emploi dans le cycle de vie d'un système. Chacune décrit les propriétés fonctionnelles d'un système d'une façon abstraite. La description comprend généralement certains aspects liés à la conception (par exemple, traitement des erreurs); elle est généralement plus complète en ce qui concerne les détails fonctionnels. Chacune doit concorder avec le modèle concret du système. Elles servent donc toutes deux de spécifications avant la mise en œuvre du système, et de documentation (descriptions) après cette même mise en œuvre.

Le LDS peut servir à représenter à divers niveaux de détail les propriétés fonctionnelles d'un système, d'une fonction ou d'une facilité, qu'il s'agisse de leurs spécifications ou de leurs descriptions. Les propriétés fonctionnelles désignent certaines propriétés structurelles (diagramme d'interaction de blocs) ainsi que le comportement. Par «comportement», on entend la manière dont un système réagit à des signaux reçus (entrées), c'est-à-dire les actions qu'il exécute, par exemple, envoi de signaux (sorties), formulation de questions (aux fins de décision) et exécution de tâches.

Les spécifications peuvent être très générales quand une Administration souhaite étudier les possibilités de mise à jour d'un système en introduisant de nouvelles caractéristiques, de nouveaux services, de nouvelles techniques, etc., tout en laissant au fournisseur la possibilité d'offrir de très nombreuses solutions pratiques. Des spécifications de ce genre ne donneront généralement que peu de détails. A l'autre extrémité, il y a les spécifications par lesquelles une Administration demande le remplacement ou l'extension d'un central existant. Dans ce cas, les détails devront probablement être plus poussés, les spécifications des interfaces devant être très détaillées.

Une spécification et une description peuvent être identiques. Il est toujours préférable de concevoir les nouvelles réalisations à partir de la spécification, afin d'en garantir le respect.

D'une manière générale, ce sont les fournisseurs qui rédigent les descriptions pour donner suite à une spécification (ou pour décrire des systèmes que le fournisseur veut mettre sur le marché). Une description sera généralement plus détaillée que la spécification puisqu'il s'agit de rendre compte du comportement détaillé du système.

Il est à noter également que le LDS permet de décrire un système de manière plus ou moins formelle.

Premièrement, il est possible de décrire un système au moyen de constructions LDS associées au langage naturel. La description ainsi obtenue permet le transfert de l'information à un lecteur qui connaît le contexte, mais pas à une machine. Les contrôles pouvant être effectués automatiquement sont très limités.

Deuxièmement, il est possible d'associer aux constructions LDS des énoncés formels constitués d'éléments de types définis et d'opérateurs sur ces éléments. Les propriétés de ces éléments ne sont pas spécifiées; exemple: «Connecter A-B», où A et B sont du type abonné et «Connecter» est une opération autorisée pour ce type. La spécification ainsi obtenue permet le transfert de l'information aux lecteurs qui connaissent la signification des

opérateurs utilisés. Une machine peut comprendre la description jusqu'à un certain niveau et peut procéder à certains contrôles; elle ne peut pas effectuer des contrôles complets ni «mettre en œuvre» le système car les propriétés des opérateurs sont inconnues.

Troisièmement, il est possible également d'indiquer toutes les propriétés de tous les opérateurs. Dans ce cas, la description est entièrement formelle; une machine peut effectuer tous les contrôles et, en principe, mettre en œuvre les systèmes décrits.

Selon l'objectif visé, les descriptions peuvent être adaptées aux besoins des usagers au moyen de différents niveaux de formalisme. Naturellement, plus la description est formelle, plus un être humain aura de la difficulté à la lire.

Dans le texte qui suit, le terme spécification sera utilisé à la fois pour la représentation nécessaire et pour celle des comportements réels.

#### D.2.1.1 *Le LDS fondé sur un modèle de machine à états finis étendue*

En cas d'emploi du LDS, le système à spécifier est représenté par un certain nombre de machines abstraites interconnectées. Une spécification complète comporte obligatoirement:

- 1) la définition de la structure du système en ce qui concerne les machines et leurs interconnexions,
- 2) le comportement dynamique de chaque machine, ses interactions avec les autres machines et avec l'environnement, et
- 3) les opérations sur les données associées aux interactions.

On décrit le comportement dynamique au moyen de modèles qui définissent les mécanismes de fonctionnement des machines abstraites ainsi que la communication entre les machines. La machine abstraite qu'emploie le LDS est une extension de la machine déterministe à états finis (FSM). La FSM est dotée d'une mémoire d'états finis internes et fonctionne avec un ensemble discret et fini d'entrées et de sorties. Pour chaque combinaison d'une entrée et d'un état, la mémoire définit une sortie ainsi que l'état suivant. On considère habituellement que la durée de transition entre deux états est nulle.

L'une des limites de la FSM est la suivante: toutes les données à mémoriser doivent être représentées sous la forme d'états explicites. Il est possible de représenter la plupart des systèmes de cette façon, mais ce n'est pas toujours pratique. On peut être appelé à mémoriser un grand nombre de valeurs importantes pour le comportement futur mais qui ne contribuent pas beaucoup à la compréhension globale du système. Cette information ne doit pas faire partie de l'espace des états explicites; en effet, ceci compliquerait la présentation. Il est possible pour ce genre d'applications d'étendre la FSM en la dotant d'une mémoire auxiliaire et d'une capacité de fonctionnement auxiliaire sur cette mémoire. Cette mémoire auxiliaire peut emmagasiner, par exemple, des informations concernant des adresses et des numéros d'ordre.

Les Recommandations relatives au LDS définissent deux opérations auxiliaires qu'il est possible d'inclure dans les transitions de la machine à états finis étendue (EFSM), à savoir les décisions et les tâches. Les «décisions» vérifient des paramètres associés aux entrées et aux données contenues dans la mémoire auxiliaire lorsque ces données sont importantes pour le séquençement de la machine principale. Les «tâches» exécutent des fonctions telles que le comptage, des opérations sur la mémoire auxiliaire et la manipulation de paramètres d'entrée et de sortie.

En LDS, des signaux représentent les interactions entre machines, c'est-à-dire que les EFSM reçoivent des signaux comme entrées et produisent des signaux comme sorties. Les signaux se composent d'un seul identificateur de signal et facultativement d'un ensemble de paramètres. Le LDS prévoit la possibilité d'un temps de transition différent de zéro, et définit un mécanisme théorique de mise en file d'attente «premier entré, premier sorti» pour les signaux qui parviennent à une machine en train d'exécuter une transition. Les signaux sont traités à tour de rôle, dans leur ordre d'arrivée.

#### D.2.2 *Formes syntaxiques du LDS*

Le LDS est un langage qui se présente sous deux formes différentes, fondées toutes deux sur le même modèle sémantique. L'une est appelée LDS/GR (LDS graphical representation) et repose sur un ensemble de symboles graphiques normalisés. L'autre s'appelle LDS/PR (LDS textual phrase representation) et repose sur des instructions analogues à un langage de programmation. L'une et l'autre représentent les mêmes concepts du LDS.

Un langage graphique présente l'avantage de montrer clairement la structure d'un système et de permettre à des êtres humains de visualiser facilement le flux de contrôle. La représentation textuelle de phrases convient mieux à l'utilisation par des machines.

En tant qu'outil de conception, le LDS devrait être présenté sous une forme permettant à l'utilisateur d'exprimer ses idées clairement et avec concision. Le LDS/GR, qui permet de le faire, correspond davantage à la représentation traditionnelle des machines à états finis étendues.

Le LDS/GR est la forme originale du LDS. Il a été conçu entre 1973 et 1976 a été publié pour la première fois dans la version de 1976 des Recommandations de la série Z.100.

Le LDS/GR a été établi sur la base de langages graphiques élaborés par différentes organisations pour leurs propres utilisations.

La représentation textuelle de phrases du LDS, c'est-à-dire le LDS/PR, a été conçu pendant la période d'études 1977-1980 mais il a fallu y apporter certaines améliorations avant qu'elle puisse faire l'objet d'une Recommandation. Ces améliorations ont été faites au cours de la période d'études suivante et, dès 1984, le LDS/PR est devenu l'une des syntaxes concrètes recommandées du LDS.

Dans un premier temps, le LDS/PR devait être utilisé comme un moyen aisé d'introduire des documents en LDS dans une machine, ce qui était trop difficile avec le GR (en effet, cela nécessitait l'intervention d'équipements périphériques graphiques). C'est pour cette raison que l'on a insisté sur une mise en correspondance élément par élément entre le PR et le GR. Les progrès réalisés en matière de terminaux graphiques (capacités accrues et réduction des coûts) ont fait que le GR est désormais susceptible d'être introduit en machine. Cela ne diminue en rien l'importance et l'utilisation du PR car certains utilisateurs le trouvent plus à leur convenance, particulièrement ceux qui travaillent avec des langages de programmation.

Du fait de cette évolution, la corrélation entre le GR et le PR est moins étroite; cependant, il est encore possible de mettre en correspondance sans difficulté l'une de ces représentations avec l'autre, bien que chacune d'elles ait ses propres particularités. A première vue, le PR ressemble fortement à un langage de programmation (voir la figure D-2.2.1).

```
—  
—  
STATE aw_off_hook;  
INPUT off_hook;  
TASK 'activate charging';  
TASK 'connect';  
OUTPUT reset_timer;  
NEXTSTATE conversation;  
—  
—
```

FIGURE D-2.2.1

**Exemple de LDS/PR**

En fait, tout dépend de ce qui caractérise un texte du point de vue du langage de programmation.

Si nous admettons qu'un programme est défini comme une «information interprétable par une machine», non seulement les PR mais aussi les GR sont des «programmes».

Il existe cependant certaines différences entre une spécification en LDS et un programme réel. Tout d'abord, il n'est pas indispensable qu'une spécification en LDS puisse être exécutée par une machine (bien que cela ne soit pas interdit); ce qui est essentiel, c'est sa capacité d'acheminer des renseignements précis d'un être humain à un autre être humain.

Si nous considérons une spécification en LDS comme un programme, ce qui peut être tenu pour une «spécification en LDS erronée» (en raison d'un texte informel incomplet) pourrait être parfaitement valable si elle était considérée comme une représentation des caractéristiques fonctionnelles d'un système.

Une autre différence réside dans le «style» d'une spécification en LDS, si on la compare à la représentation usuelle d'un programme.

Le LDS ayant pour but de faciliter la communication entre êtres humains, on s'est efforcé de préserver la possibilité de différentes présentations, afin que la présentation en LDS puisse servir à orienter le lecteur vers certains aspects considérés comme plus importants que d'autres. Cela est naturellement sans importance pour un programme, qui est censé être interprété par une machine. La machine n'insiste pas sur un quelconque aspect particulier du programme, mais doit traiter de la même manière son ensemble; en outre, elle n'essaie pas de «comprendre» le programme.

Etant donné ses similitudes avec un programme, le PR a la préférence de certains programmeurs qui utiliseront probablement aussi le CHILL pour la mise en œuvre des besoins. On serait donc probablement tenté de rechercher une correspondance point par point entre le PR et le CHILL, afin que les besoins exprimés en PR puissent être automatiquement transformés en code CHILL. L'inverse est également intéressant car cela permettrait la dérivation d'une spécification en PR à partir d'un programme en CHILL.



On trouvera au § D.9 des exemples de possibilités de mettre en correspondance le LDS avec le CHILL.

### D.2.3 *Applicabilité du LDS*

La figure D-2.3.1 donne un éventail de possibilités d'emploi du LDS aux fins d'achat ou de fourniture de systèmes de commutation pour les télécommunications.

Dans cette figure, les rectangles représentent des groupes fonctionnels typiques, dont les noms précis, qui peuvent varier d'une organisation à l'autre, ayant des activités représentatives de plusieurs Administrations ou de plusieurs fabricants. Chacune des flèches (lignes de liaison) représente la circulation d'une série de documents entre un groupe fonctionnel et un autre; le LDS peut alors être employé en tant que partie de chacune de ces séries de documents. La figure, donnée seulement à titre d'illustration, n'est ni définitive ni complète.

Les domaines d'application sont ceux effectivement modélisés par des machines à états finis étendues, communiquant entre elles, par exemple: commutation téléphonique télex ou de données, systèmes de signalisation (par exemple, système de signalisation n° 7), interfonctionnement des systèmes de signalisation, protocoles pour données et interfaces d'usagers (LHM).

Si l'on considère plus spécialement les systèmes de commutation SPC, on peut citer comme exemple de fonctions pouvant être spécifiées ou décrites à l'aide du LDS: le traitement des appels (acheminement, signalisation, comptage, etc.), la maintenance, le traitement des dérangements (par exemple, alarmes, relève automatique des dérangements, configuration des systèmes, essais périodiques, etc.), la commande des systèmes (par exemple, protection contre les surcharges), et les interfaces homme-machine. Le § D.10 donne des exemples d'application du LDS.

La spécification des protocoles où intervient le LDS fait l'objet des Recommandations de la série X du CCITT.

## D.3 *Concepts de base du LDS*

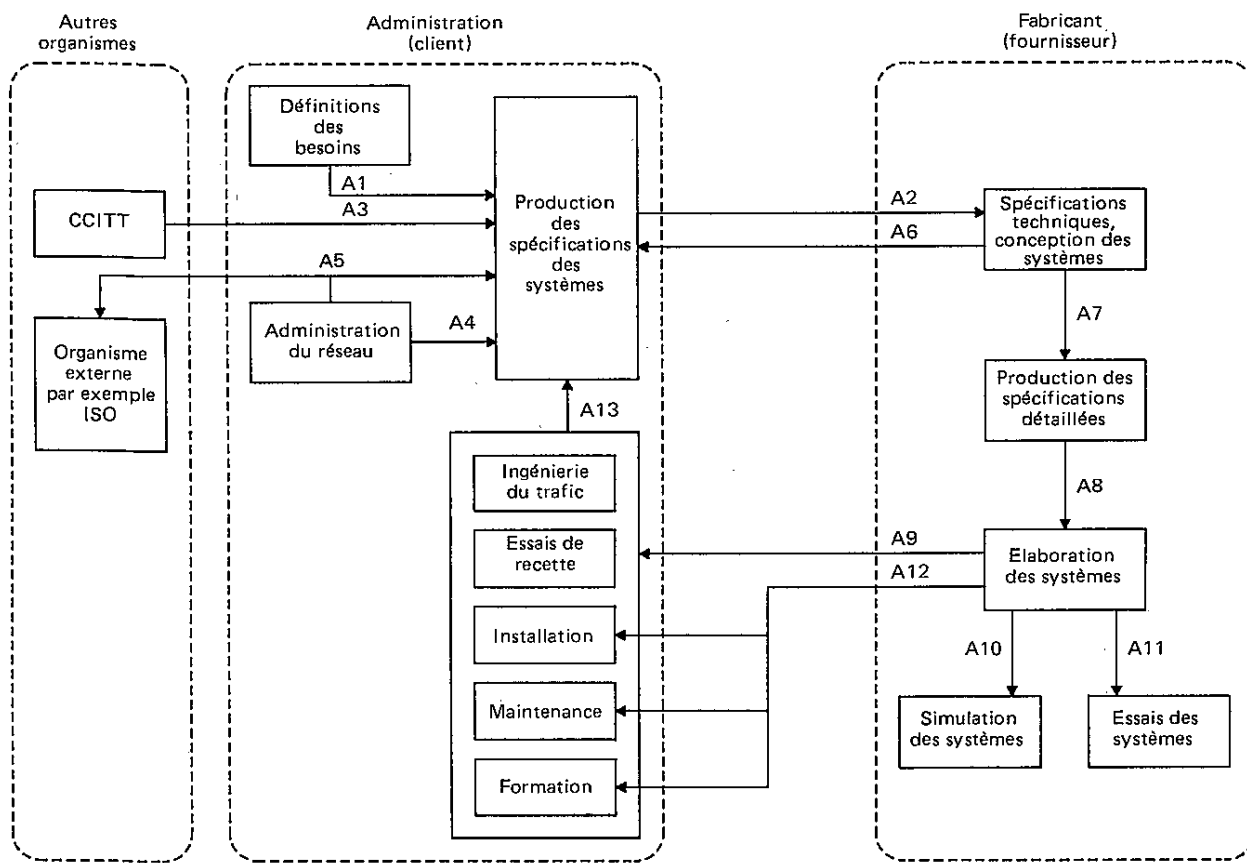
### D.3.1 *Système*

Comme on a pu le voir plus haut, le LDS représente des systèmes à l'aide de modèles. Ainsi ce qui est défini par une spécification ou une description en LDS constitue le système. Un système LDS peut donc représenter à l'aide de modèles une partie d'un système (ou d'un central) téléphonique, ou un réseau complet de systèmes téléphoniques ou des parties d'un grand nombre de centraux téléphoniques (par exemple, les dispositifs de contrôle de circuit aux deux extrémités d'un circuit). Il importe de souligner que le système LDS contient, d'un point de vue LDS, tout ce que la spécification ou la description tente de définir. L'environnement ne relève pas de la spécification et n'est pas défini en LDS.

Des canaux relient le système à l'environnement. Théoriquement, il suffit d'un seul canal bidirectionnel pour connecter le système à l'environnement. Dans la pratique, on définit généralement des canaux pour chaque jonction logique avec l'environnement.

Chaque système se compose d'un certain nombre de blocs reliés par des canaux. Chaque bloc du système est indépendant de tous les autres. Chaque bloc peut contenir un ou plusieurs processus décrivant le comportement du bloc. Seule l'émission de signaux dans les canaux permet la communication entre des processus placés dans deux blocs différents. Pour subdiviser le système en blocs, on peut prendre des critères tels que les suivants: définir des parties d'une dimension facilitant la compréhension, créer une correspondance avec la division effective entre le logiciel et le matériel, suivre les subdivisions fonctionnelles naturelles, réduire au minimum les interactions, etc.

Pour de grands systèmes LDS, il existe certaines constructions en LDS qui permettent de spécifier les sous-structures des parties d'un système, de sorte qu'en partant d'un aperçu général du système, on peut donner toujours plus de détails. Dans ce cas, on peut dire que le système est représenté à différents niveaux de détail. Ces constructions sont décrites au § D.4.



T1004860-89

- A1 Spécification d'une possibilité ou d'une caractéristique, indépendamment de la réalisation et du réseau
- A2 Spécification indépendante de la réalisation et dépendante du réseau, comprenant une description de l'environnement du système
- A3 Recommandations et directives du CCITT
- A4 Contributions à la spécification du système, indiquant les besoins du réseau en matière de gestion et exploitation
- A5 Autres Recommandations pertinentes
- A6 Description d'une proposition de réalisation
- A7 Spécification du projet
- A8 Spécification détaillée de la conception
- A9 Description complète du système
- A10 Documentation relative à la description du système et de son environnement en vue de la simulation du système
- A11 Documentation relative à la description du système et de son environnement en vue de l'essai du système
- A12 Manuels d'installation et d'exploitation
- A13 Contributions à la spécification du système émanant de groupes fonctionnels spécialisés à l'intérieur de l'Administration

*Remarque 1* – L'itération est possible à tous les niveaux.

*Remarque 2* – Dans certains cas, la documentation LDS indiquée ici comme étant interne à une seule organisation, par exemple A1, A7, A8, pourrait être fournie à une autre organisation.

FIGURE D-2.3.1

**Scénario général pour l'utilisation du LDS**

Au premier niveau de précision, la spécification en LDS d'un système décrit la structure du système et comprend les points suivants, expliqués dans les paragraphes qui suivent:

- nom du système;
- définitions de signaux: spécification des types de signaux échangés entre les blocs du système ou entre les blocs et l'environnement. Cela comprend la spécification des types de valeurs acheminés par les signaux (liste de sorties);
- définitions de listes de signaux: spécification des identificateurs groupant plusieurs signaux et/ou autres listes de signaux. De tels identificateurs peuvent servir à économiser de l'espace et à donner une spécification plus claire;
- définitions de canaux: spécification des canaux reliant les blocs du système les uns aux autres et à l'environnement. Une définition de canal comprend la spécification des identificateurs des signaux acheminés par ce canal;

- définitions de données: spécification de nouveaux types de syntypes et de générateurs définis par l'utilisateur et visibles dans tous les blocs;
- définitions de blocs: spécification des blocs dans lesquels le système est subdivisé;
- définitions de macros: des directives concernant l'utilisation des macros sont données au § D.5.1.

Selon la Recommandation concernant le LDS, il existe des types de données prédéfinis qui peuvent être utilisés par chaque système. Ils ne nécessitent pas de définition et peuvent être utilisés au moyen de leurs noms prédéfinis, c'est-à-dire: INTEGER, REAL, CHARACTER, STRING, CHARSTRING, BOOLEAN, PID, TIME, DURATION. Les types de données prédéfinis sont visibles à tous les niveaux de la définition du système; ils peuvent être considérés comme définis implicitement dans une «bibliothèque» de système accessible en tout point de la spécification.

Les noms de sorte utilisés dans les définitions de signaux au niveau de système doivent être introduits par des définitions de type partielles visibles au niveau du système, c'est-à-dire des types de données prédéfinis ou des newtypes définis par l'utilisateur ou encore des syntypes définis à ce niveau.

On trouvera des explications complémentaires sur l'utilisation des types de données aux § D.3.10 et D.6.

Le LDS/PR utilisé pour une définition de système consiste en un ensemble d'instructions se terminant par «;» (point-virgule). La définition d'une structure de système commence par l'instruction «SYSTEM nom;» et se termine par l'instruction «ENDSYSTEM nom;». Le nom de l'instruction terminale est facultatif mais s'il est donné, il doit être le même que le nom donné après le mot clé SYSTEM. Il est suggéré de placer toujours le nom dans l'instruction terminale, car cela améliore la lisibilité du document.

Le schéma du LDS/PR d'une définition de structure de système est représenté dans la figure D-3.1.1.

```
SYSTEM . . . ;
    . . . signal definitions . . .
    . . . signal list definitions . . .
    . . . channel definitions . . .
    . . . data definitions . . .
    . . . block definitions . . .
    . . . macro definitions . . .
ENDSYSTEM . . . ;
```

FIGURE D-3.1.1

**Schéma du LDS/PR utilisé pour la définition de structure de système**

Pour permettre d'obtenir une représentation plus claire et plus simple de la structure du système ou permettre une spécification descendante de système, le LDS contient un mécanisme général de référence. A ce niveau, le mécanisme de référence peut être appliqué aux définitions de bloc. Cette caractéristique du langage permet à l'utilisateur de spécifier précisément le nom de bloc à l'intérieur de la définition de la structure de système; la définition de bloc proprement dite peut être donnée séparément (voir la figure D-3.1.2).

```
SYSTEM s;
    . . .
    BLOCK b1 REFERENCED;
    BLOCK b2 REFERENCED;
    . . .
ENDSYSTEM s;
```

FIGURE D-3.1.2

**Exemple de référence de définition de bloc**

Le mécanisme de référence est particulièrement utilisé dans le LDS/GR parce que la plupart des diagrammes doivent tenir sur une seule page et que la place manque souvent pour des spécifications graphiques emboîtées.

On trouvera des exemples de LDS/GR pour une définition de système au § D.3.6.

### D.3.2 *Blocs*

Les processus peuvent communiquer entre eux à l'intérieur d'un bloc, au moyen de signaux, de valeurs partagées. Ainsi le bloc ne constitue pas seulement un mécanisme pratique pour le regroupement de processus, mais encore une limite à la visibilité des données. C'est pourquoi il convient, lors de la définition de blocs, de s'assurer du caractère raisonnable et fonctionnel du regroupement de processus au sein d'un bloc. Dans la plupart des cas, il est utile de fractionner dans un premier temps le système (ou le bloc) en unités fonctionnelles, puis de définir les processus à intégrer dans le bloc.

À l'intérieur d'un bloc, il est possible (à titre d'option) de définir les trajets de communication entre les processus ou entre ceux-ci et l'environnement du bloc (c'est-à-dire la frontière du bloc). Ces trajets de communications sont appelés acheminement de signaux.

Pour un système LDS de grandes dimensions, il est possible de décrire la sous-structure d'un bloc en fonction d'autres blocs et de canaux, comme si le bloc était un système en soi. Ce mécanisme est expliqué au § D.4.

La définition de la structure d'un bloc peut comporter les points suivants:

- nom de bloc;
- définitions de signaux: spécification des types de signaux échangés à l'intérieur du bloc. Cela comprend la spécification des types de valeurs acheminés par les signaux (liste de sorties);
- définitions de listes de signaux: spécifications ou identificateurs correspondant à des listes de signaux et/ou à d'autres identificateurs de listes de signaux. Ces identificateurs, groupant plusieurs signaux, peuvent être utilisés pour économiser de l'espace et obtenir une spécification plus claire;
- définitions d'acheminement de signaux: spécifications des trajets de communication reliant les processus du bloc les uns aux autres et à l'environnement du bloc. Une définition d'acheminement de signaux comprend la spécification des identificateurs de signaux transportés sur cet acheminement de signaux;
- connexions de canaux vers acheminements: spécifications des connexions entre les canaux extérieurs au bloc et les acheminements de signaux internes du bloc;
- définitions de processus: spécification des types de processus décrivant le comportement du bloc. Si le bloc n'est pas décrit en fonction de sa sous-structure, il faut qu'il y ait au moins une définition de type de processus à l'intérieur du bloc. Pour la définition du processus, un mécanisme de référence est fourni comme cela est indiqué, dans le cas des blocs, au § D.3.1;
- définitions des données: spécification de newtypes, de syntypes et de générateurs définis par l'utilisateur et visibles dans tous les processus définis du bloc et/ou dans la sous-structure du bloc;
- définitions de macros: des directives pour l'utilisation de macros sont données au § D.5.1.

S'il existe une sous-structure à l'intérieur du bloc, certains des points ci-dessus sont facultatifs (voir le § D.4 pour les explications concernant la structure).

Les types suivants sont visibles dans un bloc:

- des types de données prédéfinis;
- des types de données définis par l'utilisateur, définis dans le bloc proprement dit;
- des types de données définis par l'utilisateur visibles dans le bloc ascendant (en cas de subdivision des blocs).

Dans le LDS/PR, les mots clé BLOCK et ENDBLOCK servent à circonscrire une définition de bloc. On trouvera des exemples du LDS/GR pour une définition de bloc au § D.3.6.

### D.3.3 *Canaux*

Les canaux sont les moyens de communication entre différents blocs du système ou entre des blocs et leur environnement. Un canal peut relier un bloc à un autre ou un bloc à l'environnement dans une direction (canal unidirectionnel) ou dans les deux directions (canal bidirectionnel). Normalement, un canal est une entité fonctionnelle qui peut être utilisée pour désigner des chemins de communication spécifiques. En fait, par la subdivision des canaux (décrite au § D.4.5), il est possible de spécifier formellement le comportement de chaque canal.

Pour chaque direction indiquée ou chaque chemin de communication, la spécification du canal contient une liste de tous les identificateurs de signaux que le canal peut acheminer dans cette direction. Cette liste de signaux offre le moyen de garantir que chaque signal envoyé par un processus à une extrémité du canal puisse être reçu par le processus du bloc qui se trouve à l'autre extrémité du canal. Ainsi, la spécification de canal devient partie intégrante de la spécification d'interface pour chaque bloc. Dans de grands projets intéressant de nombreuses personnes, un accord dès

l'origine sur les signaux d'un canal et sur la spécification de ces signaux réduit sensiblement la probabilité que deux processus ne pourront communiquer l'un avec l'autre comme prévu.

La définition d'un canal comporte les éléments suivants:

- nom du canal;
- un ou deux chemins de communication: un chemin de communication spécifie l'origine et la destination d'une liste de signaux. Des identificateurs de bloc ou le mot clé «ENV» (environnement) peuvent être utilisés dans ce contexte;
- une ou deux listes de signaux: une liste des signaux transportés dans cette direction doit être spécifiée pour chaque chemin de communication. Cette liste peut comporter des identificateurs de signaux ainsi que des identificateurs d'autres listes;
- une définition facultative de sous-structure de canal (ou une référence à une telle définition): voir le § D.4.5.

Dans le LDS/PR, une définition de canal est comprise entre les deux mots clés CHANNEL et ENDCHANNEL. Les mots clés FROM et TO servent à désigner les chemins de communication et le mot clé WITH les listes de signaux. On trouvera dans la figure D-3.3.1 un exemple de définition de canal en LDS/PR.

Dans le LDS/GR, une définition de canal est représentée à l'aide d'une ligne reliant les deux parties mises en jeu dans la communication. Le nom du canal doit être plus proche de la ligne que tout autre symbole. Les trajets sont représentés au moyen de flèches et les listes de signaux doivent être placés entre crochets, comme indiqué dans les exemples de la figure D-3.3.2. Pour éviter une confusion entre les canaux et les acheminements de signaux, les flèches ne doivent pas être placées en l'un des deux points terminaux de la ligne (voir le § D.3.5).

Dans un canal bidirectionnel, chacune des deux listes de signaux doit être aussi proche que possible de la flèche correspondante.

#### D.3.4 *Signaux*

Les signaux peuvent être définis au niveau du système, au niveau du bloc ou dans la partie interne de la définition de processus. Les signaux définis à un certain niveau peuvent être utilisés à ce niveau ou également aux niveaux inférieurs; cependant, pour simplifier chaque niveau, il est suggéré de définir les signaux de manière aussi circonscrite que possible. Les signaux définis dans le cadre d'une définition de processus peuvent être échangés entre des instances de même type de processus (§ D.3.8) ou entre services d'un processus (§ D.5.3).

La définition d'un signal comprend les points suivants:

- nom du signal;
- liste de sortes (facultative): représente la liste des types de valeurs acheminées par ce signal;
- affinage du signal (facultatif): voir le § D.4.7.

En LDS/PR, une définition de signal est spécifiée par le mot clé SIGNAL. Plusieurs signaux (n'ayant pas fait l'objet d'un affinage) peuvent être définis à l'intérieur de la construction, donnant ainsi le nom du signal et la liste de types. Des exemples de définitions de signaux en LDS/PR sont donnés dans la figure D-3.4.1.

```

SYSTEM so_and_so;
...
SIGNALLIST s_list_id_1 = sig_b, sig_c, sig_d;
...
CHANNEL c1
  FROM block_a TO block_b WITH signal_1,signal_2,signal_3;
ENDCHANNEL c1;

CHANNEL chan_2
  FROM ENV TO block_c WITH ext_sig_1,ext_sig_2;
  FROM block_c TO ENV WITH int_sig_1;
ENDCHANNEL chan_2;

CHANNEL chan.name.3
  FROM bx TO by WITH sig_a,(s_list_id_1),sig_e;
ENDCHANNEL chan.name.3;
...
ENDSYSTEM so_and_so;

```

FIGURE D-3.3.1

Exemple de définitions de canaux en LDS/PR

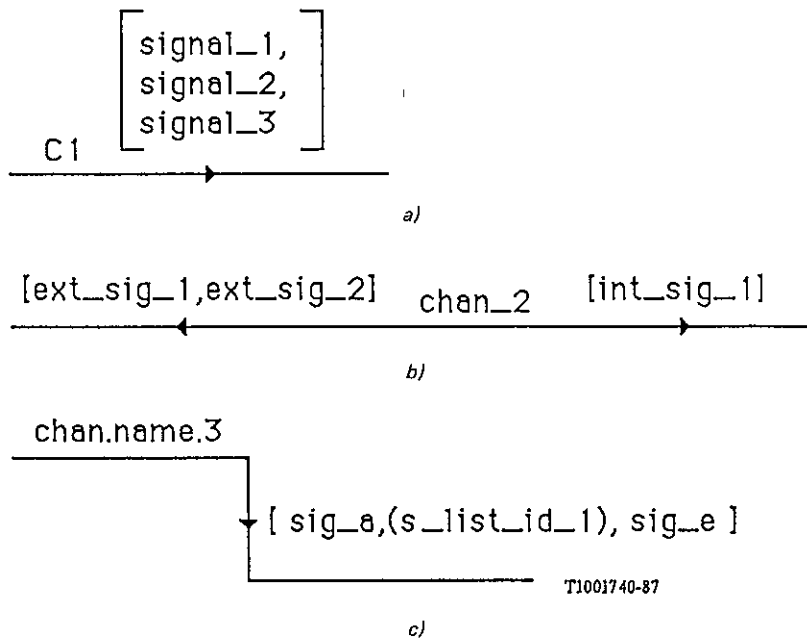


FIGURE D-3.3.2

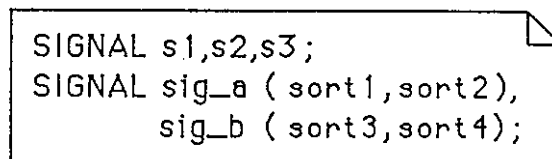
Exemple de définitions de canaux en LDS/GR

```
SIGNAL s1,s2,s3;  
  
SIGNAL sig_a(sort1,sort2),  
sig_b(sort3,sort4);
```

FIGURE D-3.4.1

**Exemples de définitions de signaux en LDS/PR**

En LDS/GR, on spécifie une définition de signal en insérant des énoncés linéaires dans un symbole de texte comme indiqué dans la figure D-3.4.2.



```
SIGNAL s1,s2,s3;  
SIGNAL sig_a ( sort1,sort2),  
sig_b ( sort3,sort4);
```

T1001750-87

FIGURE D-3.4.2

**Exemples de définitions de signaux en LDS/GR**

### D.3.5 *Acheminements de signaux*

Des acheminements de signaux servent à représenter des chemins de communication similaires aux canaux. Ils peuvent être utilisés au niveau des blocs et à celui des processus. Comme les canaux, les acheminements de signaux peuvent être unidirectionnels ou bidirectionnels mais ils ne peuvent être subdivisés.

Au niveau des blocs, ils représentent un moyen de communication entre les processus d'un bloc ou entre les processus et l'environnement de ce bloc, c'est-à-dire un canal menant à ce bloc ou venant de celui-ci.

Au niveau du processus, les acheminements de signaux peuvent être utilisés lorsque le processus est subdivisé en services (voir le § D.5.3). Dans ce cas, ils relient les services les uns aux autres ou avec les acheminements de signaux du processus.

Si un signal est remis à un acheminement de signaux aboutissant à une frontière de bloc, il passe dans le canal relié à l'acheminement de signaux. Lorsqu'un signal parvient au bloc, à partir d'un canal relié à un ou plusieurs acheminements de signaux, ce signal est placé dans l'acheminement de signaux qui est capable de la transférer.

En LDS/PR, la définition d'un acheminement de signal commence par le mot clé SIGNALROUTE. La syntaxe des chemins de communication et la liste des signaux sont les mêmes que dans le cas des canaux.

En LDS/GR, la seule différence entre un acheminement de signaux et un canal est que, pour les acheminements de signaux, des flèches doivent être placées aux points terminaux des lignes; près de chaque flèche, doit se trouver une liste de signaux appropriés. On trouvera des exemples d'acheminements de signaux dans les figures D-3.6.3 et D-3.6.5 des paragraphes qui suivent.

### D.3.6 *Diagrammes de système et de bloc*

En LDS/GR, une définition de système est représentée au moyen d'un ensemble de diagrammes. La structure d'un système en canaux et en blocs, est représentée à l'aide d'un diagramme de système.

Le diagramme de système se compose des éléments suivants:

- le symbole de cadre: symbole de forme rectangulaire contenant tous les autres symboles. Il représente la frontière du système; l'environnement du système se trouve en dehors de ce cadre;
- l'en-tête du système: mot clé SYSTEM suivi du nom du système (placé dans l'angle supérieur gauche);
- une numérotation de page facultative (placée dans l'angle supérieur droit du cadre);

- des symboles de texte: un tel symbole peut contenir des énoncés linéaires. Il sert généralement à présenter dans le diagramme des définitions de signaux, des listes de signaux et les données;
- la zone d'interaction de blocs qui comprend la spécification des blocs du système, les canaux et les listes de signaux acheminés par les canaux;
- des diagrammes de macros: les directives concernant l'utilisation des macros sont données au § D.5.1.

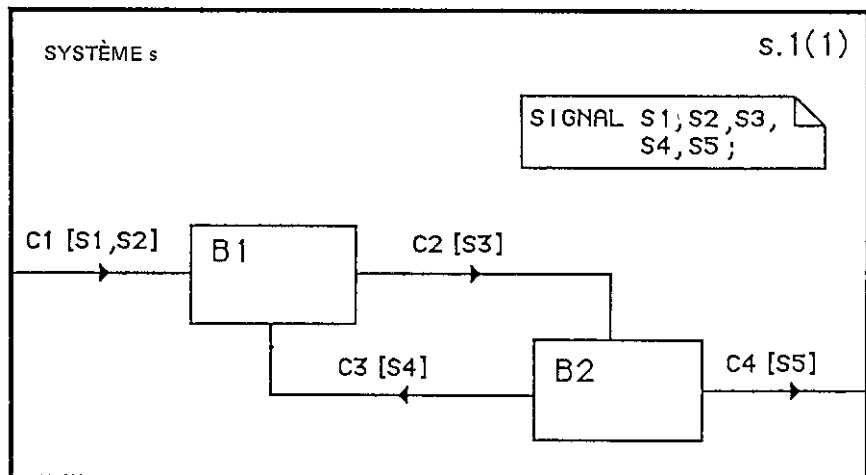
Dans le diagramme de système, la spécification d'un bloc peut être l'un des deux points suivants:

- une référence de bloc: symbole de bloc contenant uniquement le nom de bloc;
- un diagramme de bloc: cadre contenant la spécification de la structure de bloc en fonction de ses processus et interactions. Si un bloc est subdivisé en sous-blocs, la spécification de la sous-structure ou la référence à celle-ci doit être placée à l'intérieur du cadre de bloc (§ D.4.3).

On trouvera dans le résumé concernant le LDS/GR la forme des symboles utilisés dans le système et celle des diagrammes de bloc. En ce qui concerne les définitions de symboles, il convient de se référer au § D.7.1.4.

La figure D-3.6.1 donne un exemple d'un diagramme de système pour le système «s». Dans cet exemple, le système s'est divisé en deux blocs B1 et B2 reliés l'un à l'autre et à l'environnement par les canaux C1, C2, C3 et C4. Pour les blocs B1 et B2, on ne donne dans cet exemple qu'une référence. Des explications complémentaires sur les canaux et les symboles de listes de signaux sont données dans les paragraphes qui suivent:

Le même exemple en LDS/PR est représenté dans la figure D-3.6.2.



T1001760-87

FIGURE D-3.6.1

Exemple de diagramme de système

```

SYSTEM s;
  SIGNAL S1,S2,S3,S4,S5;
  CHANNEL C1 FROM ENV TO B1 WITH S1,S2;
  ENDCHANNEL C1;
  CHANNEL C2 FROM B1 TO B2 WITH S3;
  ENDCHANNEL C2;
  CHANNEL C3 FROM B2 TO B1 WITH S4;
  ENDCHANNEL C3;
  CHANNEL C4 FROM B2 TO ENV WITH S5;
  ENDCHANNEL C4;
  BLOCK B1 REFERENCED;
  BLOCK B2 REFERENCED;
ENDSYSTEM s;

```

FIGURE D-3.6.2

Exemple de définition de structure de système en LDS/PR



La structure d'un bloc en ce qui concerne les processus et les acheminements de signaux est représentée en LDS/GR à l'aide d'un diagramme de bloc.

Le diagramme de bloc se compose des éléments suivants:

- le symbole cadre: symbole de forme rectangulaire contenant tous les autres symboles. Il représente les frontières du bloc: au-delà du cadre commence l'environnement du bloc;
- l'en-tête du bloc: le mot clé BLOCK suivi du nom de bloc (placé dans l'angle supérieur gauche du cadre);
- une numérotation de page facultative (placée dans l'angle supérieur droit du cadre);
- des symboles de texte: ces symboles peuvent englober des énoncés linéaires. Ils sont généralement utilisés pour présenter les définitions de signaux, de listes de signaux et de données;
- la zone d'interaction de processus: cette zone comprend la spécification des processus du bloc et éventuellement des acheminements de signaux et des listes de signaux transportés par les acheminements de signaux. Dans cette zone, il est également possible de représenter les processus qui créent d'autres processus; cette caractéristique est décrite au § D.3.8.1;
- des identificateurs de canaux: si le diagramme fait apparaître des acheminements de signaux aboutissant à l'environnement du bloc ou venant de celui-ci, les identificateurs des canaux reliés à ces acheminements de signaux doivent être spécifiés en dehors du cadre, de façon correspondant aux lignes des acheminements de signaux;
- des diagrammes de macros: on trouvera au § D.5.1 des directives sur l'utilisation des macros.

Dans le diagramme de bloc, la spécification d'un processus peut être:

- soit une référence à un processus: symbole de processus contenant le nom de processus et, en option, la spécification d'instances de processus. Une telle spécification comprend deux couples de nombres entiers séparés par une virgule et placés entre parenthèses (voir le § D.3.8);
- soit un diagramme de processus: cadre contenant un graphique de symboles reliés décrivant le comportement du processus en ce qui concerne les états, les entrées, les sorties, les actions, etc. (voir le § D.3.8). Si le processus est subdivisé en services, le diagramme de processus contient la zone d'interaction de service (§ D.5.3).

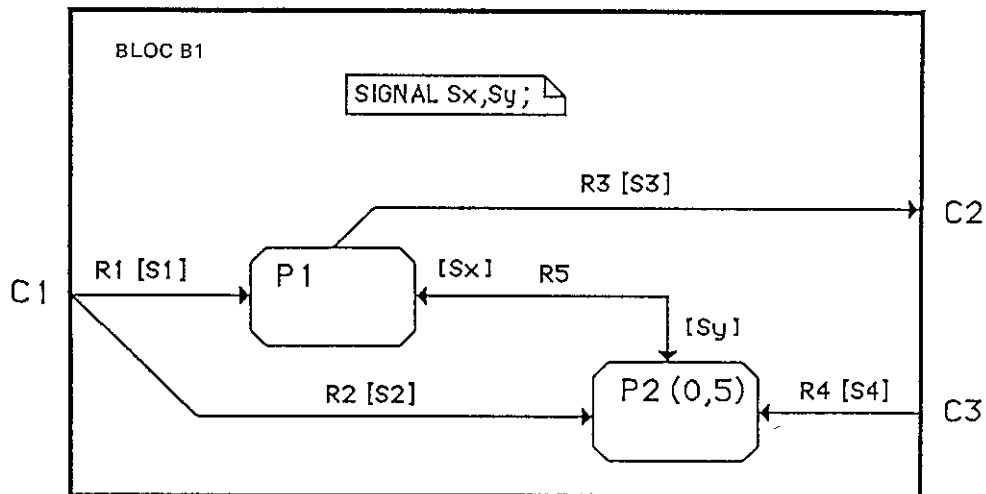
S'il existe une sous-structure du bloc, certains des points ci-dessus sont facultatifs (voir le § D.4 pour l'explication de la structure).

Dans la figure D-3.6.3, on trouvera un exemple de diagramme de bloc pour le bloc «B1» présenté dans l'exemple de la figure D-3.6.1. Ce bloc B1 est décrit du point de vue des deux processus P1 et P2, reliés par les acheminements de signaux R1, R2, R3, R4 et R5. Pour les processus P1 et P2, on se borne à donner des références dans cet exemple. Les identificateurs de canaux C1, C2 et C3 sont également spécifiés en dehors du cadre.

Le même exemple est présenté en LDS/PR dans la figure D-3.6.4.

Comme cela a déjà été indiqué précédemment, des diagrammes de bloc peuvent être compris dans les diagrammes de système, en lieu et place de références. Dans la figure D-3.6.5 par exemple, les diagrammes des figures D-3.6.1 et D-3.6.3 sont réunis en un seul diagramme de système.

Une recommandation générale concernant l'établissement de diagrammes de ce genre, est qu'ils ne doivent pas être trop complexes afin de rester lisibles et qu'ils doivent tenir sur une seule page.



T1001770-87

FIGURE D-3.6.3  
Exemple de diagramme de bloc

BLOCK B1;

SIGNAL Sx,Sy;

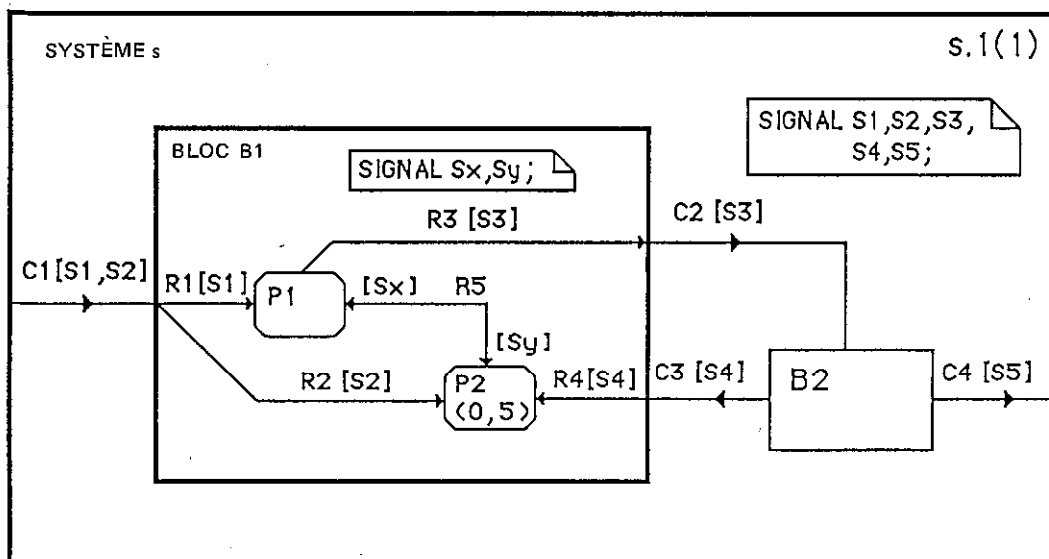
SIGNALROUTE R1 FROM ENV TO P1 WITH S1;  
 SIGNALROUTE R2 FROM ENV TO P2 WITH S2;  
 SIGNALROUTE R3 FROM P1 TO ENV WITH S3;  
 SIGNALROUTE R4 FROM ENV TO P2 WITH S4;  
 SIGNALROUTE R5 FROM P1 TO P2 WITH Sy;  
 FROM P2 TO P1 WITH Sx;

CONNECT C1 AND R1,R2;  
 CONNECT C2 AND R3;  
 CONNECT C3 AND R4;

PROCESS P1 REFERENCED;  
 PROCESS P2 REFERENCED;

ENDBLOCK B1;

FIGURE D-3.6.4  
Exemple d'une definition de structure de bloc en LDS/PR



T1001780-87

FIGURE D-3.6.5

Exemple d'un diagramme de système comportant un diagramme de bloc

### D.3.7 Commentaires et extension de texte

#### D.3.7.1 Commentaires

Il est possible d'ajouter des commentaires à une spécification en LDS pour aider le lecteur et préciser certains points. Deux types de commentaires adaptés au LDS/PR et au LDS/GR ont été introduits dans le LDS.

Le premier type, appelé «note» est utilisé particulièrement en LDS/PR. Il est délimité par «\*/» au début et par «\*/» à la fin.

En LDS/PR, un tel commentaire peut s'insérer partout où se trouve un espace. Il ne doit pas contenir la séquence spéciale «\*/».

En LDS/GR, un tel commentaire peut se présenter partout où il existe un espace à l'intérieur des instructions linéaires.

On trouvera dans les figures D-3.7.1 et D-3.7.2 certains exemples de cette forme de commentaire, respectivement en LDS/PR et en LDS/GR.

```

SYSTEM NSS;

/* Définition du système national de commutation.

   Nom du système: NSS */

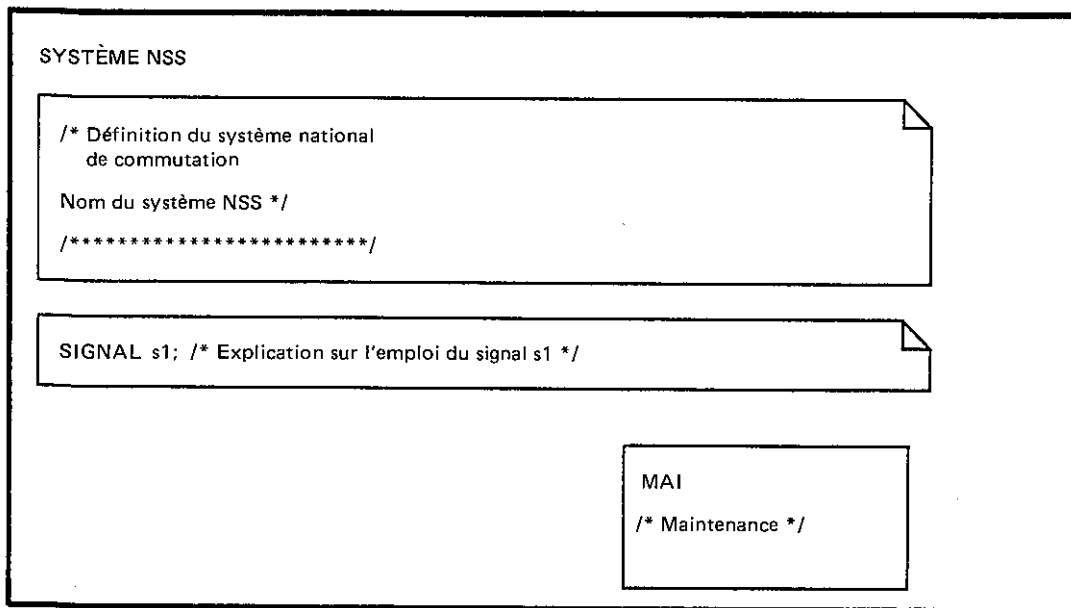
/.....*/

...
SIGNAL s1; /* Explications sur l'emploi du signal s1 */
...
BLOCK MAN REFERENCED; /* Maintenance */
...
ENDSYSTEM NSS;

```

FIGURE D-3.7.1

Exemples de la première forme de commentaire en LDS/PR



T1001790-87

FIGURE D-3.7.2

**Exemples de la première forme de commentaire en LDS/GR**

La seconde forme de commentaire permet une mise en correspondance élément par élément entre le LDS/PR et le LDS/GR; elle convient mieux aux applications comportant des traductions automatiques.

En LDS/PR, un tel commentaire se compose du mot clé COMMENT suivi d'une chaîne de caractères; il peut être inséré comme une instruction (c'est-à-dire suivi de «;») partout où une instruction de tâche peut être insérée. De plus, il peut être inséré avant le symbole «;» (;) à la fin de toute instruction.

En LDS/GR, cette forme de commentaire est représentée à l'aide d'un symbole de commentaire contenant le texte du commentaire. Le symbole de commentaire est un symbole de forme rectangulaire auquel le côté gauche ou droit fait défaut. Ce symbole doit être étendu aussi bien horizontalement que verticalement, de manière à contenir tout le texte. Il peut être relié à un symbole quelconque en LDS/GR ou à une ligne de liaison. Pour indiquer la connexion il faut employer une ligne en traits discontinus. Si l'association entre le texte du commentaire et le symbole du commentaire ne présente pas d'ambiguïté, le symbole de commentaire peut se présenter simplement sous la forme de crochets.

Dans les figures D-3.7.3 et D-3.7.4, on trouvera certains exemples de cette forme de commentaire, respectivement en LDS/PR et en LDS/GR.

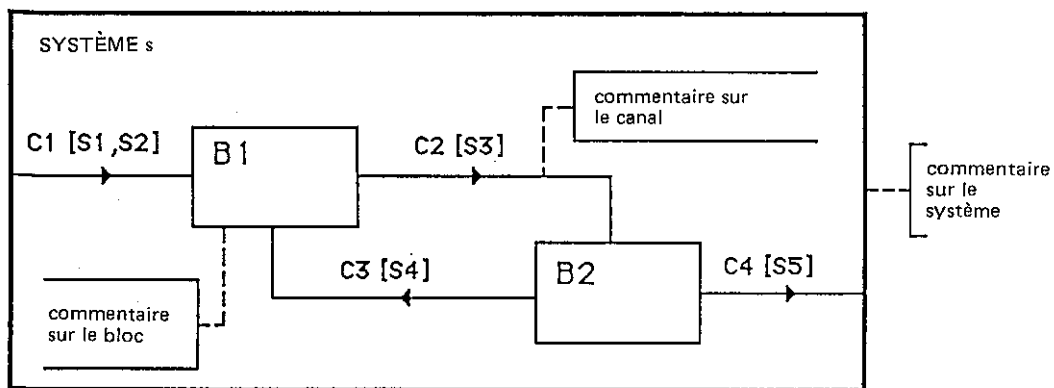
```

SYSTEM s COMMENT 'commentaire sur le système';
CHANNEL C2
  FROM B1 TO B2
  WITH s3      COMMENT 'commentaire sur le canal';
ENDCHANNEL C2;
BLOCK B1
  COMMENT 'commentaire sur le bloc';
  ...
  PROCESS p1;
  ...
  TASK 't1';
  TASK 't2';
  ...
ENDPROCESS p1;
ENDBLOCK B1;
ENDSYSTEM s;

```

FIGURE D-3.7.3

Exemples de la seconde forme de commentaire en LDS/PR



T1001801-69

FIGURE D-3.7.4

Exemples de la seconde forme de commentaire en LDS/GR

### D.3.7.2 Extension de texte

Normalement, le texte associé à un symbole graphique devrait être placé à l'intérieur de ce symbole. Cependant, cela n'est pas toujours possible ou pratique. Une autre solution consiste à placer le texte dans un symbole d'extension de texte rattaché au symbole associé. Le symbole d'extension de texte est similaire au symbole de commentaire; la seule différence est que la ligne le reliant est en trait continu et non en trait discontinu (voir la figure D-3.7.5).

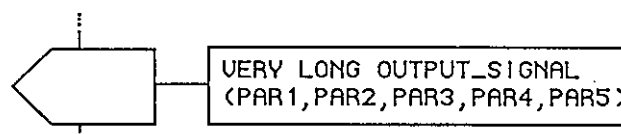
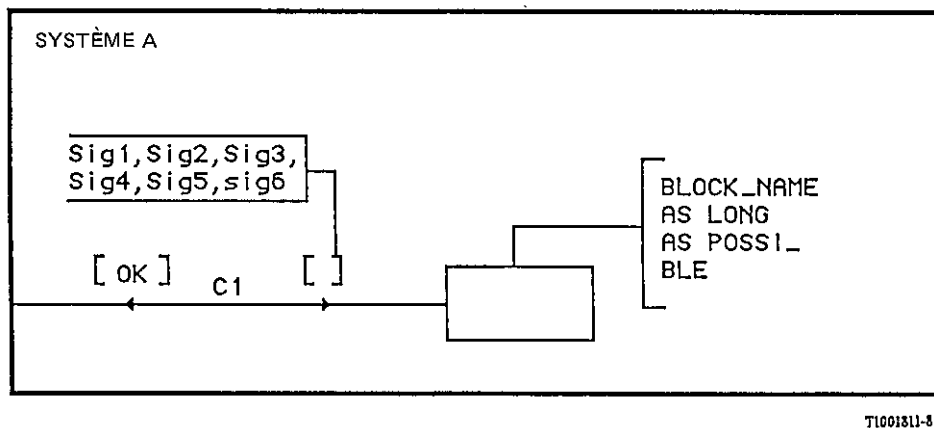


FIGURE D-3.7.5

**Exemples de l'utilisation d'un symbole d'extension de texte**

Un caractère de soulignement («\_») peut être utilisé à la fin d'une ligne de texte comme caractère de continuation. Dans ce cas, les espaces restant sur la même ligne ne sont pas considérés comme faisant partie du texte. On trouvera au § D.3.13 des directives plus détaillées sur la syntaxe des noms.

D.3.8 *Processus*

Un processus est une machine à états finis étendue qui définit le comportement dynamique d'un système. Les processus sont foncièrement dans un état d'attente des signaux. A la réception d'un signal, le processus répond en accomplissant les actions précises qui correspondent à chaque type de signal pouvant être reçu par le processus. Les processus contiennent de nombreux états qui leur permettent d'accomplir différentes actions en cas de réception d'un signal. Ces états mémorisent les actions précédemment accomplies. Après l'accomplissement de toutes les actions liées à la réception d'un signal donné, un nouvel état est atteint et le processus se met en attente d'un autre signal.

Les processus peuvent soit exister au moment de la création du système, soit être créés suite à une demande de création émise par un autre processus. En outre, les processus peuvent durer indéfiniment ou s'arrêter du fait du déclenchement d'une action d'arrêt.

Une définition de processus représente la spécification d'un type de processus; plusieurs instances du même type de processus peuvent être créées et exister simultanément; elles peuvent fonctionner indépendamment et concurremment. Une définition de processus comprend les points suivants (dont certains sont facultatifs):

- nom de processus;
- une paire de nombres entiers. Le premier de ces nombres entiers spécifie le nombre d'instances de processus créées lors de la création du système; s'il est omis, il a une valeur par défaut de 1; le second entier spécifie le nombre maximal d'instances de processus simultanées; s'il est omis, la valeur par défaut maximale est illimitée;
- paramètres formels: liste d'identificateurs de variables associés à leurs types qui est utilisée pour transmettre l'information au processus au moment de la création. Dans la demande de création de processus, une liste de paramètres réels peut être donnée à cet effet. Les valeurs des paramètres formels de processus créés au moment de la création du système sont indéfinies;
- ensemble de signaux d'entrée valides: liste d'identificateurs de signaux définissant les signaux que le processus peut recevoir;
- définitions de signaux: spécification des signaux qui peuvent être échangés entre instances du même processus ou entre services de ce processus (§ D.5.3);

- définitions de procédures: spécification des procédures qui peuvent être appelées par le processus. Une référence de procédure peut être utilisée dans ce contexte (les procédures font l'objet du § D.3.9);
- définition de données: spécifications des nouveaux types, syntypes et générateurs définis par l'utilisateur et localisés dans le processus;
- définition de variables: déclarations des variables du processus. A titre facultatif, on peut indiquer qu'une variable peut être partagée avec plusieurs autres processus du même bloc (variable REVEALED) ou exportée vers d'autres processus ainsi que vers d'autres blocs (variable EXPORTED). Pour chaque variable déclarée, il faut spécifier l'identificateur de sa sorte. Une variable initiale peut facultativement être spécifiée;
- définitions de visibilité: déclarations d'identificateurs de variable qui peuvent servir à l'obtention des valeurs de variables appartenant à d'autres instances de processus. Pour chaque identificateur de variable, la sorte de variable doit être spécifiée;
- définitions d'import: spécification d'identificateurs de variables appartenant à d'autres processus et que le processus veut importer. Pour chaque identificateur, il convient de spécifier la sorte de la variable;
- définition de temporisateur (timer): fait l'objet du § D.3.11;
- définitions de macros: on trouvera des directives sur l'utilisation des macros au § D.5.1;
- corps de processus: spécification du comportement réel du processus exprimé à l'aide d'états, entrées, sorties, tâches, etc. Si le processus est divisé en sous-parties (services), la définition du processus comporte une section de décomposition en services en lieu et place du corps du processus. On trouvera des directives à ce sujet au § D.5.3.

Des exemples et des explications concernant les données, les variables, les définitions de visibilité et les définitions d'import sont donnés au § D.3.10.

On trouvera dans la figure D-3.8.1 un exemple partiel de définition de processus en LDS/PR (les mots clés du langage sont écrits en lettres majuscules).

```

PROCESS p1 (2,100);
    FPAR var1,var2 sort1, var3 sort2, var4 sort3; } Paramètres formels
    SIGNALSET s1,s2,s3;                          } Ensemble des signaux d'entrée valides
    SIGNAL s4,s5;                                }
    SIGNAL s6 (sort6,sort7);                     } Définitions de signaux
    PROCEDURE ...                               } Définitions de procédure
    ... datatype definitions ...                 } Définitions de type de données
    DCL ...                                     } Définitions de variables
    ... process body ...                       } Corps de processus
ENDPROCESS p1;

```

FIGURE D-3.8.1

**Exemple partiel de définition de processus en LDS/PR**

Le corps de processus représente le graphe réel de la machine à états finis. Il consiste en une séquence d'instructions ordonnées en LDS/PR et, en LDS/GR, c'est une séquence de symboles reliés par des arcs orientés (similaires à un organigramme). La spécification du corps de processus doit toujours commencer par l'indication START suivi d'un ensemble d'actions (transition). L'interprétation d'une instance de processus commence à la création de cette instance de processus.

Les actions qui peuvent être exécutées dans une transition sont les suivantes:

- tâche: affectation de variable (ou texte informel);
- exportation: exportation de variable;
- initialisation (set): demande d'activation d'un temporisateur;

- réinitialisation (reset): réinitialisation d'un temporisateur;
- sortie: émission d'un signal en direction d'un autre processus;
- demande de création: création d'une instance d'un type de processus spécifié;
- décision: sélection d'un ensemble d'actions dépendant d'une question;
- appel de procédure: demande d'interprétation d'un ensemble d'actions séparé autonome (même usage que dans des langages de programmation);
- branchement (join): spécification d'un «saut» vers un autre ensemble d'actions.

Une transition peut se terminer par l'une des actions suivantes:

- nextstate (état suivant): spécification de l'état dans lequel se trouvera l'instance de processus;
- arrêt: arrêt immédiat de l'instance de processus.

Après la spécification de l'action de départ et de la transition de départ facultative, le corps du processus comprend les définitions de tous ses états possibles. Chaque définition d'état commence par la spécification des stimuli possibles, attendus par le processus dans cet état. Les stimuli possibles sont les suivants:

- entrées: signaux qui peuvent être reçus;
- mises en réserve: signaux qui doivent être mis en réserve pour traitement ultérieur;
- conditions de validation: décrites au § D.3.8.5;
- signaux continus: décrits au § D.3.8.5.

Une transition doit être spécifiée en correspondance avec chaque stimulus sauf en ce qui concerne les mises en réserve. De telles transitions représentent la séquence d'actions que le processus exécutera si le stimulus apparaît.

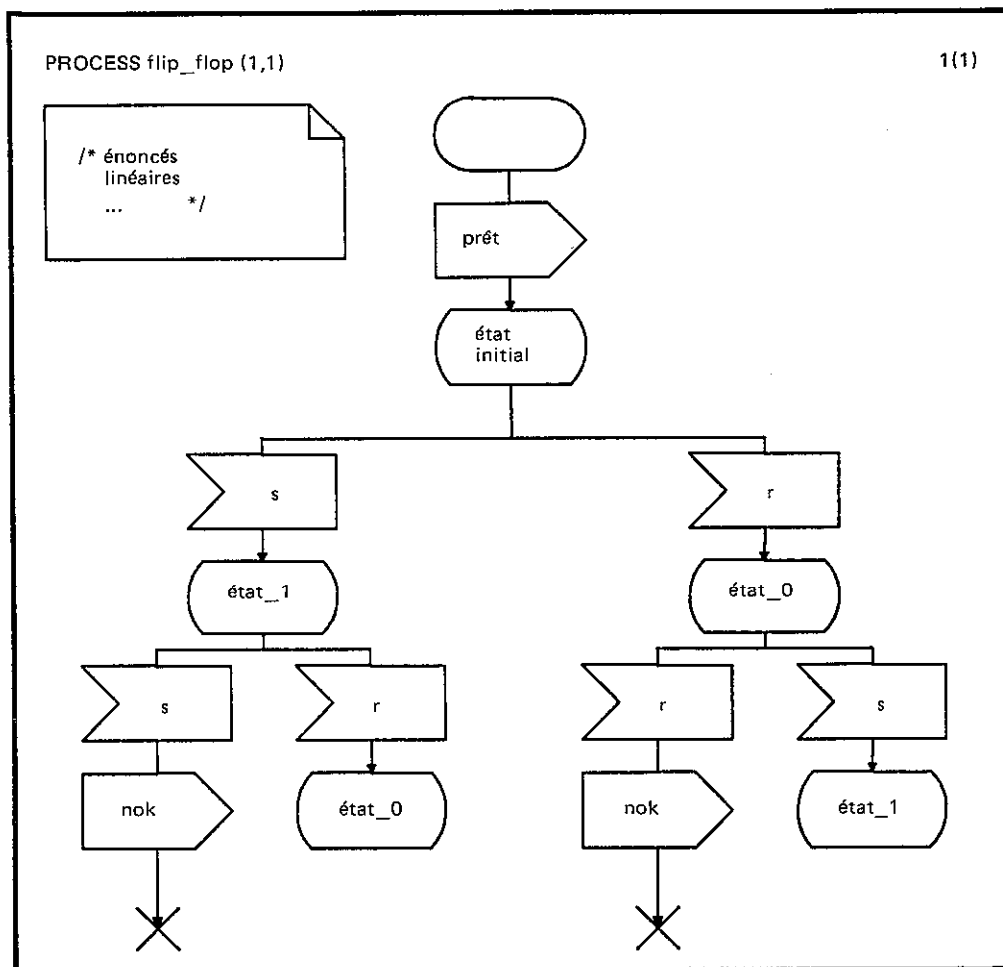
Si un processus exécute une action d'arrêt et que des signaux émis mais non encore reçus se trouvent en suspens, ces signaux sont mis au rebut.

En LDS/GR, une définition de processus est représentée à l'aide du diagramme de processus. Un diagramme de processus se compose des éléments suivants:

- un symbole de cadre: symbole de forme rectangulaire contenant tous les autres symboles. Si aucun acheminement du signal n'est rattaché au symbole de cadre, celui-ci peut être omis;
- l'en-tête du processus: le mot clé PROCESSUS suivi de l'identificateur de processus, puis éventuellement de la spécification des instances de processus et de la spécification des paramètres formels. L'en-tête de processus est placé dans l'angle supérieur gauche du cadre;
- une numérotation de page facultative (placée dans l'angle supérieur droit);
- des symboles de texte: dans le cas d'un diagramme de processus, un symbole de texte peut être utilisé pour contenir des définitions de signal, de variable, de visibilité, d'importation, de données et de temporisateur (timer);
- références de procédure: symbole de procédure contenant un nom de procédure représentant une procédure du processus qui est définie séparément;
- diagrammes de procédure: un pour chaque procédure locale du processus qui n'est pas référencée;
- la zone de graphe de processus: spécification du comportement du processus en termes de départ, d'états, d'entrées, de sorties, de tâches, . . . et d'arcs orientés. Si le processus est structuré en services, la zone du graphe de processus contient la spécification des services ou leurs références (voir le § D.5.3). Les symboles en GR utilisés pour le corps du processus sont indiqués dans le résumé du LDS/GR;
- diagrammes de macro: on trouvera au § D.5.1 des directives sur l'utilisation des macros.

La figure D-3.8.2 donne un exemple de définition de processus en LDS/GR; on trouvera des explications et des exemples complémentaires sur les graphes de processus dans les paragraphes qui suivent.





**FIGURE D-3.8.2**  
**Exemple de diagramme de processus**

Si un graphe de processus ne tient pas sur une seule page, le diagramme peut être présenté sur plusieurs pages; il faut noter:

- qu'une numérotation de page contenant le numéro de la page et le nombre total de pages devraient être fournis, c'est-à-dire: 1 (9);
- que le symbole de brachement (join) ou le symbole d'état suivant (nextstate) peut être utilisé pour représenter des connexions entre différentes parties du graphe du processus.

Un bon critère pour diviser un graphe de processus en plusieurs parties consiste à représenter une définition d'état sur chaque page. Si une définition d'état ne tient pas sur une seule page, il est possible d'utiliser le symbole de brachement pour représenter des connexions avec une partie du graphe qui se trouve sur une autre page.

#### D.3.8.1 *Création de processus*

Comme indiqué au paragraphe précédent, des processus (c'est-à-dire des instances de processus) peuvent être créés à la suite d'une demande explicite ou lors de la création du système.

La demande explicite de création ne peut être formulée que par un autre processus du même bloc que le processus à créer; elle permet la spécification de paramètres réels pour le transfert de l'information vers la nouvelle instance créée. La figure D-3.8.3 donne un exemple de création en PR et en GR.

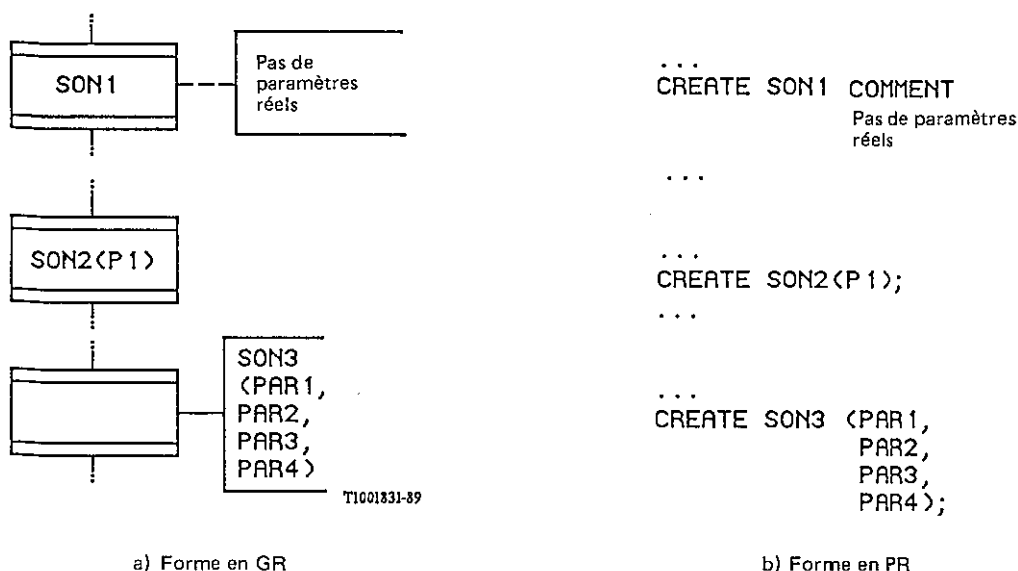


FIGURE D-3.8.3  
Exemples de demandes de création en LDS/GR et LDS/PR

Si une ou plusieurs instances de processus d'un type de processus donné sont créées au moment de la création du système, la définition de ce (ou ces) processus ne doit pas avoir accès aux paramètres formels car ils seront indéfinis. En conséquence, chaque définition de processus ayant des paramètres formels doit également faire le nécessaire pour interdire l'accès à ses paramètres formels avant que ceux-ci aient reçu une valeur ou qu'ils comportent explicitement la spécification d'aucune instance de processus au moment de la création du système (voir la figure D-3.8.4).

```
PROCESS explicitly_created_proc_1 (0, ...);
  FPAR ...
...
ENDPROCESS explicitly_created_proc_1;
```

FIGURE D-3.8.4  
Définition d'un processus comportant la spécification de paramètres formels

Lors de la création d'un processus, il est possible de déterminer des valeurs d'instance de processus à l'aide des expressions prédéfinies suivantes:

- (OFFSPRING) DESCENDANT: renvoie la valeur PID de la dernière instance créée,
- SELF: renvoie la valeur PID de l'instance de processus proprement dite,
- (PARENT) ASCENDANT: renvoie la valeur PID de l'instance qui procède à la création
- (SENDER) ÉMETTEUR: renvoie la valeur PID du processus émettant le dernier signal utilisé.

Lorsqu'un processus est créé par suite de la création du système, seule l'expression SELF renvoie une valeur PID; les expressions OFFSPRING et PARENT donnent la valeur NULL.

De telles valeurs revêtent une grande importance lorsqu'il existe plusieurs instances de processus du même type de processus car elles constituent le seul moyen d'adresser sans ambiguïté les signaux aux instances. En fait, comme cela est mieux expliqué au § D.3.8.6, lorsqu'un processus émet un signal, il doit spécifier l'instance de destination, à moins que celle-ci ne puisse être déterminée sans ambiguïté.

L'utilisateur doit veiller à ce que les instances de processus créées puissent, au besoin, communiquer les unes avec les autres. Pour y parvenir, il faut souvent prévoir certains types de processus d'initialisation. Ces processus, créés en même temps que le système, devront créer les autres processus et éventuellement communiquer les valeurs PID appropriées à tous les processus auxquels elles seront nécessaires.

Il convient de noter d'autres points importants concernant la création de processus:

- 1) après la création du système, les processus ne peuvent être créés que par un autre processus du même bloc. Une possibilité de création d'un processus dans d'autres blocs consiste à avoir un processus spécial dans chaque bloc; ce processus provoque la création d'un processus à la réception d'un signal provenant d'un processus d'un autre bloc;
- 2) après leur création, les processus ont une durée de vie qui leur est propre. Ils ne cessent d'exister que lorsqu'ils accomplissent une action d'arrêt pendant une transition. On peut construire des systèmes qui autorisent les opérations extérieures d'élimination de processus en employant un signal spécial d'élimination. A la réception du signal d'élimination, le processus accomplit une action d'arrêt.

La relation entre les processus créateurs et les processus créés peut être représentée dans un diagramme de bloc à l'aide de symboles de lignes de création, comme indiqué dans la figure D-3.8.5.

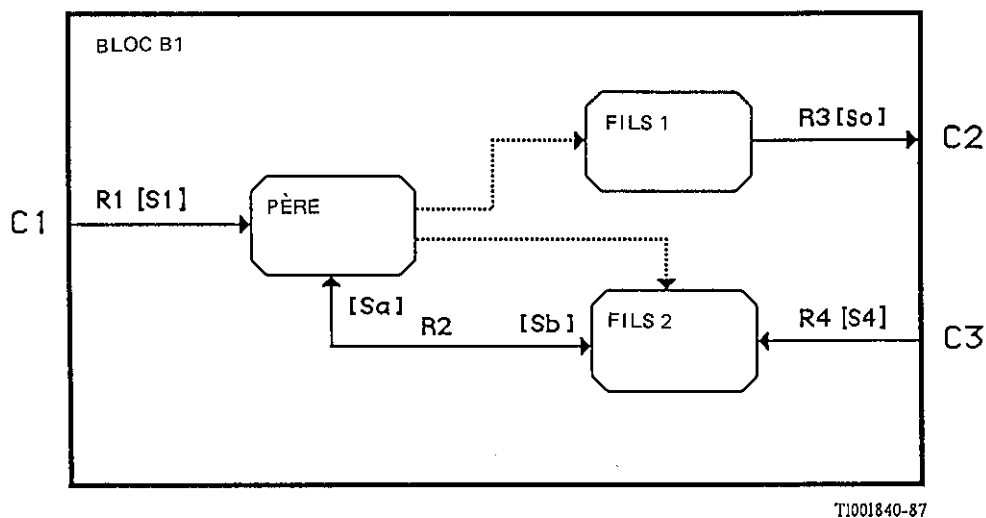


FIGURE D-3.8.5

Exemple de diagramme de bloc avec symboles de lignes de création

### D.3.8.2 Etats

Un état est une étape du processus pendant laquelle aucune action ne s'accomplit, mais où la file d'entrée contrôle l'arrivée des signaux entrants. Grâce à l'identificateur de signaux que contient le signal d'entrée, l'arrivée du signal fait sortir le processus de l'état et lui fait accomplir une succession donnée d'actions. Un signal qui est arrivé et a entraîné une transition a été «absorbé» et cesse d'exister.

En LDS/PR, un état est représenté par le mot clé STATE suivi du nom de l'état. La spécification de l'état se termine soit à l'énoncé d'état suivant soit à la fin du processus ou au moyen du mot clé explicite ENDSTATE (FIN D'ÉTAT). On peut utiliser un astérisque dans l'énoncé d'état au lieu du nom de l'état; il s'agit là d'une notation abrégée indiquant que les entrées pour les mises en réserve suivantes et les transitions correspondantes doivent être interprétées pour chaque état.

Le mot clé NEXTSTATE (ÉTAT SUIVANT), suivi du nom de l'état, est utilisé pour indiquer l'état qui suit. On peut employer un tiret dans l'énoncé d'état suivant au lieu du nom de l'état pour indiquer que l'état suivant est précisément l'état dont la transition actuelle tire son origine.

La figure D-3.8.6 donne un exemple partiel en LDS/PR.

```

SYSTEM s;
...
BLOCK b;
...
PROCESS p;
...
START;
...
STATE st1;
...
STATE st2;
...
NEXTSTATE st1;
...
ENDSTATE st2;
STATE st3;
...
ENDPROCESS p;
ENDBLOCK b;
ENDSYSTEM s;

```

FIGURE D-3.8.6

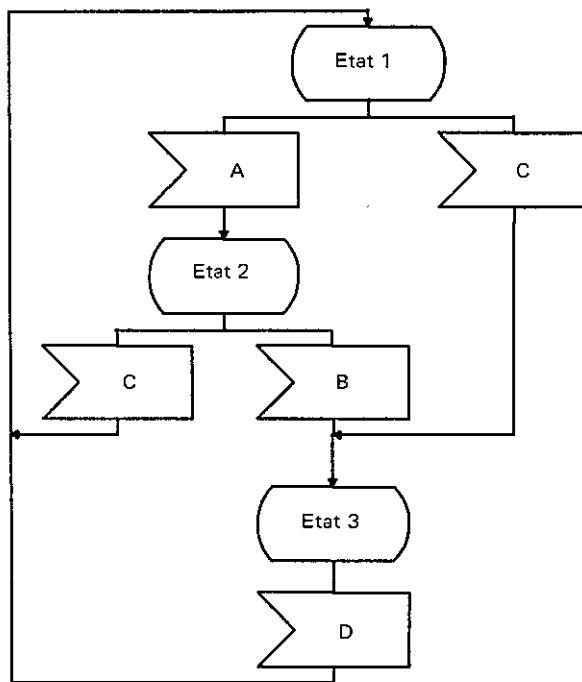
**Exemple partiel du LDS/PR pour des définitions d'état**

En LDS/GR, un état est représenté à l'aide d'un symbole d'état contenant le nom d'état et est relié aux symboles d'entrées ou de mises en réserve. Le même symbole d'état, avec une flèche d'arrivée, sert à représenter l'énoncé état suivant.

Par commodité, pour simplifier le dessin ou faciliter la compréhension, le même état peut apparaître plusieurs fois dans un diagramme en LDS. Si tel est le cas, le diagramme est considéré comme entièrement équivalent au diagramme qui résulterait, de la fusion de toutes les apparitions multiples du même état. Les figures D-3.8.7 et D-3.8.8 donnent des exemples de cette situation. Dans la figure D-3.8.7 b), on utilise un symbole d'état comme lien avec l'état principal portant le même nom, lorsqu'il est mentionné dans le symbole d'état suivant. Dans la figure D-3.8.8 un état est représenté par des symboles multiples n'ayant chacun qu'un sous-ensemble des entrées (ou des mises en réserve).

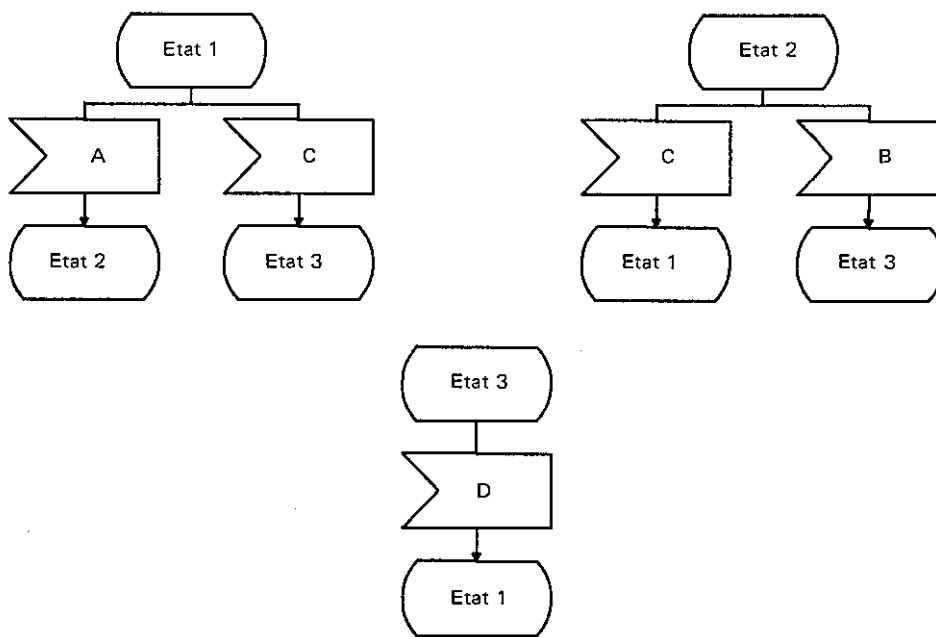
Les diagrammes a) et b) de la figure D-3.8.7 sont logiquement équivalents. Le diagramme a) contient une seule apparition de chaque état tandis que le diagramme b) utilise des apparitions multiples. Dans le diagramme b), l'état a une apparition principale, dans laquelle tous ses symboles d'entrées (et de mises en réserve) associés sont indiqués. Lorsque cet état peut être atteint à partir d'autres points du diagramme (par exemple le point de terminaison d'une transition), il est indiqué comme un état sans entrée ni mise en réserve associée. Un commentaire du symbole d'état suivant se référant au symbole d'état améliorera la clarté, notamment lorsqu'ils apparaissent sur des pages différentes.

La figure D-3.8.8 utilise les apparitions multiples d'un état pour constituer l'ensemble total d'entrées (et de mises en réserve). Chaque apparition de l'état est indiquée uniquement avec un sous-ensemble de ces entrées.



T1001850-87

FIGURE D-3.8.7 a)  
Apparition unique d'un état



T1001860-87

FIGURE D-3.8.7 b)

Diagramme a) indiquant les principaux états et les états ultérieurs utilisés comme connecteurs avec les états

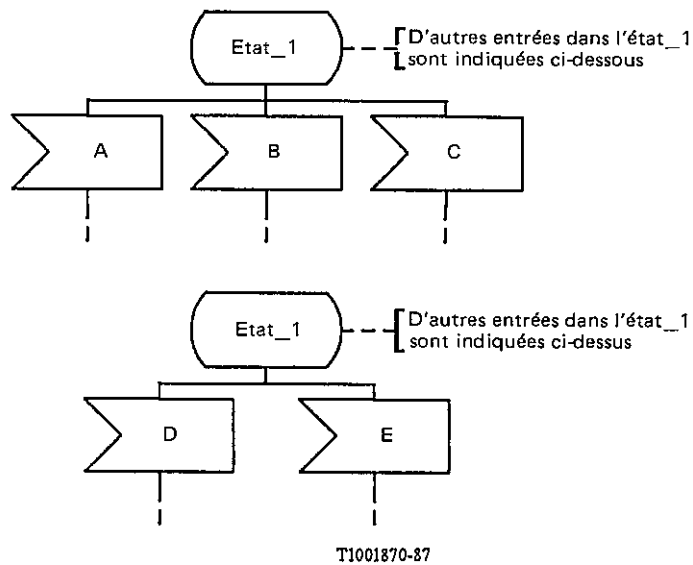


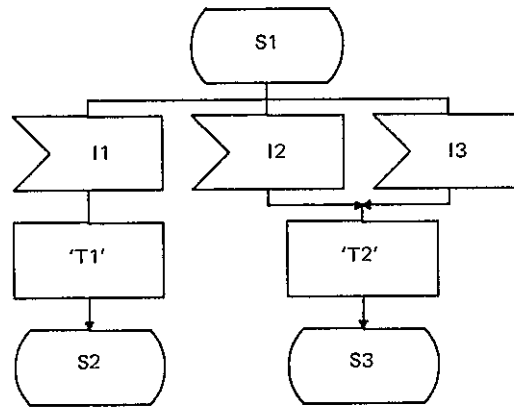
FIGURE D-3.8.8

**Apparitions multiples d'un état pour le cas où toutes les entrées ne peuvent être clairement indiquées par le même symbole**

Cette approche a été appliquée avec succès lorsque des états avaient un nombre relativement grand d'entrées ou de mises en réserve mais présentaient le risque que le lecteur interprète de façon erronée le diagramme s'il n'était pas conscient de l'existence d'occurrences multiples. Pour éviter ce malentendu, les états n'indiquant qu'un sous-ensemble d'entrées/mises en réserve devraient être accompagnés d'un commentaire indiquant une référence à d'autres états avec leurs entrées et mises en réserve associées, comme indiqué dans la figure D-3.8.8.

Des apparitions multiples peuvent être utilisées pour attirer l'attention du lecteur sur certains aspects (par exemple la séquence normale des signaux traités), laissant d'autres aspects pour d'autres pages (par exemple le traitement de situations d'alarme).

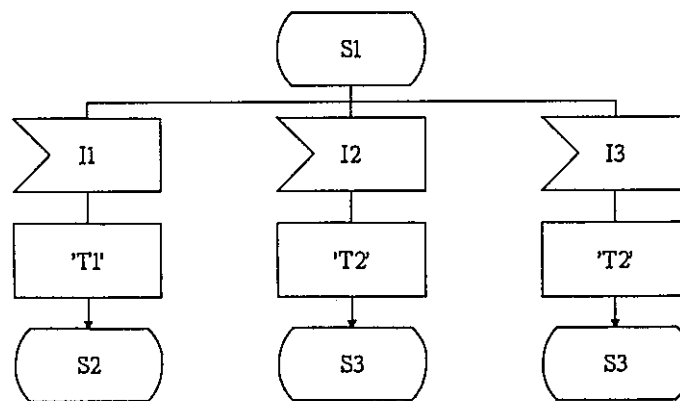
Au cours d'une transition un processus ne sait pas explicitement quel signal d'entrée a occasionné la transition. Seul le contexte permet de le déduire (c'est-à-dire que cette transition ne peut se produire qu'en cas de réception d'un signal donné). Dans la figure D-3.8.9, la tâche T1 ne s'accomplit qu'en cas de réception de I1. Toutefois, la tâche T2 s'accomplit en cas de réception de I2 ou de I3. S'il importe que T2 sache quel signal d'entrée a été reçu, il est souhaitable de concevoir le processus comme l'illustre la figure D-3.8.10.



T1001880-87

FIGURE D-3.8.9

**Exécution d'une tâche qui dépend de deux signaux reçus sur trois mais qui est indépendante de chacun**



T1004870-89

FIGURE D-3.8.10

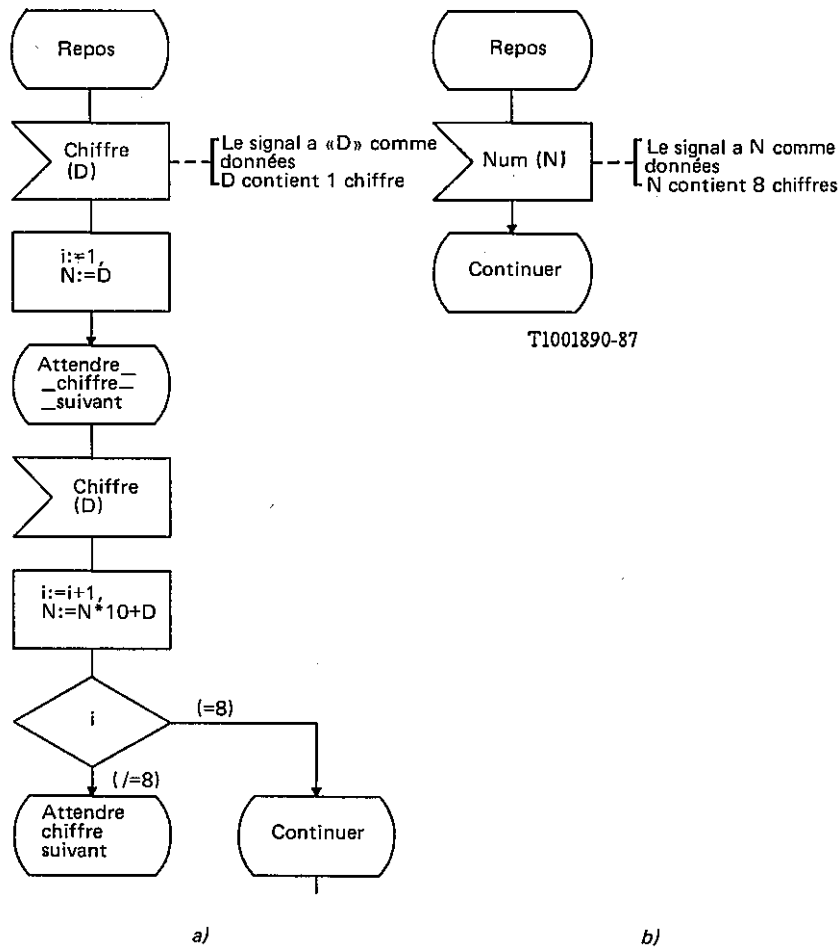
**Exécution d'une tâche qui dépend de deux signaux reçus**

#### D.3.8.2.1 Détermination des états requis

L'auteur d'un diagramme LDS dispose généralement d'une certaine latitude pour définir les états d'un processus. Il peut avoir besoin d'une stratégie qui lui permette d'identifier les états du processus et cette stratégie peut être informelle ou formelle. Un jugement sûr (que seule une longue expérience permet d'acquérir) est nécessaire si l'on veut établir des diagrammes LDS tels que l'identification d'un trop grand nombre d'états ne les complique pas inutilement ou qu'un nombre artificiellement réduit d'états ne les empêche d'exploiter les avantages qu'offre le LDS. Avant que l'auteur ne commence à établir le diagramme, certaines étapes préliminaires (étudiées au § D.3.2) doivent être achevées, par exemple:

- la structuration du système en blocs fonctionnels;
- la représentation d'un ou plusieurs processus LDS par bloc;
- le choix des signaux d'entrée et de sortie;
- l'utilisation des données dans le processus.

Tous les facteurs ci-dessus exercent un effet important dans la détermination des états de chaque processus. L'effet qu'exerce le choix des signaux d'entrée sur le nombre d'états d'un diagramme LDS est illustré par les deux exemples de la figure D-3.8.11.



*Remarque* — Les exemples a) et b) représentent la même fonction à différents niveaux de détail. Le nombre des états varie en conséquence.

FIGURE D-3.8.11

**Réception d'un numéro téléphonique à huit chiffres**

D.3.8.2.2 Réduction du nombre des états

Ayant appliqué une stratégie pour l'identification des états d'un processus, l'auteur d'un diagramme LDS estimera peut-être qu'un trop grand nombre d'états ont été utilisés. Le nombre des états est important car la dimension et la complexité d'un diagramme LDS sont souvent étroitement liées à ce nombre. Il existe certes des moyens qui permettent de réduire le nombre des états mais le fait qu'un diagramme LDS soit complexe n'est pas, en soi, une raison justifiant sa modification; en effet, la complexité du diagramme peut être simplement due à la complexité inhérente au processus défini. Le choix de l'ensemble d'états doit généralement offrir le plus de clarté possible à la séquence d'interactions entre le processus et son environnement. Ce n'est d'ordinaire pas en minimisant le nombre d'états que l'on obtient cette clarté. Le nombre de séquences indépendantes traitées par un processus exerce un effet multiplicateur sur le nombre d'états. Il est donc souhaitable que les séquences indépendantes des processus séparés soient traitées de façon à réduire le nombre d'états et à obtenir une plus grande clarté.

On peut réduire le nombre des états en effectuant la séparation des fonctions communes ou la fusion des états ou encore en recourant au concept de service. Certaines structures de données peuvent également conduire à une réduction du nombre des états. Pour d'autres représentations, l'emploi des macros peut être avantageux.



### D.3.8.2.3 Séparation des fonctions communes

Lors de la mise en place d'un diagramme LDS, on peut constater que la définition d'un aspect particulier et répétitif d'un processus nécessite la représentation d'états répétitifs. La figure D-3.8.12 représente une partie d'un diagramme LDS définissant un processus de signalisation de ligne et illustrant la condition selon laquelle une tonalité de signalisation de ligne doit être présente pendant une certaine durée avant que l'on considère que le signal de ligne a été détecté.

Pour spécifier cet aspect, il convient d'insérer un état intermédiaire entre les états `aucun_signal_de_ligne_n'est_détecté` et `conversation`. Supposons que, dans un diagramme complet, une telle fonction commune doit être répétée à chaque point où le signal est détecté. Une autre solution consiste à définir un processus séparé qui est responsable du contrôle de la tonalité de signalisation de ligne et de la détection des signaux sur la base du temps de reconnaissance spécifié. L'existence de ce nouveau processus permettrait de représenter le diagramme de la figure D-3.8.12 de la manière indiquée dans la figure D-3.8.13. (Dans un contexte donné, les figures peuvent être rendues équivalentes moyennant l'introduction d'un nouveau signal `signal_de_ligne_valide`.)

Il convient de noter la légère différence entre le processus de la figure D-3.8.12 et les deux processus de la figure D-3.8.13.

Le processus de la figure D-3.8.12 commence à compter dès réception de T1 alors que le processus de traitement des appels [processus b) de la figure D-3.8.13] commence à compter dès réception de `signal_de_ligne_valide` qui est engendré et émis à la réception de T1 par le processus a) de la figure D-3.8.13. Il s'ensuit que, dans le second cas, le comptage démarre après T1 + temps de génération, d'émission et de réception du signal. En outre, d'autres signaux peuvent avoir été mis dans la file d'attente pendant le temps nécessaire à la génération et à l'émission de `signal_de_ligne_valide`.

Une seconde particularité est que cette séparation d'une sous-fonction ne serait pas possible si le signal que doit recevoir la sous-fonction devait être reçu également par la fonction principale, un signal étant toujours acheminé vers un seul processus. Si, dans l'exemple, le même signal Sff devait être reçu dans un autre état où il n'est pas besoin de le valider pour le temps T1, il n'aurait pas été possible de séparer la sous-fonction de validation en un autre processus.

Les solutions qui font appel à des processus distincts sont généralement utiles lorsque les signaux doivent être traités indépendamment des états dans le processus principal. Dans ce cas, les opérations qui précèdent et qui suivent les processus peuvent traiter les séquences détaillées et décharger un processus principal de tous les détails. Ceci engendre souvent une modularité utile permettant, par exemple, d'isoler les caractéristiques particulières des systèmes de signalisation, du processus principal plus axé sur le service.

Une autre manière de traiter le problème consiste à appliquer le concept de service, expliqué au § D.5.3.

Une solution différente du problème consisterait à employer la notation MACRO, comme indiqué à la figure D-3.3.1. Dans ce cas, on obtient pour le diagramme la compacité voulue sans modifier en rien la sémantique du diagramme original. Il est en outre possible d'appeler la MACRO à partir de plusieurs états si la logique du processus l'exige.

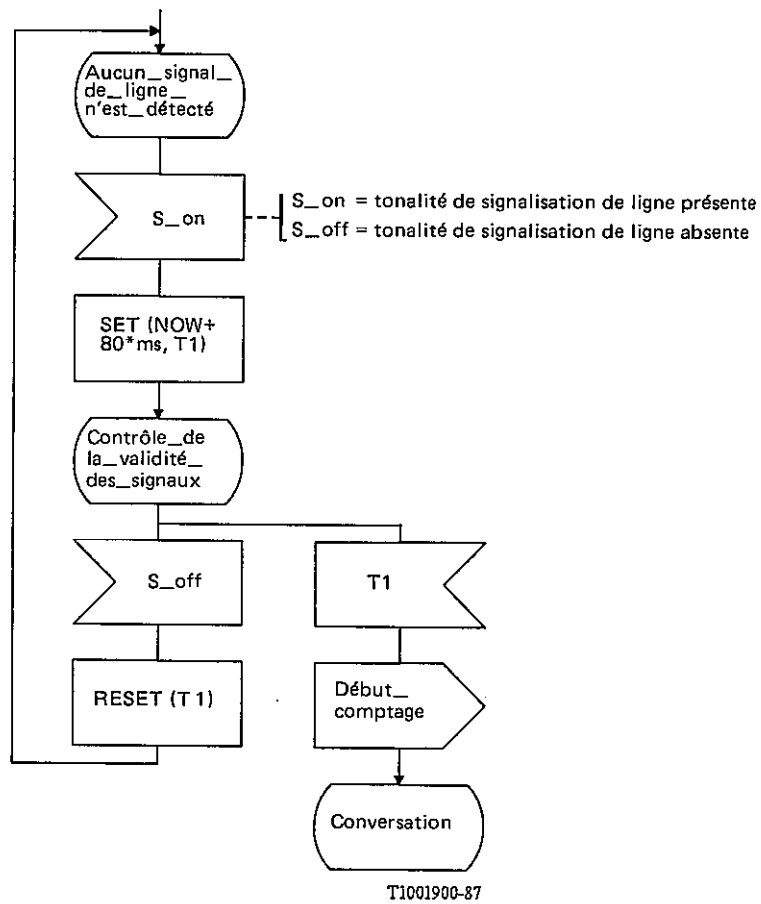
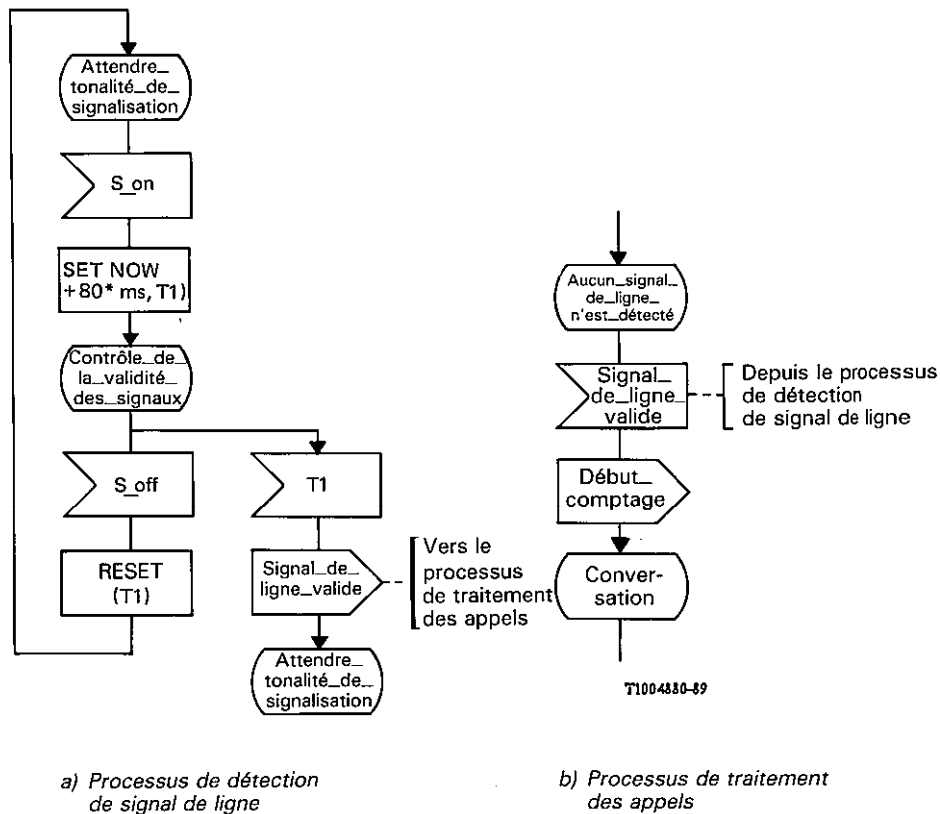


FIGURE D-3.8.12

Exemple de diagramme LDS correspondant à une fonction composite de traitement des appels et de détection de signal de ligne



**FIGURE D-3.8.13**  
**Exemple de ségrégation d'une fonction commune**  
**(détection de signal de ligne)**  
**afin d'éviter des états répétitifs tels que le contrôle de la validité des signaux**

D.3.8.2.4 Fusion des états

Si, dans un diagramme LDS, la destination future de deux états est la même, on peut, indépendamment de leur évolution antérieure, effectuer la fusion de ces deux états en un seul, sans affecter la logique du diagramme.

La figure D-3.8.14 représente une partie d'un diagramme LDS comportant deux états dont le «passé» est différent mais dont le «futur» est identique. Dans la figure D-3.8.15, les deux états ont été combinés en un seul. Il s'agit là d'un exemple relativement simple dans lequel la réduction de la complexité est peu importante, mais cette même technique peut être utilisée pour obtenir une plus grande simplification. La sémantique du LDS ne prévoit pas une décision consécutive à un état pour déterminer le «passé» du processus antérieurement à cet état (c'est-à-dire de déterminer, pour cet exemple, si A6 ou B4 a été émis), à moins que cette information n'ait été explicitement stockée avant l'entrée dans l'état. A noter, que l'attribution d'un nom aux données d'une entrée a pour effet de mettre la valeur en mémoire.

Un état doit représenter une situation logique du processus et il n'est donc pas souhaitable d'effectuer la fusion de situations logiques différentes en un seul état.

Des précautions sont à prendre si un diagramme fusionné doit être modifié ultérieurement. L'utilisateur devrait rechercher si la modification envisagée a ou non le même effet sur les diverses branches initiales.

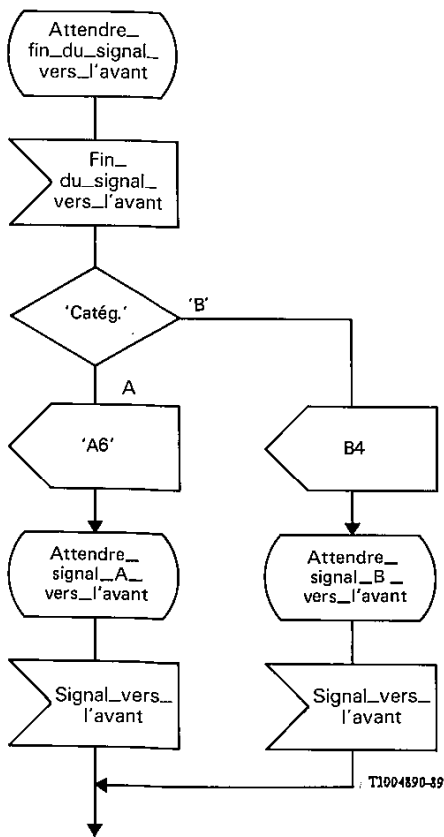


FIGURE D-3.8.14  
Exemple d'une partie d'un diagramme LDS  
avant la fusion des états

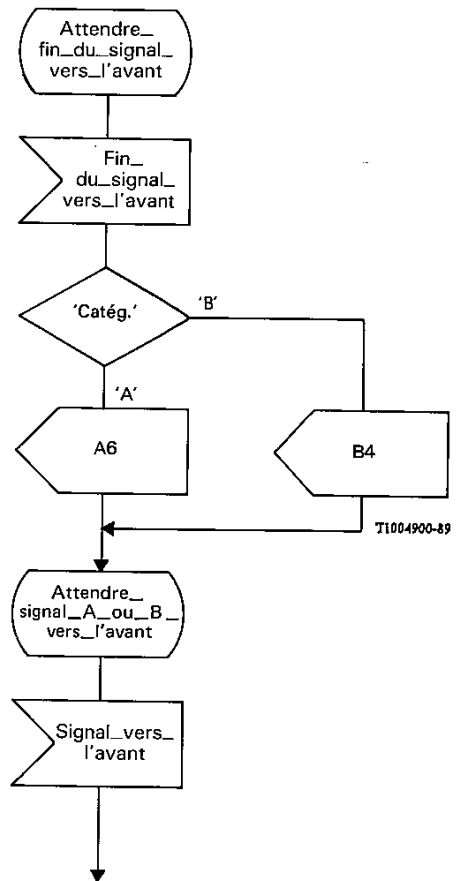


FIGURE D-3.8.15  
Exemple d'une partie d'un diagramme LDS  
après fusion des états

### D.3.8.3 Entrées

Le présent § D.3.8.3 a pour objet d'expliquer la notion d'entrée ainsi que l'utilisation des entrées dans des diagrammes LDS ne faisant pas appel à la notion de mise en réserve. La notion de mise en réserve et l'utilisation de cette notion en même temps que la notion d'entrée font l'objet du § D.3.8.4.

#### D.3.8.3.1 Considérations générales

Un symbole d'entrée attaché à un état signifie que, si le signal dont le nom figure dans le symbole d'entrée arrive pendant que le processus est dans cet état, il faut interpréter la transition qui suit le symbole d'entrée. Lorsqu'un signal a déclenché l'interprétation d'une transition, ce signal n'existe plus et on dit qu'il a été absorbé.

Un signal peut être accompagné de données associées. Par exemple, un signal portant le nom «chiffre» sert non seulement à déclencher l'exécution d'une transition par le processus de réception mais encore à véhiculer la valeur du chiffre (0 à 9), ces données pouvant être utilisées par le processus récepteur.

En LDS/PR, l'instruction INPUT contient une liste d'identificateurs de signaux. Les valeurs contenues dans les signaux sont indiquées à l'aide d'identificateurs de variables. Les identificateurs de variables doivent être du type indiqué dans la définition de signal, de sorte que leur position est très importante. Ces identificateurs de variables sont placés entre parenthèses et séparés par des virgules (voir la figure D-3.8.16). Si une ou plusieurs valeurs de signal sont rejetées, les variables correspondantes font défaut, ce qui est indiqué par deux ou plusieurs virgules consécutives (figure D-3.8.17).

```

...
SIGNAL sig1 (INTEGER,BOOLEAN,INTEGER);
...
PROCESS ...
...
  DCL a INTEGER, b BOOLEAN, c INTEGER; /* declarations */
...
  STATE st1;
    INPUT sig1(a,b,c); /* a correct input */
...
  STATE st2;
    INPUT sig1(a,c,b); /* an incorrect input */
...
ENDPROCESS ...

```

FIGURE D-3.8.16  
Énoncés d'ENTRÉE

```
INPUT a(var1,var2,,var4);
```

*Remarque* — Dans cet énoncé, la troisième valeur du signal est mise au rebut.

FIGURE D-3.8.17  
Entrée d'un signal dont 3 valeurs seulement sont définies sur 4

En LDS/GR, une entrée est représentée à l'aide d'un symbole d'entrée contenant la liste d'identificateurs de signaux et les identificateurs de variables correspondants pour les valeurs acheminées. Pour pouvoir être communiquées au processus, ces valeurs doivent être désignées nommément dans les symboles d'entrée, entre parenthèses.

On trouvera des exemples de réception de valeurs des entrées dans les figures D-3.8.18, D-3.8.19 et D-3.8.20.

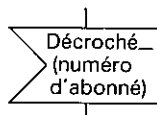
Les données auxquelles un nom est assigné peuvent être utilisées par le processus récepteur quand l'entrée est interprétée.

La figure D-3.8.18 montre la réception du signal «décroché». Le signal «décroché» est accompagné de données associées (numéro\_de\_l'abonné) avec pour valeur 9269. Le signal peut être reçu de trois manières, comme indiqué en a) et c) de la figure.

La figure D-3.8.19 montre comment envoyer et recevoir plusieurs valeurs en un seul signal. Chaque élément doit être séparé du suivant par une virgule. La figure D-3.8.20 c) montre comment ignorer les valeurs indésirables en laissant un blanc dans la liste de sortes.

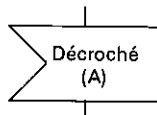
A noter que, dans le signal de sortie, il est impossible d'écrire des expressions pour E1, E2 ou E3, alors que dans le signal d'entrée, il faut employer des variables pour recevoir les valeurs émises.

Dans le LDS, il n'est pas nécessaire de dessiner des symboles d'entrée pour représenter les signaux dont l'arrivée nécessiterait une transition nulle (c'est-à-dire une transition qui ne contient aucune action et qui ramène au même état). La convention admise est la suivante: pour tout signal qui n'est pas représenté dans un symbole d'entrée explicite à un état donné, il existe, dans cet état, un symbole d'entrée implicite et une transition nulle. Conformément à cette convention, les deux diagrammes représentés dans la figure D-3.8.21 sont logiquement équivalents et peuvent être indistinctement utilisés.



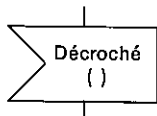
*Remarque* — La valeur 9269 est stockée dans une variable appelée numéro\_d'abonné.

a)



*Remarque* — La valeur 9269 est stockée dans une variable appelée A.

b)



T1004910-89

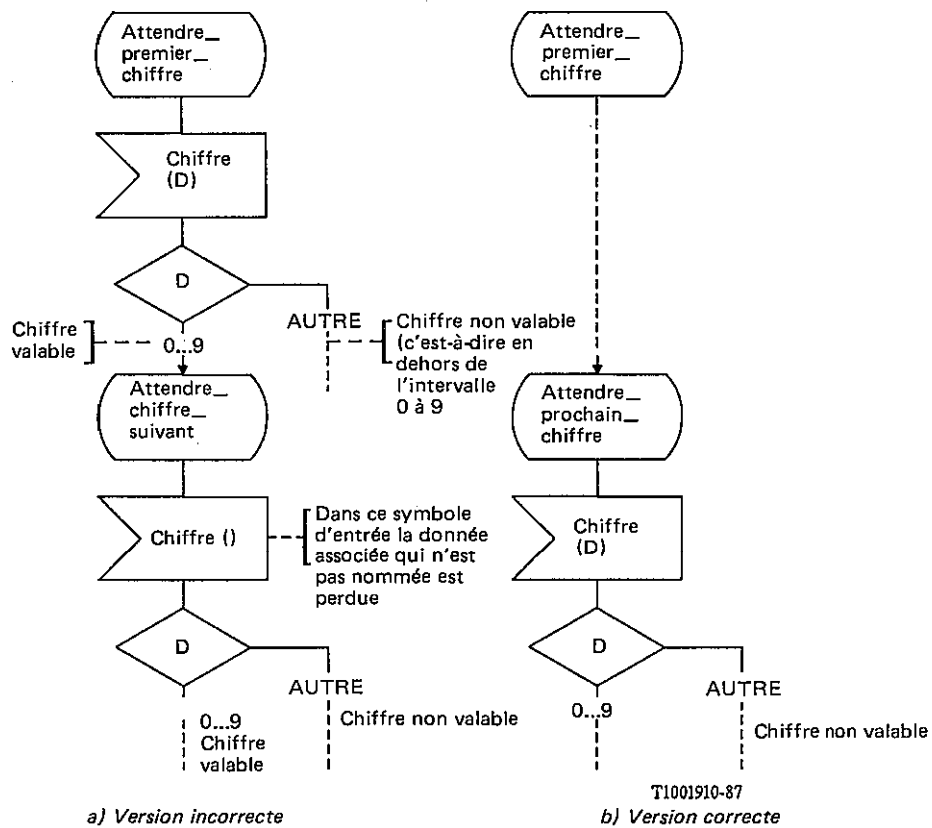
*Remarque 1* — La donnée n'a pas de nom, la valeur 9269 est perdue et le processus de réception ne peut l'obtenir.

*Remarque 2* — Le nom du signal (décroché) doit correspondre au nom du signal de sortie mais les noms des données peuvent correspondre ou non.

c)

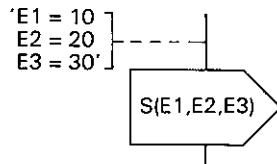
**FIGURE D-3.8.18**

**Exemple de réception des données dans un processus**



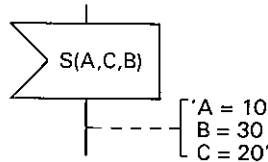
*Remarque 1* – Dans a) on retiendra pour la deuxième décision la valeur de D obtenue avec le premier chiffre.  
*Remarque 2* – Dans b) on retiendra pour D la valeur du chiffre du moment. Les valeurs des chiffres antérieures seront perdues.

FIGURE D-3.8.19  
 Partie d'un processus récepteur de chiffres



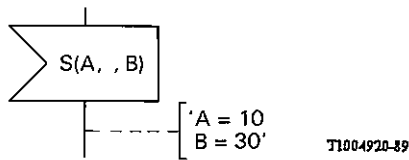
Remarque — Le signal de sorties S a trois variables appelées E1, E2 et E3. Ces éléments correspondent à trois valeurs, dans le cas présent 10, 20 et 30.

a)



Remarque — Le signal d'entrée correspondant S dans le processus de réception nomme respectivement ces éléments A, C et B.

b)

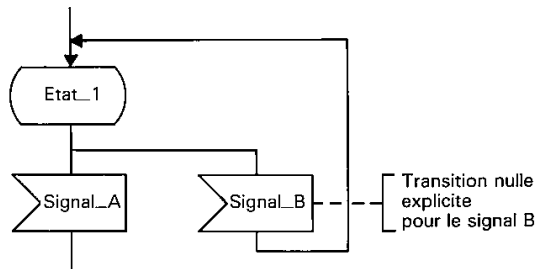


Remarque — Ce signal d'entrée ne nomme que deux variables. La valeur du milieu est perdue.

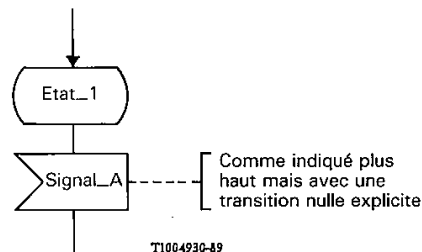
c)

FIGURE D-3.8.20

Signal avec plusieurs données élémentaires associées



a) Transition « nulle » explicite



b) Transition « nulle » implicite

Remarque — Si une donnée est associée au signal B, elle est perdue dans l'un et l'autre cas. Si cependant un nom de donnée apparaît (par exemple, signal\_B(x)) dans le cas a) la valeur de la donnée est maintenue. Les deux cas ne sont plus alors identiques.

FIGURE D-3.8.21

Représentation explicite et implicite d'une transition « nulle »



Lorsqu'un certain nombre d'entrées aboutissent à la même transition, tous les identificateurs de signaux qui s'y rapportent peuvent être placés dans un même symbole d'entrée. Le LDS prévoit une notation abrégée pour représenter une entrée de tous les signaux (valable pour ce processus) non explicitement mentionnés dans cet état. La figure D-3.8.22 illustre cet aspect et les diagrammes qui y sont représentés sont logiquement équivalents. Si tous les identificateurs de signaux sont mentionnés, il faut les séparer par des virgules.

### D.3.8.3.2 Mécanisme implicite de mise en file d'attente

Un ou plusieurs signaux peuvent être en attente d'absorption lorsqu'un processus parvient à un nouvel état. Cela signifie que les signaux doivent être mis en attente d'une certaine manière si l'on veut éviter qu'ils soient perdus. Lorsqu'un signal parvient au bloc de destination, il est placé dans la file d'attente d'entrée du processus de réception. La sémantique du LDS définit pour chaque processus un mécanisme théorique de mise en file d'attente selon lequel le mode de sélection des signaux pour l'absorption par le processus est fondé sur l'ordre d'arrivée des signaux dans ce processus. Lorsque le processus parvient à un état, il reçoit un seul signal en provenance de la file d'attente. Ceci signifie que si la file d'attente n'est pas vide, le processus absorbe le premier signal qui vient de la file d'attente en question. Si cette dernière est vide, le processus demeure en attente dans l'état jusqu'à l'arrivée à la file d'attente d'un signal qui est ensuite absorbé par le processus.

La figure D-3.8.23 utilise le concept de file d'attente pour expliquer le fonctionnement d'un processus LDS dans lequel les temps de transition sont différents de zéro. Il convient de noter les éléments suivants:

- le concept de mise en réserve n'est pas appliqué et les signaux sont absorbés dans l'ordre de leur arrivée;
- l'ordre de succession de l'arrivée des signaux est important. Si «C» était arrivé avant «B» dans la transition entre l'état 1 et l'état 2, la séquence des états aurait été 1, 2, 3 au lieu de 1, 2, 4;

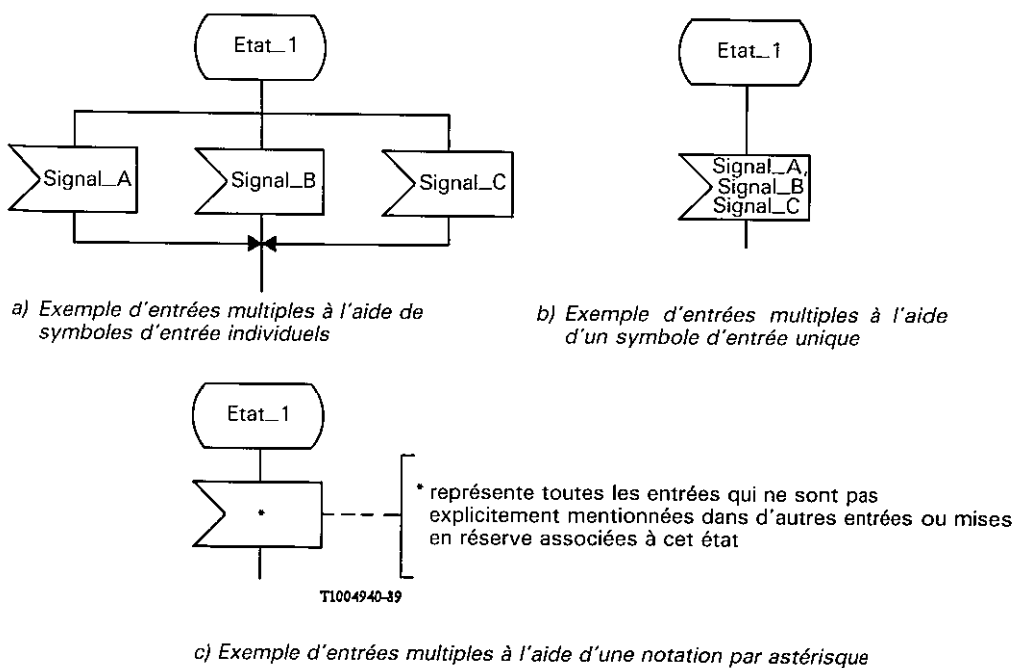


FIGURE D-3.8.22

#### Autre représentation possible des entrées multiples

- la file d'attente n'étant pas vide lorsque le processus arrive aux états 2 et 4, ce processus ne doit attendre ni dans l'un ni dans l'autre de ces états;
- il n'est pas possible d'attribuer la priorité à un signal. Un mécanisme spécial est prévu pour les communications entre services, afin que les signaux échangés entre service soient traités avant les autres signaux (§ D.5.3).

Si les temps de transition sont nuls, chaque signal sera absorbé au moment de son arrivée dans un processus, sauf si l'on a recours à la mise en réserve (voir le § D.3.8.4).

#### D.3.8.3.3 *Réception des signaux qui ne se présentent pas normalement*

Dans chacun des états, tous les signaux possibles doivent être indiqués implicitement ou explicitement. Des exceptions (signaux inattendus mais théoriquement possibles, signaux non définis ou logiquement faux à un endroit donné, etc.) peuvent se présenter dans presque tous les états. Normalement, l'auteur n'indique pas ces possibilités; il s'ensuit qu'un tel signal sera éliminé s'il se présente. Si toutefois l'auteur tient à faire figurer les exceptions dans son diagramme, il doit prévoir pour tous les états une entrée supplémentaire.

Une autre possibilité consiste à profiter des apparitions multiples d'un état portant le symbole «tous» (\*). Par exemple, si le signal A\_RACCROCHÉ peut être reçu dans tous les états et si les actions postérieures sont identiques, on peut recourir à la méthode indiquée à la figure D-3.8.24.

#### D.3.8.3.4 *Arrivée simultanée de signaux*

La Recommandation Z.100 (§ 2) prévoit que les signaux peuvent parvenir simultanément à un processus et précise qu'ils seront placés dans un ordre arbitraire.

Si un usager met au point un processus capable de recevoir des signaux simultanés, il doit veiller à ce que l'ordre d'arrivée ne bouleverse pas le fonctionnement escompté du processus.

Le LDS ne préconise pas de priorité parmi les signaux; ainsi, en cas d'arrivée simultanée de signaux, l'un d'eux est choisi arbitrairement. A noter cependant que les signaux pour communications entre services sont toujours traités les premiers (§ D.5.3).

Si plusieurs signaux sont disponibles au moment de l'entrée du processus dans un état, un seul signal est présenté au processus puis reconnu comme une entrée. Selon la sémantique du LDS, les autres signaux sont en fait retenus.

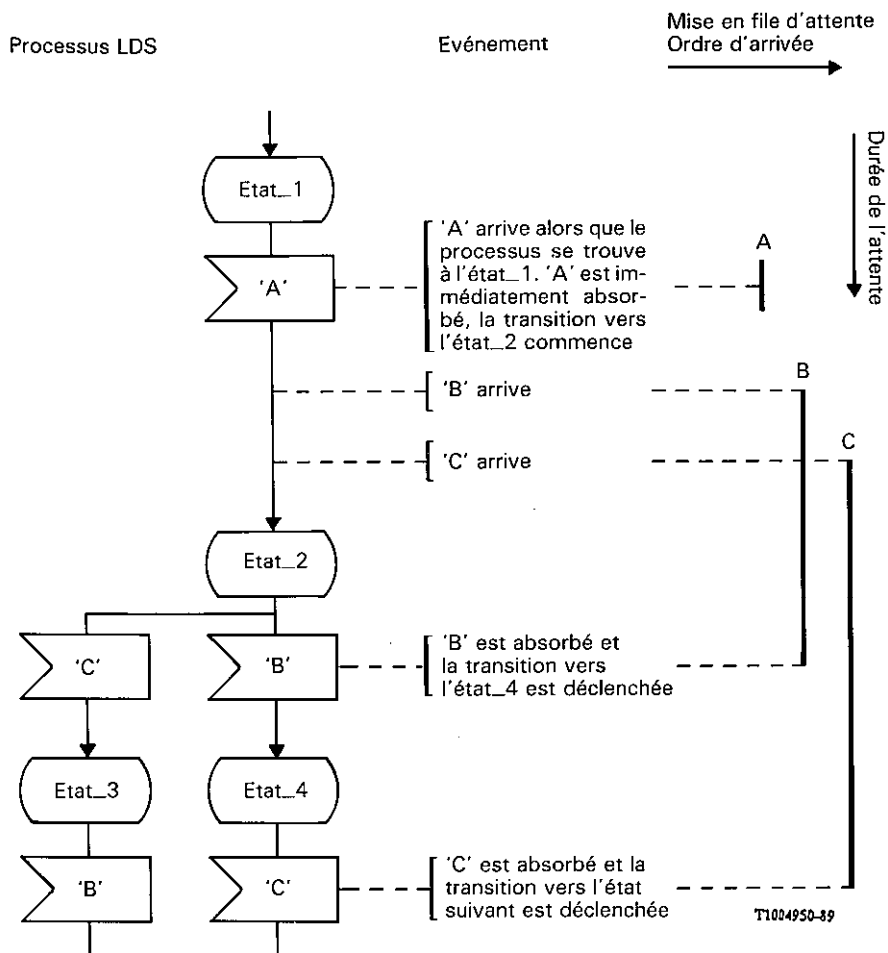


FIGURE D-3.8.23

Exemple de fonctionnement du mécanisme de mise implicite en file d'attente

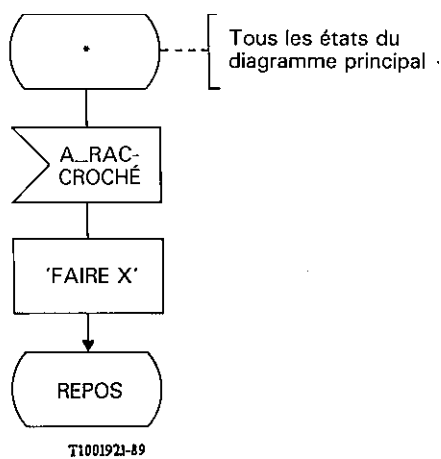


FIGURE D-3.8.24

Exemple de traitement des signaux pour qu'ils apparaissent dans plusieurs états

### D.3.8.3.5 Identification de l'émetteur

Chaque signal est porteur de la valeur d'instance du processus (PID) du processus émetteur. Lorsqu'un signal est absorbé, la valeur PID du processus émetteur peut être obtenue au moyen de l'expression SENDER. On trouvera dans la figure D-3.8.25 un exemple d'emploi de cette variable.

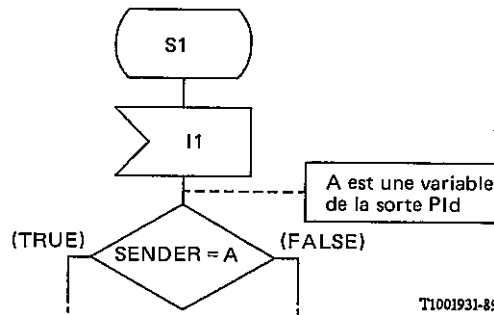


FIGURE D-3.8.25

#### Emploi de l'expression SENDER

### D.3.8.4 Mises en réserve

Le concept de mise en réserve permet de différer l'absorption d'un signal jusqu'à ce qu'un ou plusieurs signaux ultérieurs aient été absorbés. Comme l'indique le § D.3.8.3, les signaux sont absorbés dans l'ordre dans lequel ils se présentent, sauf en cas d'emploi du concept de mise en réserve.

L'on peut faire appel au concept de mise en réserve afin de simplifier les processus lorsque l'ordre relatif d'arrivée de certains signaux n'a pas d'importance et que leur ordre effectif d'arrivée est indéterminé.

Dans chaque état, chaque signal est traité comme suit:

- il est représenté comme un symbole d'entrée ou,
- il est représenté comme un symbole de mise en réserve ou,
- il est, par convention, couvert par une entrée implicite aboutissant à une transition nulle implicite.

Le fonctionnement du mécanisme implicite de mise en file d'attente décrit dans le § D.3.8.3 s'applique également au concept de mise en réserve. A leur arrivée, les signaux sont placés dans la file d'attente et lorsque le processus atteint un état donné, les signaux qui se trouvent dans la file d'attente sont examinés un à un dans l'ordre de leur arrivée. Un signal couvert par un symbole d'entrée explicite ou implicite est absorbé et la transition qui s'y rapporte est exécutée. Un signal représenté dans un symbole de mise en réserve n'est pas absorbé et reste dans la file d'attente dans la même position séquentielle; le signal suivant de la file d'attente est considéré. Aucune transition ne suit un symbole de mise en réserve.

En LDS/PR, la construction de mise en réserve est exprimée à l'aide du mot-clé SAVE suivi d'une liste d'identificateurs de signaux. On trouvera un exemple simple d'énoncés de MISE EN RÉSERVE dans la figure D-3.8.26.

```
...
STATE State_31;
  SAVE s;
  INPUT r;
  NEXTSTATE State_32;
STATE State_32;
  INPUT s;
...
```

FIGURE D-3.8.26

#### Exemple d'utilisation de la mise en réserve

En LDS/GR, le concept de mise en réserve est représenté à l'aide du symbole de mise en réserve contenant les identificateurs de signaux.

Comme dans le cas des entrées, une «notation avec astérisque» peut servir à représenter la mise en réserve de tous les signaux (valides pour ce processus) qui ne sont pas explicitement mentionnés dans cet état.

La figure D-3.8.27 représente un exemple d'un processus LDS qui comporte un symbole de mise en réserve. Il convient de noter que les signaux S et R sont consommés dans l'ordre R, S, c'est-à-dire dans l'ordre inverse de leur réception. Un symbole de mise en réserve unique peut servir à mettre un signal en réserve tant que le processus se trouve dans l'état dans lequel le symbole apparaît; ce signal est mis en réserve pour la durée de la transition vers le prochain état. Dans le prochain état, le signal sera absorbé par l'intermédiaire d'une entrée explicite ou implicite (voir la figure D-3.8.27) sauf dans les cas suivants: lorsque le symbole de mise en réserve comportant le nom du signal est répété ou lorsque dans la file d'attente implicite, il existe avant lui, un autre signal de sauvegarde disponible pour absorption (voir la figure D-3.8.28).

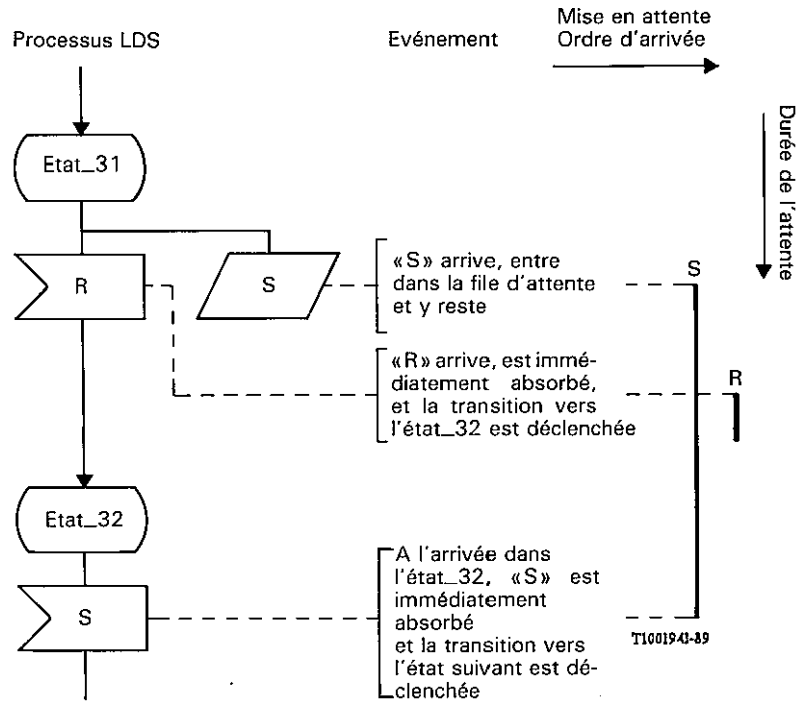


FIGURE D-3.8.27

Exemple de diagramme LDS avec symbole de mise en réserve montrant le fonctionnement d'un mécanisme implicite de mise en attente

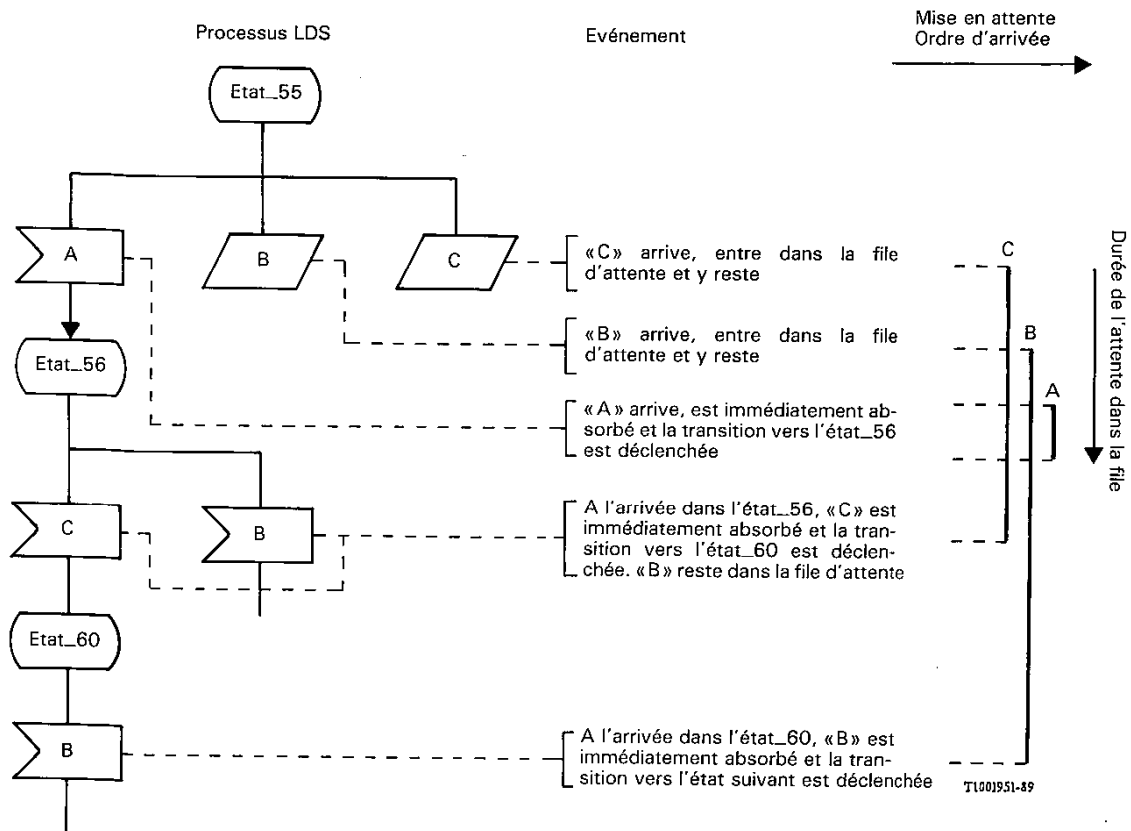


FIGURE D-3.8.28

Deuxième exemple de l'utilisation du symbole de mise en réserve

Un signal mis en réserve n'est mis à disposition d'un processus que par l'intermédiaire d'un symbole d'entrée correspondant (explicite ou implicite). Aucune question relative à un signal mis en réserve ne peut être posée dans une décision avant la reconnaissance de ce signal comme une entrée; de même les données qui lui sont associées ne sont pas disponibles.

Dans un état où plusieurs signaux doivent être mis en réserve, on peut attribuer un symbole de mise en réserve à chaque signal ou on peut les représenter tous à l'intérieur du même symbole de mise en réserve. Si plusieurs signaux doivent être mis en réserve, la sémantique du symbole de mise en réserve exige que l'ordre de leur arrivée soit préservé.

Un troisième exemple de l'utilisation de la notion de mise en réserve est donné dans la figure D-3.8.29 et la figure D-3.8.30 décrit le fonctionnement du mécanisme de formation de la file d'attente.

L'utilisation du symbole de mise en réserve peut simplifier les diagrammes. Ainsi, en mettant un signal en réserve, l'on peut éviter de le recevoir et de devoir mettre en mémoire ses données associées jusqu'à l'état suivant.

Bien que le symbole de mise en réserve puisse être utilisé à chaque niveau de description, il y aurait peut-être lieu, au niveau inférieur, de décrire le mécanisme effectif qui permet cette mise en réserve.

Sans un minimum de précautions dans l'emploi de la mise en réserve, la file d'attente des signaux mis en réserve peut augmenter considérablement ou garder des signaux en mémoire trop longtemps, de sorte qu'un signal ancien peut être absorbé lorsqu'un signal récent est demandé.

Le LDS ne prévoit pas de limite à la longueur de la file d'attente, ce qui peut poser des problèmes de mise en œuvre.

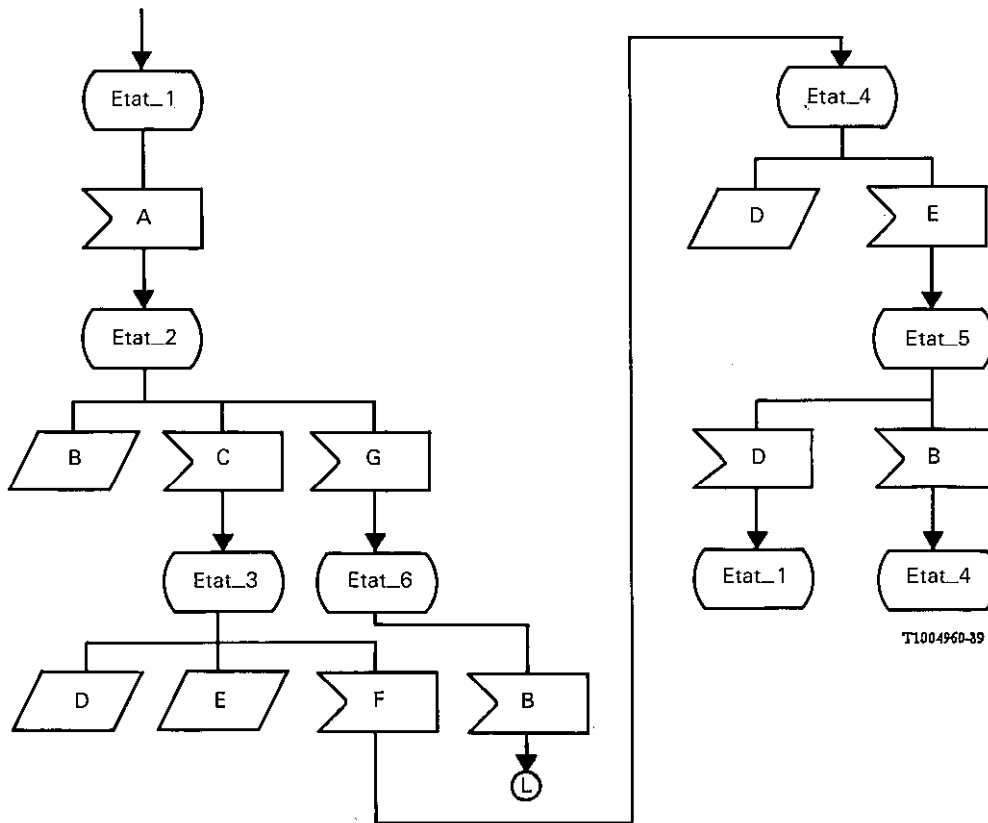


FIGURE D-3.8.29

Exemple de diagramme LDS plus complexe avec plusieurs entrées et mises en réserve

Etat actuel	Evénement	Formation de file d'attente
Etat_1	(Le processus entre dans l'état_1 avec les signaux «A», «B», «C», «D», «E» dans une file d'attente) Le premier signal de la file d'attente, «A», est absorbé (entrée explicite) et la transition vers l'état_2 est déclenchée	<p>Ordre d'arrivée →</p> <p>Durée d'attente dans la file</p>
Etat_2	Le premier signal de la file, «B», apparaît dans un symbole de mise en réserve et reste dans la file d'attente	
Etat_2	Le deuxième signal, «C», est absorbé (entrée explicite) et la transition vers l'état_3 est déclenchée	
Etat_3	Le premier signal de la file d'attente, «B», est absorbé (entrée implicite) et une transition nulle est déclenchée	
	«F» arrive et entre dans la file d'attente	
Etat_3	(En arrivant à nouveau dans l'état_3) le premier signal de la file, «D», apparaît dans un symbole de mise en réserve et reste dans la file d'attente	
Etat_3	Le deuxième signal, «E», apparaît dans un symbole de mise en réserve et reste dans la file d'attente	
Etat_3	Le troisième signal, «F», est absorbé (entrée explicite) et la transition vers l'état_4 est déclenchée	
Etat_4	Le premier signal de la file, «D», apparaît dans un symbole de mise en réserve et reste dans la file d'attente	
Etat_4	Le deuxième signal, «E», est absorbé (entrée explicite) et la transition vers l'état_5 est déclenchée	
Etat_5	Le premier (et seul) signal de la file, «D», est absorbé (entrée explicite) et la transition vers l'état_1 est déclenchée	

FIGURE D-3.8.30

Fonctionnement du mécanisme de mise en file d'attente

T1004970-39

### D.3.8.5 Condition de validation et signaux continus

Les conditions de validation permettent une réception conditionnelle de signaux fondée sur la condition de validation spécifiée. Le signal est reçu et la transition interprétée si la condition est vraie. En cas de condition fautive, le signal est mis en réserve et le processus ne change pas d'état jusqu'à ce qu'un autre signal arrive ou que la condition devienne vraie. L'exemple de la figure D-3.8.31 illustre ceci. Lorsque P1 passe à l'état S1, la condition (c'est-à-dire de savoir si `IMPORT (X,P2)` est égal à 10) est évaluée. Si elle est vraie, le signal B peut être reçu. Sinon, le signal B est mis en réserve. Dans cet exemple, A arrive et provoque une transition vers S2. Pendant la transition, X passe à la valeur 11 et P2 exporte sa nouvelle valeur; alors, la condition liée au signal B dans l'état S2 est vraie. Etant donné que B est le premier signal de la file d'attente, la transition qui suit est exécutée et le processus prend fin à l'état S3.

Certaines caractéristiques des conditions de validation sont importantes:

- 1) la condition de validation est testée lorsque le processus arrive à un état; il est alors continuellement contrôlé tandis que le processus reste dans cet état. Ainsi, si la valeur exportée de X avait passé de 9 à 11 puis à 12 pendant la transition qui faisait suite à la réception de A, le processus serait resté à S2;
- 2) les conditions de validation peuvent être fondées sur des variables locales et/ou toute construction de langage qui peut être comprise dans une expression (par exemple, `IMPORT (IMPORTATION)`, `VIEW (VUE)`, `NOW (MAINTENANT)`);
- 3) alors qu'il est possible d'employer plus d'une condition par état, l'emploi de plus d'une condition de validation pour le même signal n'est pas autorisé. Ainsi, la condition indiquée dans la figure D-3.8.32 n'est pas autorisée. Si un signal donné exige des conditions multiples, il est possible de les combiner en une expression booléenne ainsi que le montre la figure D-3.8.33.

On peut évaluer les conditions de validation plusieurs fois et dans un ordre quelconque, de sorte que l'utilisateur doit être vigilant si les expressions ont des effets secondaires réciproques (par exemple `IMPORT` et `SENDER` combinés).

Il faut noter en outre que le signal spécifié dans la condition de validation ne peut influencer la valeur booléenne de la condition car ses valeurs transportées ne sont pas affectées avant l'absorption du signal. A titre d'exemple, les énoncés:

```
INPUT x(A) PROVIDED (A=5);...
```

```
INPUT y PROVIDED(SENDER)=pid1);
```

prêtent à confusion car les valeurs de A et de `SENDER` dans les conditions correspondent à la situation telle qu'elle était avant l'absorption du signal.

Les signaux continus ont les mêmes propriétés fondamentales que la condition de validation, excepté le fait qu'ils ne sont accompagnés d'aucun signal. Ainsi, en l'absence de signaux dans la file d'attente susceptibles de provoquer une transition à leur entrée dans l'état, les signaux continus sont contrôlés; si l'un d'entre eux est vrai, la transition qui le suit est exécutée. L'exemple de la figure D-3.8.34 en donne l'illustration. A l'origine, le processus se trouve à l'état S1 et la valeur exportée de X est 9. En arrivant, le signal A provoque la transition vers l'état S2. Pendant la transition, X prend la valeur 11. Vu qu'aucun autre signal ne se trouve dans la file d'attente, la transition vers l'état S3 s'accomplit.

L'on trouvera ci-dessous certaines caractéristiques importantes des signaux continus:

- 1) de même que pour les conditions de validation, la valeur de la condition n'est contrôlée qu'à l'arrivée à un état;
- 2) les signaux continus multiples sont autorisés pour chaque état. Lorsque plusieurs signaux continus sont liés à un état, le signal continu ayant le rang de priorité le plus élevé (le numéro le plus bas) sera traité le premier. Deux signaux continus ne peuvent avoir le même rang de priorité. Le signal continu est toujours moins prioritaire que tout autre signal. Ceci est de nouveau dû au système sous-jacent du LDS: toutefois, la représentation à l'aide de modèles des signaux continus en LDS (emploi des signaux émis pendant l'exportation de variables), permet le recours à des priorités pour les signaux continus: ceci est d'ailleurs nécessaire afin d'éviter toute ambiguïté en cas de présence de plusieurs signaux continus. La figure D-3.8.35 en donne une illustration. Le processus commence à l'état S1 et ses variables locales X et Y ont respectivement les valeurs 10 et 11. Etant donné que les deux signaux continus sont vrais, celui qui a la plus haute priorité (rang de priorité le plus bas) est choisi et la transition vers l'état S2 s'accomplit. En S2, la condition de Y n'est plus vraie, et bien que la priorité du signal continu de X soit inférieure à celle de Y, la transition qui le suit est exécutée et le processus parvient à l'état S3;
- 3) lorsque la transition d'un signal continu a une suite, l'expression `SENDER (ÉMETTEUR)` retourne la même valeur de `SELF`.



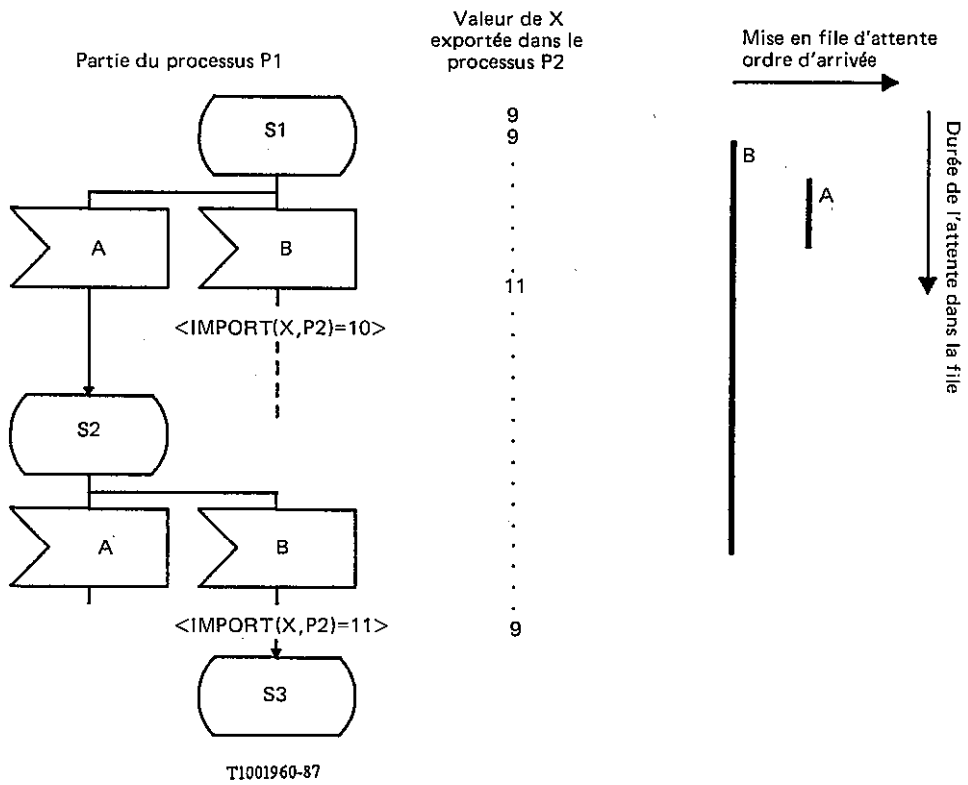


FIGURE D-3.8.31  
Condition de validation

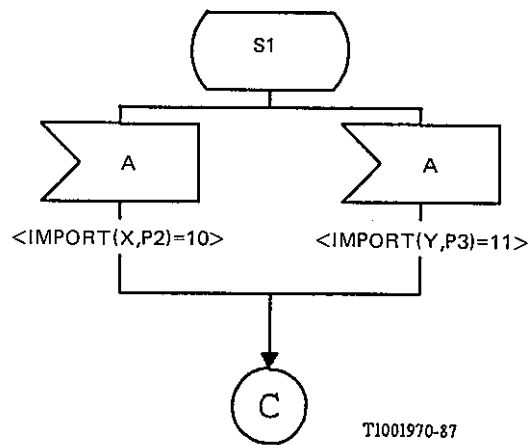


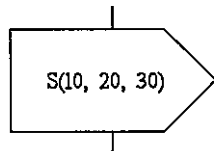
FIGURE D-3.8.32  
Condition de validation illicite





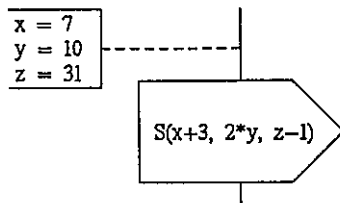
En LDS/GR, une sortie est représentée à l'aide d'un symbole de sortie contenant la spécification de signaux, de paramètres réels et, en option, de destination et/ou de construction VIA.

Chaque sortie doit être dirigée vers une instance de processus donnée. Etant donné qu'il est généralement impossible de connaître l'instance de processus de tout processus au moment de l'élaboration d'une spécification, la méthode normale pour diriger les signaux consiste à employer une variable ou une expression dans le mot-clé TO (VERS). On trouvera dans les figures D-3.8.38, D-3.8.39 et D-3.8.40 des exemples. Dans la figure D-3.8.38, le paramètre de processus «outo» prend la valeur d'une instance de processus lors de la création du processus. Le rôle de «outo» au sein du processus est d'assurer la liaison entre le processus en question et le processus auquel il est connecté. Il convient de veiller lors de la conception du système, à ce que le type de processus indiqué par «outo» puisse recevoir les signaux qui sont émis. Dans la figure D-3.8.39, l'expression prédéfinie SENDER sert à renvoyer un signal au processus qui a émis le signal reçu peu avant. Dans la figure D-3.8.40, le signal est envoyé vers le descendant le plus récent du processus.



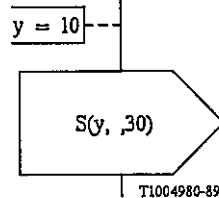
Remarque — Le signal S a trois valeurs, 10, 20 et 30 qui lui sont associées.

a)



Remarque — Pour l'interprétation de la sortie: x, y et z ont (dans le présent exemple) les valeurs 7, 10, 31 respectivement. La sortie émet les valeurs 10, 20, 30.

b)



Remarque — Pour l'interprétation de la sortie, y a (dans le présent exemple) la valeur 10. La sortie émet les valeurs 10 ainsi qu'une valeur indéfinie et 30.

c)

FIGURE D-3.8.37

Sortie avec valeurs associées

```

PROCESS X (0,5);
  FPAR (out_to PID);
  ...
  OUTPUT sig TO out_to;
  ...

```

FIGURE D-3.8.38

Signaux d'adressage utilisant des paramètres formels

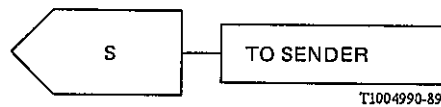


FIGURE D-3.8.39

Renvoi de signaux à l'ÉMETTEUR

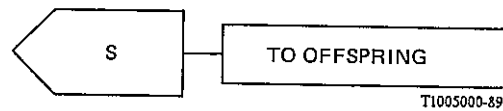


FIGURE D-3.8.40

Adressage d'un signal vers un descendant

### D.3.8.7 Tâche

Dans une transition, une tâche sert à représenter à l'aide d'un texte informel des opérations sur des variables ou une opération spéciale.

En LDS/PR, une tâche est représentée par le mot-clé TASK (TÂCHE) suivi d'une liste d'instructions ou de textes informels séparés par des virgules et se terminant par un point-virgule. Une instruction d'une tâche peut consister simplement en une affectation. Un texte informel consiste en une phrase délimitée par des apostrophes. On trouvera dans la figure D-3.8.41 des exemples de tâches valables en LDS/PR.

```

TASK a:=b;

TASK 'connecter l'abonné';

TASK c:=d+e;

TASK var1:=var2*var3,
  var4:=var5 MOD var6;

```

FIGURE D-3.8.41

Exemples de tâches

En LDS/GR, une tâche se compose d'un symbole de tâche contenant la liste des instructions ou des textes informels.

Les usagers du LDS peuvent parfois éprouver de la difficulté à décider si un aspect du système défini doit être représenté par une tâche ou un processus distinct. Considérons l'exemple du processus représenté dans la figure D-3.8.42: l'action «connecter-trajet-de-commutation» doit-elle être représentée par une tâche ou par un processus distinct? Si l'on n'a pas identifié un processus distinct de commande de trajet de commutation, il serait indiqué d'utiliser le symbole tâche [voir la figure D-3.8.42 a)]. Si on a identifié un processus distinct de commande de trajet de commutation, on doit utiliser les signaux qui communiquent avec le processus de commande [voir la figure D-3.8.42 b)].

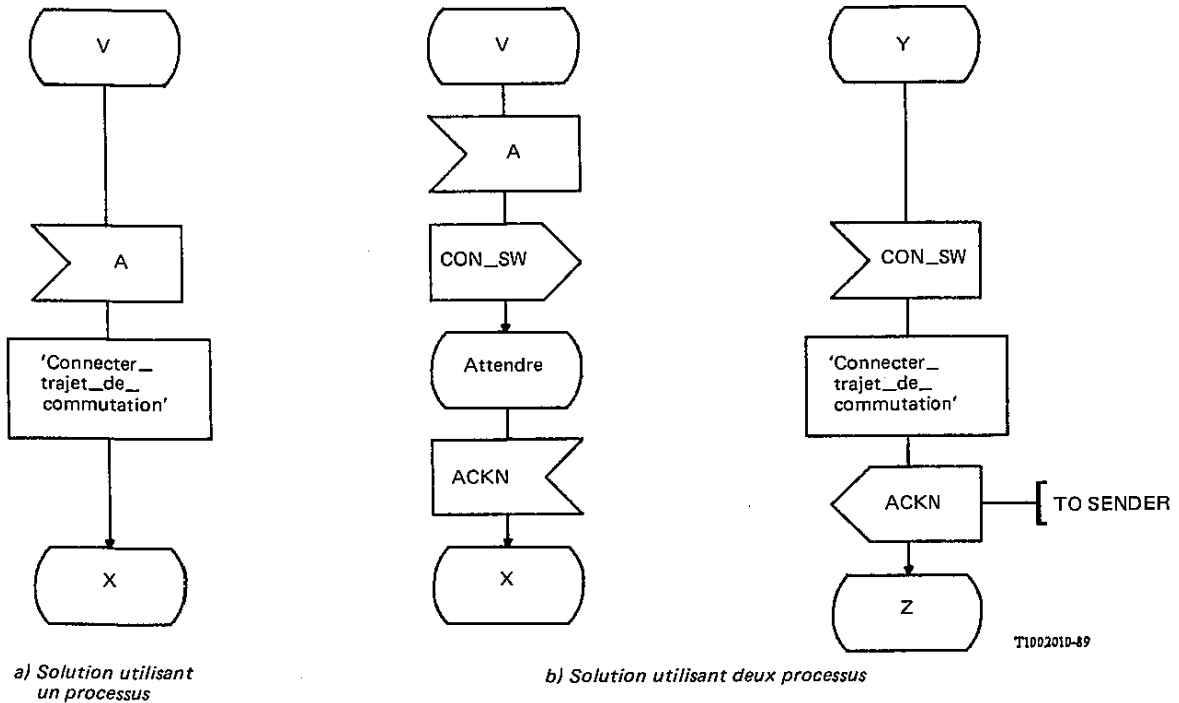


FIGURE D-3.8.42

**Deux solutions possibles pour «connecter-le-trajet-de-commutation»**

D.3.8.8 Décisions

Une décision est une action qui se produit au cours d'une transition et qui correspond à une question concernant la valeur d'une expression au moment de l'exécution de l'action; le processus a le choix entre deux ou plusieurs trajets, ce choix étant déterminé par la réponse consécutive à la décision. Les auteurs des diagrammes LDS doivent veiller à ce que les processus soient définis de manière qu'ils ne puissent tenter d'exécuter des décisions pour lesquelles des réponses (ou les données) ne sont pas disponibles; de telles décisions rendraient le diagramme tout à fait incorrect et entraîneraient une confusion considérable.

La question à laquelle correspond une décision peut être une expression ou un texte informel. Les réponses apportées par une décision sont représentées par une ou plusieurs valeurs possibles obtenues par l'évaluation de l'expression de la question ou par un ou plusieurs textes informels. Si la question ou l'une des réponses est informelle, toute la décision est informelle. Des réponses différentes sont séparées par des virgules. Les valeurs sont représentées par des expressions constantes, par des expressions constantes ayant un opérateur comme préfixe ou par des gammes dont les limites supérieures et inférieures sont des expressions constantes. Les valeurs de réponse doivent être du même type que l'expression contenue dans la question.

Il est possible d'indiquer explicitement certaines réponses et de grouper toutes les autres réponses possibles en employant le mot-clé ELSE (AUTRE).

En LDS/PR, la décision est représentée par le mot-clé DÉCISION suivi par la spécification de la question et la liste des réponses possibles, chacune étant associée à la transition correspondante. Les réponses sont indiquées entre parenthèses. L'ensemble des transitions sortantes est délimité par le mot-clé ENDDECISION (FIN DE DÉCISION) placé à la fin (voir la figure D-3.8.43).

```

DECISION question;
  (answer_a): ...
  (answer_b): ...
  (answer_c): ...
  ELSE: ...
ENDDÉCISION;

```

FIGURE D-3.8.43

**Schéma d'une décision**

On trouvera certains exemples de décisions dans la figure D-3.8.44.

```

...
DCL x INTEGER,a BOOLEAN;
...
DECISION x;
  (2): ...;
  (2+5): ...;
  (6+8,10+9): ...;
  (=3): ...;
  (20:30): ...;
  (>=100): ...;
  ELSE: ...;
ENDDÉCISION;
...
DECISION a;
  (TRUE): NEXTSTATE s1;
  (FALSE): NEXTSTATE s2;
ENDDÉCISION;
...
DECISION 'Catégorie d'abonné':
  ('International', 'National'): ...
  ('Local'): ...
ENDDÉCISION;
...

```

FIGURE D-3.8.44

**Exemples de décisions en LDS/PR**

Toutes les transitions se terminent par le mot-clé ENDDÉCISION (FIN DE DÉCISION). Les décisions qui ne se terminent pas par un énoncé terminal (c'est-à-dire jonction, état suivant, arrêt) continuent dans l'énoncé qui suit le mot ENDDÉCISION, comme indiqué dans les deux branches équivalentes de la figure D-3.8.45.

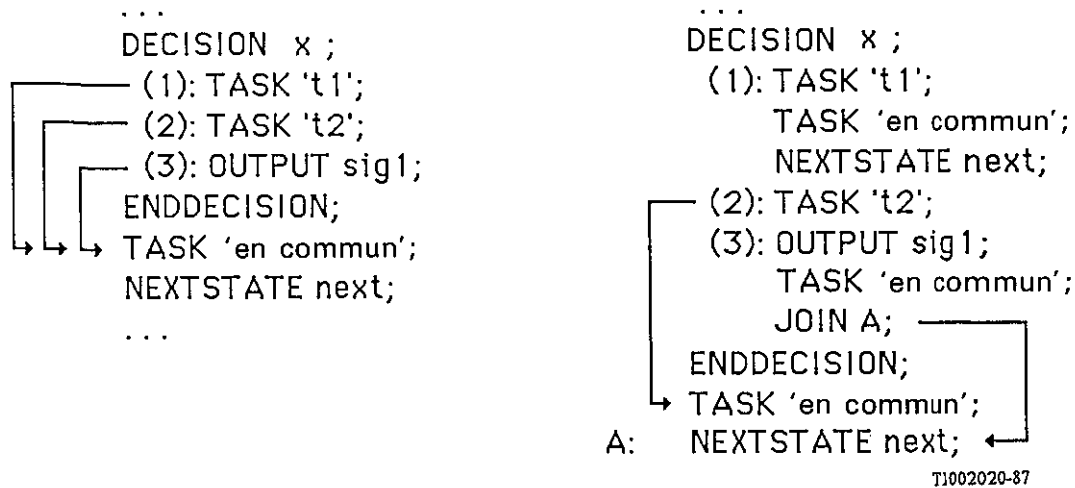


FIGURE D-3.8.45  
Branches équivalentes d'une décision

L'instruction de décision peut en outre servir à former la structure IF-THEN (SI-ALORS), la structure DO-WHILE (FAIRE-PENDANT) et la structure LOOP-UNTIL (BOUCLE-JUSQU'À) comme en programmation structurée.

En LDS/GR, une décision est représentée à l'aide d'un symbole de décision contenant le texte de la question. Le symbole doit avoir deux branches ou plus associées aux réponses correspondantes. Chaque réponse doit être placée à la droite ou au sommet de la branche correspondante ou au-dessus de la branche qui interrompt la ligne de liaison. En LDS/GR, les parenthèses servant à délimiter les réponses sont facultatives mais il est suggéré de les utiliser pour éviter tout malentendu.

On trouvera certains exemples de décisions en LDS/GR dans la figure D-3.8.46.

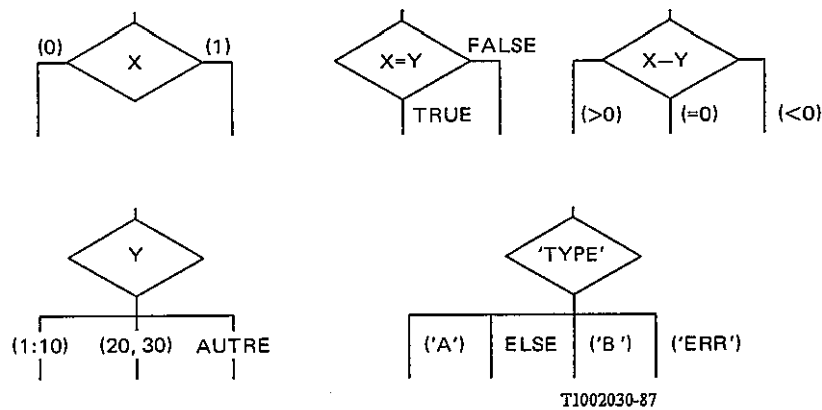


FIGURE D-3.8.46  
Exemples de décisions en LDS/GR

Si une réponse ramène à la décision de la même transition, il convient d'exécuter certaines actions qui influencent la question ayant trait à la décision. Toutefois, même si cette règle est appliquée, des boucles infinies peuvent apparaître, comme indiqué dans la figure D-3.8.47. Il faut donc toujours faire attention lorsque des réponses se réfèrent à une décision de la même transition.



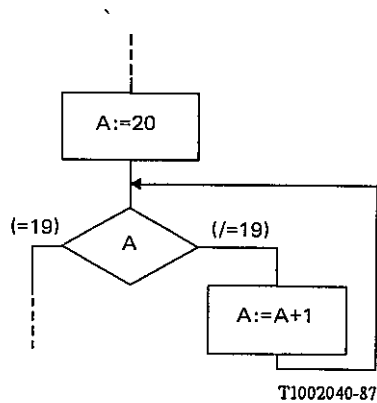


FIGURE D-3.8.47

**Exemple d'un emploi correct d'une décision qui crée une boucle infinie**

Des décisions peuvent être prises à l'aide de toute valeur existant dans le processus et notamment:

- des valeurs reçues par une entrée;
- des valeurs transmises en tant que paramètres effectifs au moment de la création du processus;
- des valeurs partagées.

L'expression qui figure dans la question peut comprendre des constantes de l'un quelconque des types de valeurs susmentionnés.

D.3.8.9 *Branchements et connecteurs*

Les branchements permettent de transférer la commande d'un point à un autre d'un corps de processus (ainsi qu'à l'intérieur d'un corps de procédure ou d'un corps de service).

En LDS/PR, les branchements sont équivalents à des énoncés «GO TO». Des étiquettes sont utilisées comme points d'introduction associés aux instructions, comme indiqué dans la figure D-3.8.48. A l'intérieur d'un corps de processus (ou d'un corps de procédure), il est impossible de transférer la commande (et par conséquent les étiquettes associées) au type instructions indiqué dans la figure D-3.8.49. Les étiquettes demeurent toujours localisées dans un processus; il est donc impossible de transférer la commande d'un processus à un autre à l'aide d'un branchement.

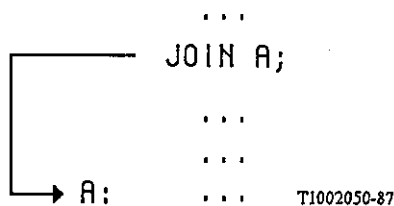


FIGURE D-3.8.48

**Etiquette**

STATE  
 ENDSTATE,  
 INPUT,  
 SAVE,  
 ENDDCISION

FIGURE D-3.8.49

**Points où les étiquettes sont interdites**

En LDS/GR, les branchements correspondent aux connecteurs (de sortie et d'entrée). Elles peuvent être utilisées pour subdiviser les programmes, en cas de manque de place, ou pour éviter le croisement de lignes de liaison qui enlèverait de leur clarté aux diagrammes. En outre, il est généralement préférable, lorsque l'on trace un diagramme en LDS, que la liaison se dirige du haut au bas de la page.

En GR, toute ligne de liaison peut être interrompue par une paire de connecteurs associés; on admet alors que la liaison se dirige du connecteur de sortie au connecteur d'entrée. Chaque symbole de connecteur contient un nom, associé aux connecteurs portant le même nom. Il existe un seul connecteur d'entrée pour chaque nom mais il peut y avoir un ou plusieurs connecteurs de sortie.

En GR, il est souhaitable que la page de référence se rapportant au connecteur d'entrée approprié soit spécifiée pour le connecteur de sortie et que la ou les références de pages relatives aux connecteurs de sortie appropriés devraient être données pour le connecteur d'entrée (voir l'exemple de la figure D-3.8.50).

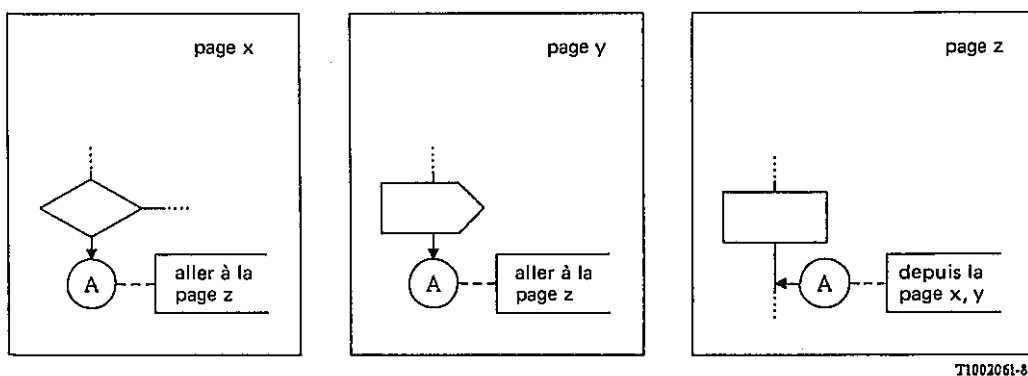


FIGURE D-3.8.50

**Références de page pour les connecteurs**

D.3.9 *Procédures*

En LDS, les procédures sont similaires aux procédures du CHILL et d'autres langages de programmation. Elles visent à:

- a) permettre de structurer un processus en plusieurs niveaux de précision;
- b) conserver la compacité des spécifications en permettant de représenter comme un seul élément un assemblage complexe d'éléments qui peuvent être considérés isolément;
- c) permettre de définir et d'utiliser à plusieurs reprises les assemblages d'éléments utilisés.

Une définition de procédure ne peut exister que dans une définition de processus, dans une définition de service ou une définition de procédure et, par conséquent, une procédure n'est visible que pour le processus ou la procédure dans lesquels elle est définie.

Une définition de procédure se compose des éléments suivants (dont certains sont facultatifs):

- nom de la procédure;
- paramètres formels de procédure: liste de noms de variables associées à leur sorte. Ces paramètres servent à transférer l'information de la procédure ou à partir de celle-ci au moment de l'appel. Les paramètres de procédure peuvent être passés par valeur (paramètre IN) ou par référence (paramètre IN/OUT). Si un paramètre est passé par valeur, la spécification du paramètre formel définit une variable locale à la procédure; s'il est passé par référence, la spécification définit un synonyme pour la variable;
- définitions de procédure: procédures qui peuvent être appelées tout comme la procédure proprement dite;
- définitions de données: spécification de types de données locales à la procédure;
- définitions de variables: variables locales à la procédure;
- corps de procédure: spécification du comportement effectif de la procédure pour ce qui est des états et des actions (comme pour le corps de processus).

On trouvera dans la figure D-3.9.1 un exemple partiel de définition de procédure en LDS/PR (les mots-clés du langage sont écrits en majuscules). A noter que les paramètres formels sans attribut explicite ont un attribut implicite IN (var5 dans la figure).

```
PROCEDURE prcd1;
  FPAR IN/OUT var1,var2 sort1,
      IN var3 sort2,
      IN/OUT var4 sort3, var5 sort4;
  PROCEDURE ...
  PROCEDURE ...
  ... data denifitions ...
  DCL ...
  ... procedure body ...
ENDPROCEDURE prcd1;
```

Paramètres formels de procédure  
Définitions de procédure  
Définitions de données  
Définitions de variables  
Corps de procédure

FIGURE D-3.9.1

**Exemple de définition partielle de procédure en LDS/PR**

D.3.9.1 *Corps de procédure*

Le corps de procédure présente de grandes similitudes avec le corps de processus; toutefois les différences sont les suivantes:

- la procédure termine son interprétation avec une indication «retour» au lieu d'une indication «arrêt». En LDS/PR, l'énoncé de retour est représenté par le mot-clé RETURN;
- en LDS/GR, le symbole de début de procédure est légèrement différent du symbole de début de processus.

Les symboles de début et de retour de procédure sont indiqués dans le résumé du LDS/GR.

Une procédure peut utiliser la construction avec branchements mais seulement pour se référer à une étiquette interne. Un branchement peut ne pas être utilisé ou être utilisé pour accéder à une procédure de l'extérieur ou quitter celle-ci.

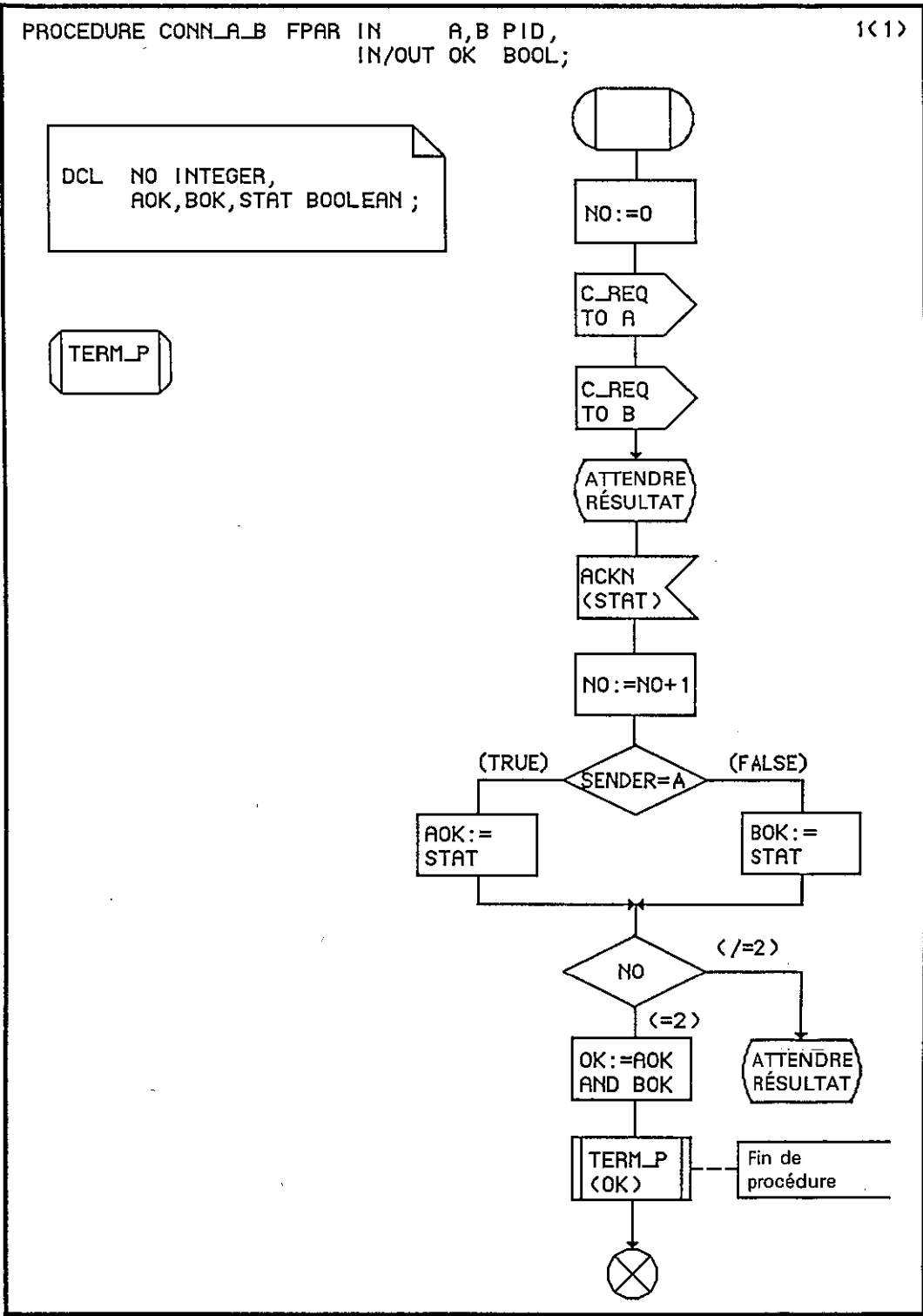
En LDS/GR, une définition de procédure est représentée par un diagramme de procédure très similaire au diagramme de processus. Le diagramme de procédure comprend les éléments suivants:

- un symbole de cadre facultatif: symbole de forme rectangulaire contenant tous les autres symboles;
- l'en-tête de procédure: le mot-clé PROCÉDURE suivi du nom de la procédure et de la spécification des paramètres formels de procédure. Généralement, l'en-tête de procédure est placé dans l'angle supérieur gauche du cadre ou, s'il n'y a pas de cadre, dans l'angle supérieur gauche du support sur lequel le diagramme est tracé;
- une numérotation de pages facultatives (à placer dans l'angle supérieur droit);

- des symboles de texte: dans le cas d'un diagramme de procédure, un symbole de texte peut être utilisé pour contenir la spécification de définition de paramètres formels, de données et de variables;
- références de procédure: symboles de procédure, contenant chacun un nom de procédure représentant une procédure locale définie séparément;
- des diagrammes de procédure: permettant de définir directement les procédures locales;
- la zone graphique de procédure: spécification du comportement de la procédure pour ce qui est du début, des états, des entrées, des sorties, des tâches... et des arcs orientés.

Dans la figure D-3.9.2, on trouvera un exemple de définition de procédure en LDS/GR. La procédure portant la référence «TERM\_P» dans l'exemple est locale à la procédure d'appel.

Comme indiqué dans le cas des diagrammes de procédure (§ D.3.8), si une seule page ne suffit pas à contenir un diagramme de procédure, celui-ci peut être représenté sur plusieurs pages, avec répétition du symbole de cadre à la suite de l'en-tête et du numéro de page.



T1002071-89

FIGURE D-3.9.2  
Exemple de diagramme de procédure

### D.3.9.2 Appel de procédure

Les appels de procédure peuvent se produire chaque fois qu'une tâche est autorisée dans un graphe de processus ou de procédure. En un sens, une procédure peut être interprétée comme une tâche, avec les exceptions suivantes:

- 1) une procédure peut contenir des états et, si tel est le cas, recevoir des signaux;
- 2) une procédure peut émettre des signaux. L'instance de processus d'origine est celle qui a appelé la procédure.

Lorsqu'une procédure est appelée, son environnement est créé et l'interprétation de la procédure commence. Elle se poursuit jusqu'à ce que l'on ait atteint l'indication RETURN (retour). Pendant l'interprétation de la procédure, tous les signaux adressés au processus sont soit mis en réserve implicitement soit traités explicitement par la procédure. La procédure n'a pas sa propre file d'attente mais utilise celle du processus qui l'a appelée.

En LDS/PR, un appel de procédure est représenté par le mot-clé CALL suivi de l'identificateur de procédure et de la liste de paramètres réels mise entre parenthèses. Si un paramètre n'est pas donné, il convient de l'indiquer par deux virgules consécutives. Dans ce cas, le paramètre formel correspondant a la valeur «indéfinie». A noter aussi que la déclaration de IN ou IN/OUT est faite dans la définition de procédure, de sorte qu'elle ne doit pas être répétée par l'énoncé d'appel. On trouvera certains exemples d'appel en LDS/PR dans la figure D-3.9.3.

```
CALL proced1;  
CALL proced2(var1,var2);  
CALL proced3 (5,a + b,Pid1);  
CALL proced4 (W,X,,Z);
```

FIGURE D-3.9.3

#### Exemples d'instructions d'appel en LDS/PR

En LDS/GR, un appel de procédure est représenté à l'aide d'un symbole d'appel de procédure contenant le nom de procédure et la liste des paramètres réels mise entre parenthèses. On trouvera un exemple d'appel en LDS/GR dans la figure D-3.9.2.

### D.3.10 Traitement des données

#### D.3.10.1 Déclarations 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 propriétaire peut modifier la valeur des variables.

Les variables déclarées dans la figure D-3.10.1 sont locales à chaque instance de processus P; elles ne peuvent être atteintes et modifiées que par chaque instance de processus P (chaque instance de processus peut avoir accès à sa propre copie de variables ou modifier celle-ci).

```

SYSTEM S;
...
BLOCK B;
...
PROCESS P(3,10);
  DCL
  A Integer,
  D Integer,
  ...
  ...
ENDPROCESS P;
ENDBLOCK B;
...
ENDSYSTEM S;

```

FIGURE D-3.10.1

**Exemple de déclaration de variables**

Une variable peut être initialisée immédiatement après la déclaration, comme indiqué dans la figure D-3.10.2.

```
DCL A Integer := 1;
```

FIGURE D-3.10.2

**Initialisation de variables**

Le LDS permet deux manières d'initialiser des variables. Il est possible de déclarer une valeur initiale pour toutes les variables d'une certaine sorte en utilisant l'instruction DEFAULT dans la définition de type de données (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 indiqué dans la figure D-3.10.2. S'il existe à la fois un énoncé DEFAULT et une initialisation lors de la déclaration, c'est cette dernière qui prévaut. Si une variable n'est pas initialisée, sa valeur initiale est considérée comme «indéfinie» dans le système.

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).

```

PROCESS P1;
  NEWTYPE S Struct
  I Integer;
  B Boolean;
ENDNEWTYPE;
DCL
A S := (. 1,True.);
...
ENDPROCESS P1;

```

FIGURE D-3.10.3

**Autre exemple d'initialisation de variables**

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).

```

PROCESS P;
  NEWTYPE Arr1 Array (Nat1,Integer)
  ENDNEWTYPE Arr1;
  SYNTYPE Nat1 Natural CONSTANTS 1:3
  ENDSYNTYPE Nat1;
  DCL
    A Arr1,
    I Natural;
  START;
  TASK I:=1;
  Lab1: DECISION I <= 3;
    (True): TASK A(I):=I;
           TASK I:=I+1;
           JOIN Lab1;
    (False);;
  ENDDECISION;
  ...
ENDPROCESS P;

```

FIGURE D-3.10.4

**Exemple plus complexe d'initialisation de variables**

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 œuvre des systèmes ainsi spécifiés et la présente section a pour but d'aider les utilisateurs à éviter ou à surmonter de tels problèmes. Décrire des systèmes du LDS qui ont mis en œuvre une valeur révélée/visualisée est moins difficile, car les problèmes auront été surmontés dans la mise en œuvre 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 processus 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 œuvre que R n'a pas libéré les données stockées qu'elle utilisait.

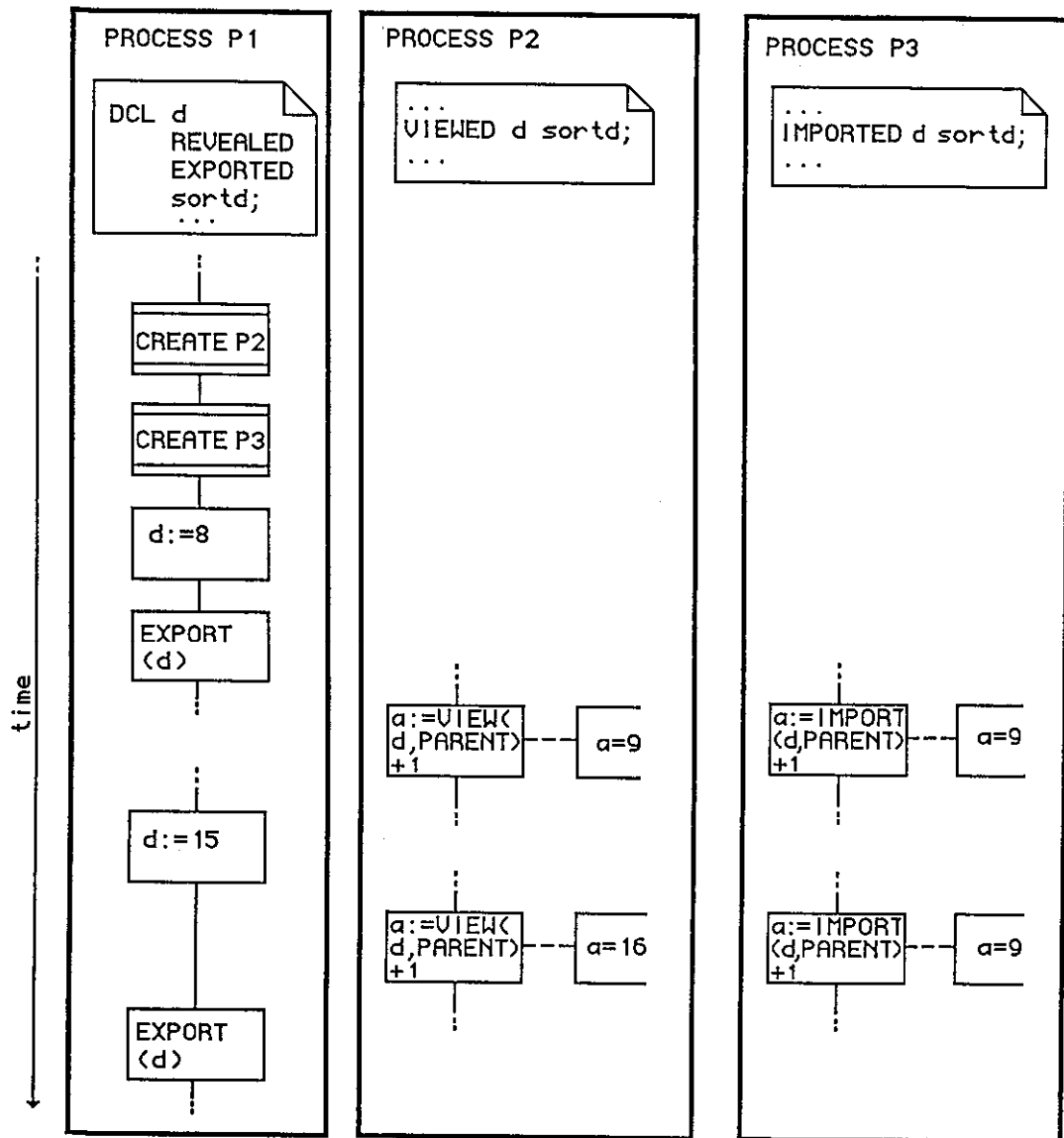
On trouvera dans la figure D-3.10.5 un exemple de variables révélées/visualisées.

### D.3.10.3 Valeurs exportées/importées

Un processus peut déclarer «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 déclarer la variable dans sa définition d'importation.

Lorsque le processus exportateur effectue une exportation, la valeur de la variable est copiée 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 différer de la valeur obtenue grâce à l'accès ordinaire par le propriétaire, même si ces deux opérations sont effectuées au même moment.

On trouvera un exemple dans la figure D-3.10.5.



T1002081-39

FIGURE D-3.10.5

Exemple de différences entre l'utilisation de variables révélées/visualisées et exportables

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, tâches, 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 être 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 être 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 être 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 être une cause de confusion. De plus, ce type de changement peut aboutir à des axiomes incohérents et rendre non valable la spécification du LDS.

```
NEWTYPE Newbool
  INHERITS Boolean
  OPERATORS ("+"="AND", "*"="OR", "=">")
ENDNEWTYPE Newbool;

...

A => B * C + D
```

FIGURE D-3.10.6

**Exemple de notation erronée dans une expression**

Tous les autres opérateurs définis par l'utilisateur sont des fonctions qui doivent être 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 être un autre opérande dans une expression.

### D.3.11 *Expression 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 être 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 spécifié et l'opération RESET annule la temporisation spécifiée. (A noter qu'une opération SET comporte implicitement une opération RESET pour toute temporisation provenant de ce temporisateur qui n'aurait pas expiré.)

La construction «set» contient l'expression temporelle du délai requis, le nom du temporisateur dont il s'agit et, en option, une liste d'expressions. Cette liste spécifie des valeurs qui seront comprises dans le signal du temporisateur de même 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.

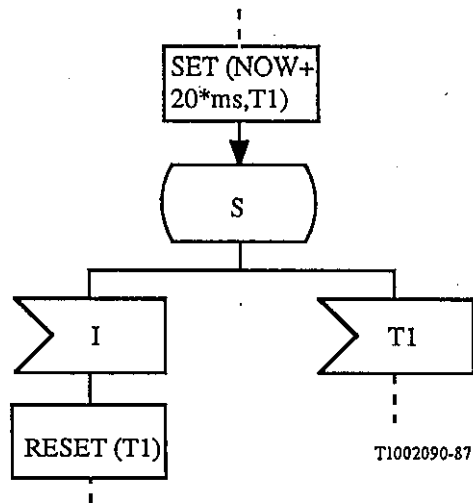


FIGURE D-3.11.1

Exemple d'utilisation de temporisateurs

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.

```

PROCESS p;
  TIMER T1 subscriber_id;
  ...
  DCL Sa subscriber_id;
  ...
  SYNONYM
    Sec Duration = 1000.0,
    Min Duration = 60000.0,
  ...
  START;
  ...
  SET (NOW + 20*Min + 30*Sec , T1 (Sa) );
  ...
  RESET (T1 (Sa));
  ...
ENDPROCESS p;

```

FIGURE D-3.11.2

Synonymes pour la représentation de durées

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 évité.

### 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 être 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.

```

SYSTEM s;
...
BLOCK b;
  SIGNAL x;
  ...
  PROCESS p;
  ...
  SIGNAL x;
  ...
  STATE wait;
  INPUT SYSTEM s/BLOCK b x;
  ...
  INPUT PROCESS p x;
  ...
  ENDSTATE wait;
  ...
  ENDPROCESS p;
  ENDBLOCK b;
ENDSYSTEM s;

```

FIGURE D-3.12.1  
Exemple d'utilisation de qualificatifs

### 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.

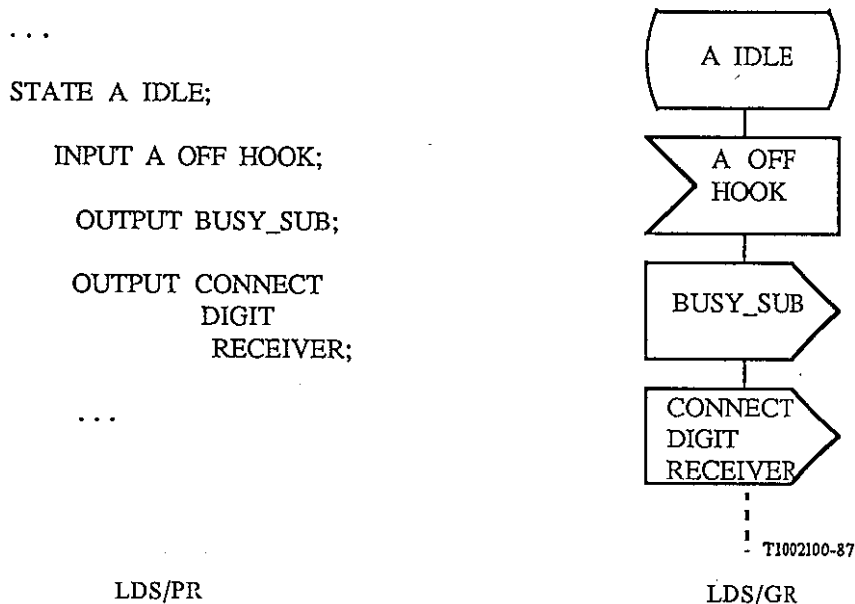


FIGURE D-3.13.1  
Exemples de noms composés de plusieurs mots

Chaque fois que l'emploi de plusieurs mots dans un nom donne lieu à une ambiguïté (voir les exemples de la figure D.3.13.2), les délimiteurs entre deux mots doivent être remplacés par un caractère de soulignement («\_»). La chaîne de caractères obtenue par la transformation de délimiteurs en soulignements désigne toujours le même nom; ainsi,

par exemple, BUSY SUB désigne le même nom que BUSY\_SUB. On trouvera dans la figure D-3.13.3 certains exemples d'emploi sans ambiguïté de noms.

- a) A := BLOCK B1 B / PROCESS P1 P C;
- b) DCL A B C D;
- c) NEWTYPE X Y Z (PAR) ENDNEWTYPE;

FIGURE D-3.13.2

**Exemples d'emploi ambigu de noms**

- a) A := BLOCK B1\_B/PROCESS P1\_P C;
- b) A := (BLOCK B1 B)/(PROCESS P1\_P C);
- c) DCL A B\_C\_D;
- d) DCL A\_B C\_D;
- e) DCL A\_B\_C D;
- f) NEWTYPE X Y\_Z(PAR) ENDNEWTYPE;
- g) NEWTYPE X\_Y Z(PAR) ENDNEWTYPE;

FIGURE D-3.13.3

**Exemples d'emploi sans ambiguïté de noms**

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.

- a) CONNECT DIGIT RECEIVER
- b) CONNECT\_DIGIT\_RECEIVER
- c) CONNECT DIGIT\_RECEIVER
- d) CONNECT  
DIGIT  
RECEIVER
- e) CONNECT DI\_  
GIT RECEIVER
- f) CONNECT DI\_ GIT RECEIVER
- g) CONNECT\_  
\_DIGIT\_  
\_RECEIVER

FIGURE D-3.13.4

**Exemples de désignations différentes pour le même nom**

## 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. Considéré 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).

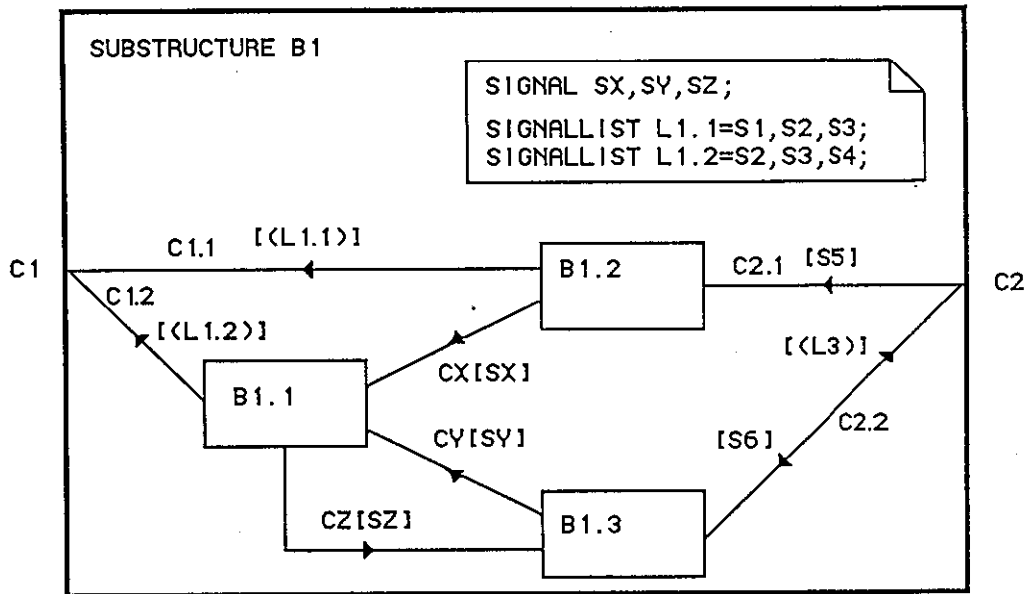
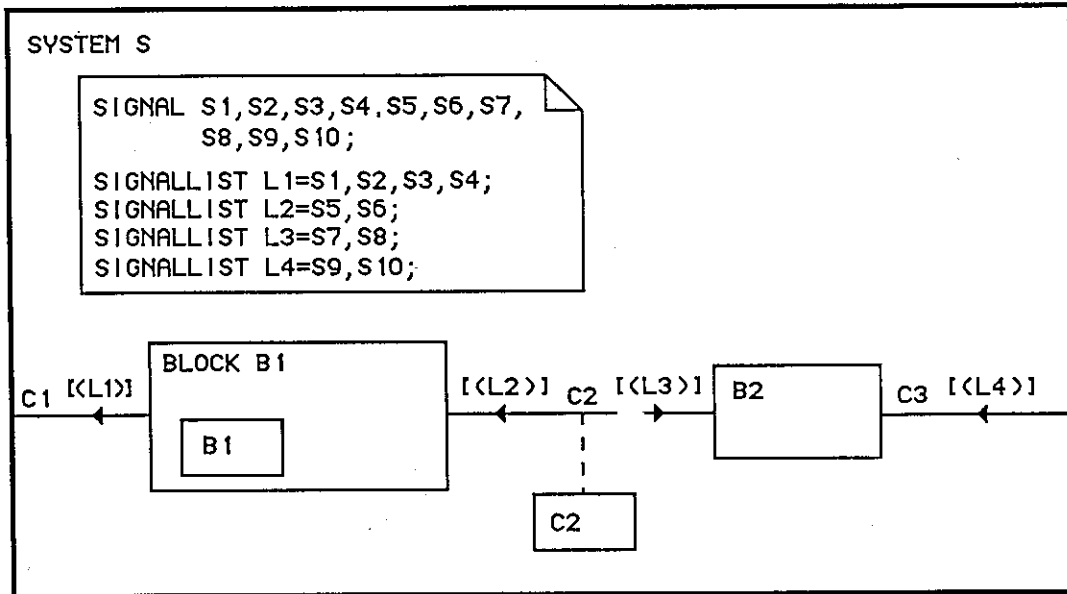
Les critères qui seront effectivement adoptés dépendront de plusieurs facteurs et notamment du degré de précision requis.

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'utilisateur 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 traits 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.



T1002110-87

FIGURE D-4.3.1  
Subdivision de blocs



```

SUBSTRUCTURE B1;

  SIGNALLIST L1.1 = S1,S2,S3;
  SIGNALLIST L1.2 = S2,S3,S4;

  SIGNAL SX,SY,SZ;

  CHANNEL C1.1
    FROM B1.2 TO ENV WITH (L1.1);
  ENDCHANNEL C1.1;
  CHANNEL C1.2
    FROM B1.1 TO ENV WITH (L1.2);
  ENDCHANNEL C1.2;
  CHANNEL C2.1
    FROM ENV TO B1.2 WITH S5;
  ENDCHANNEL C2.1;
  CHANNEL C2.2
    FROM ENV TO B1.3 WITH S6;
    FROM B1.3 TO ENV WITH (L3);
  ENDCHANNEL C2.2;
  CHANNEL CX
    FROM B1.2 TO B1.1 WITH SX;
  ENDCHANNEL CX;
  CHANNEL CY
    FROM B1.3 TO B1.1 WITH SY;
  ENDCHANNEL CY;
  CHANNEL CZ
    FROM B1.1 TO B1.3 WITH SZ;
  ENDCHANNEL CZ;

  CONNECT C1 AND C1.1,C1.2;
  CONNECT C2 AND C2.1,C2.2;

  BLOCK B1.1 REFERENCED;
  BLOCK B1.2 REFERENCED;
  BLOCK B1.3 REFERENCED;

ENDSUBSTRUCTURE B1;

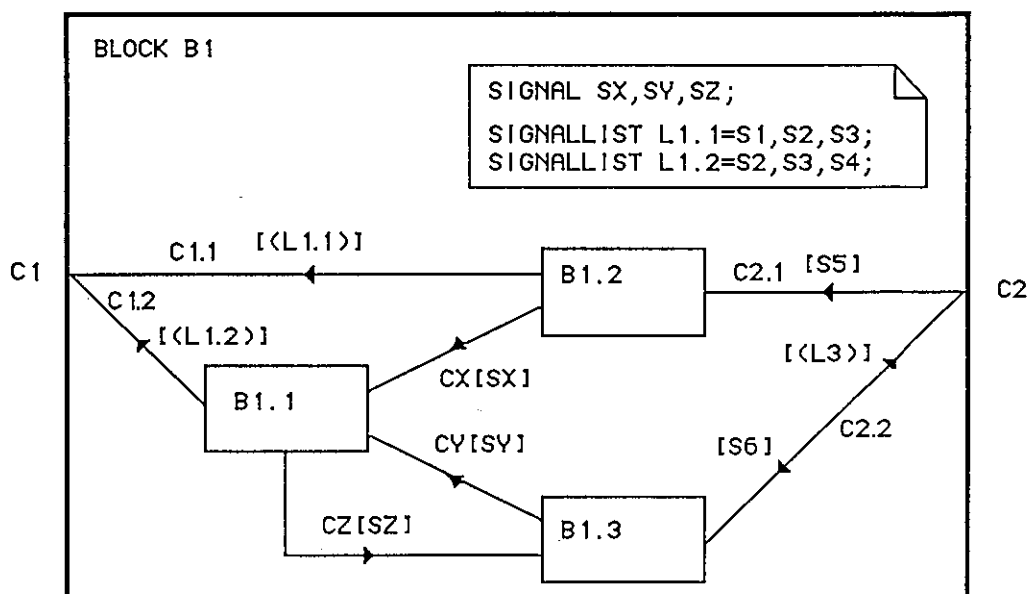
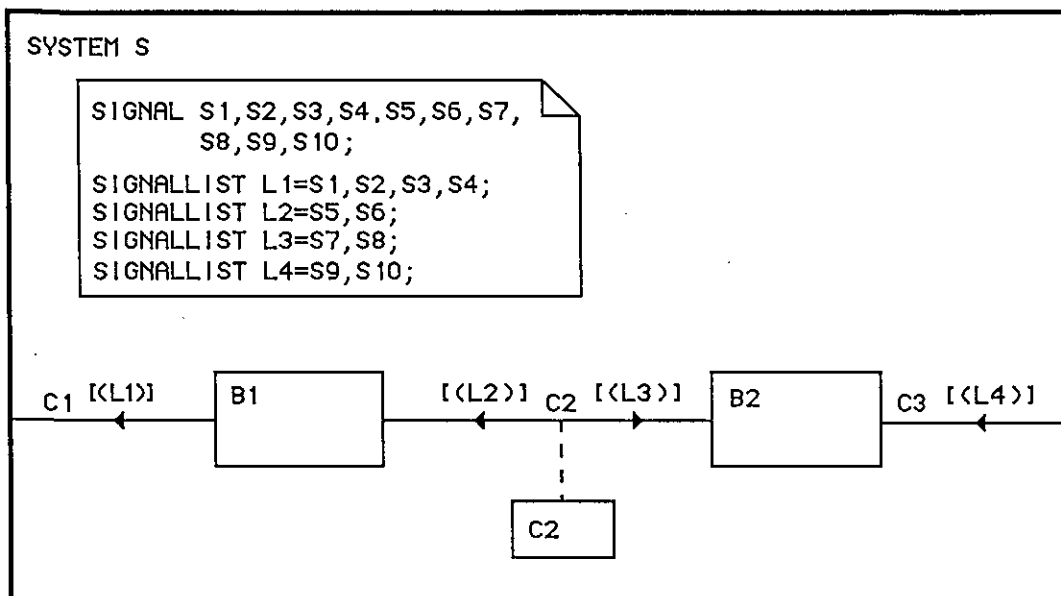
```

FIGURE D-4.3.2

**LDS/PR correspondant à l'exemple de la figure D-4.3.1**

La spécification des processus et celle d'une sous-structure de bloc peuvent coexister à l'intérieur d'une définition de bloc. Dans ce cas, les processus du bloc représentent le comportement de ce bloc pour un niveau de précision donné; d'autres processus internes des définitions des sous-blocs représenteront d'une manière plus détaillée le même comportement. On trouvera au § D.4.7 (figure D-4.7.1) un exemple de bloc décrit tant en fonction de processus que de sous-structures.

S'il n'existe pas de processus à l'intérieur d'un bloc mais seulement une sous-structure de bloc et que celle-ci ne fait pas l'objet d'un repérage, il existe en LDS/GR une notation abrégée permettant de simplifier le diagramme. Cette notation permet d'emboîter des blocs en considérant le cadre du bloc extérieur comme impliquant le cadre de sous-structure. Grâce à cette notation, l'exemple de la figure D-4.3.1 peut être tracé comme dans la figure D-4.3.3.



T1002120-87

FIGURE D-4.3.3

Notation abrégée en LDS/GR pour la subdivision de blocs

Chaque bloc provenant de la subdivision d'un bloc peut être lui-même subdivisé, ce qui donne une structure arborescente hiérarchique de blocs et de leurs sous-blocs. Un diagramme auxiliaire appelé «diagramme d'arbre de bloc» représentant cette structure générale est décrit au § D.4.4.

A moins qu'il ne s'agisse de l'affinage de signaux, les règles suivantes doivent être observées dans le processus de subdivision:

- 1) les listes de signaux des sous-canaux reliés à un canal d'entrée 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'entrée tous les signaux de L2. En outre, aucun des signaux transportés par C2.1 ne peut apparaître sur le trajet d'entrée de C2.2;
- 2) les listes de signaux des sous-canaux (par exemple C1.1 et C1.2) reliés 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 mêmes identificateurs de signaux;

- 3) si le bloc original contient des processus, il existe deux options. Tout d'abord, une copie de chaque processus peut être redéfinie dans l'un ou l'autre des nouveaux sous-blocs. Deuxièmement, de nouveaux processus peuvent être définis dans les sous-blocs de telle manière que l'interface reste sans changement;
- 4) des définitions de données du bloc ascendant se trouvent dans ses sous-blocs, de sorte que chacun de ceux-ci peut utiliser un type de données défini dans le bloc ascendant sans avoir à le redéfinir;
- 5) si un type de données défini dans le bloc ascendant est redéfini avec le même nom dans un sous-bloc, la nouvelle définition s'applique au sous-bloc définissant tandis que l'ancienne définition reste valable pour les autres sous-blocs. Il faut éviter une redéfinition servant uniquement à caractériser un sous-bloc car un lecteur peut la négliger, supposant que l'ancienne définition est valable. Dans certains cas cependant, il convient de procéder à une telle redéfinition, notamment lorsqu'il s'agit d'un affinage du comportement. Il faut veiller à souligner cette redéfinition par une annotation appropriée.

#### D.4.4 Diagramme d'arbre de blocs

Le diagramme d'arbre de blocs représente la structure d'un système en fonction de ces blocs et sous-blocs. Ce diagramme vise à donner au lecteur un aperçu de la structure totale d'un sous-système.

Le diagramme est un arbre hiérarchique de symboles de blocs et de lignes de subdivision comme indiqué dans la figure D-4.4.1.

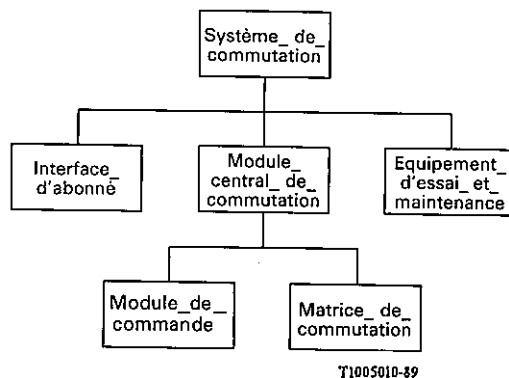


FIGURE D-4.4.1

Exemple de diagramme d'arbre de blocs

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.

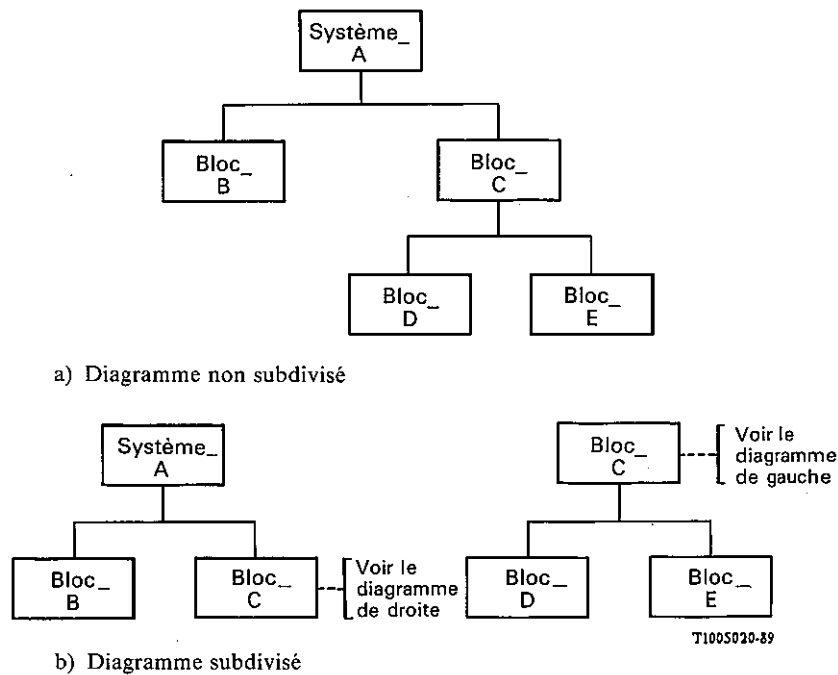


FIGURE D-4.4.2

**Exemple de diagrammes d'arbre de blocs partiels**

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.

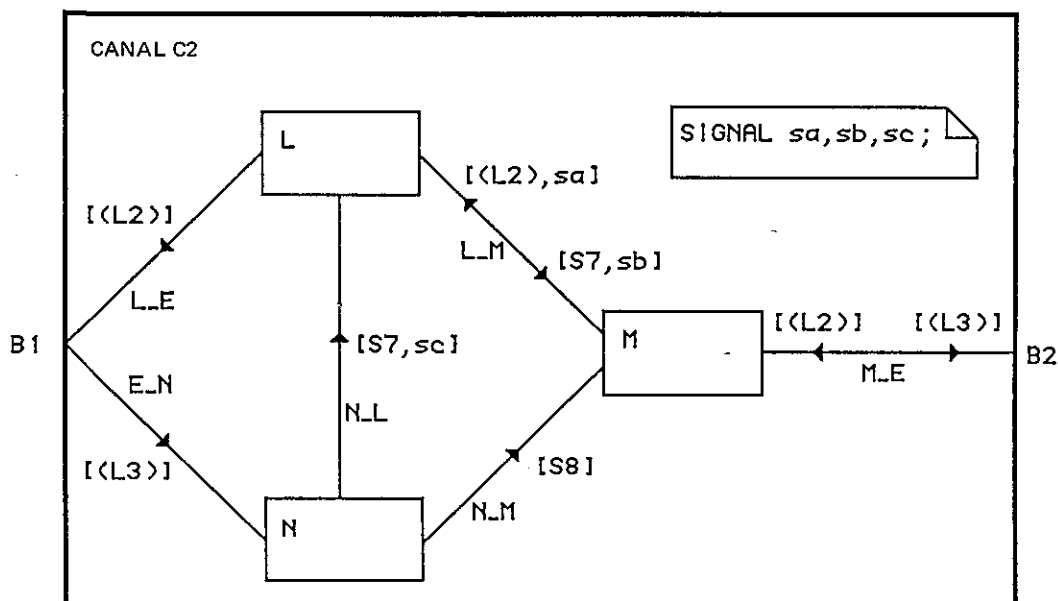
**D.4.5 Subdivision des canaux**

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 où apparaissent les canaux subdivisés, il faut faire référence au diagramme de sous-structure de canaux décrivant la subdivision.



T1002130-88

FIGURE D-4.5.1

Exemple de diagramme de sous-structure de canaux

En LDS/PR, la forme est semblable à celle de la définition de sous-structure de blocs; la seule différence est que, dans l'instruction CONNECT, les sous-canaux d'extrémité sont raccordés à des blocs extérieurs (B1 et B2 dans la figure D-4.5.2) et non à des canaux extérieurs.

L'import/export des valeurs est autorisé entre un bloc et une sous-structure de canaux. Cela permet une représentation directe du modèle OSI dans laquelle les communications de couches homologues sont modélisées par l'échange de signaux et les communications de couches contiguës par des valeurs partagées.

D.4.6 Représentation du système en cas de subdivision

Lorsqu'un système est représenté sous la forme d'un ensemble de blocs interconnectés par des canaux et que l'on exprime le comportement de chaque bloc au moyen d'un ou de plusieurs processus, nous nous trouvons en présence d'une représentation sur un seul niveau. Ceci revient à dire que nous pouvons voir tous les éléments de la représentation au même niveau. Nous introduisons, avec la subdivision une relation hiérarchique entre les différents documents. Le document qui en résulte représente la structure du système lorsque le système se compose de n blocs.

Un autre document peut présenter le système composé d'un ensemble différent de blocs, dont certains proviennent des blocs du document précédent (certains blocs du document précédent ont été remplacés par les sous-blocs résultant de la subdivision des blocs). Ce nouveau document doit être rapporté au précédent document.

Mais pour obtenir une représentation complète du système, il ne suffit pas d'établir des rapports entre les documents, il faut également les organiser de telle sorte qu'il soit possible d'accéder à la représentation du système par niveaux, en commençant par une vue d'ensemble générale pour passer ensuite à des représentations de plus en plus détaillées. Ceci exige le regroupement des différents documents afin de représenter le système à différents niveaux.

On ne devrait pas trouver les mêmes éléments à tous les niveaux. La représentation du système au premier niveau peut se composer de représentations des blocs et des canaux, à l'exclusion des processus qui décrivent le comportement de chaque bloc. A un niveau inférieur, 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.

```

SUBSTRUCTURE C2;

SIGNAL sa,sb,sc;

CHANNEL L_E
  FROM L TO ENV WITH (L2);
ENDCHANNEL L_E;

CHANNEL E_N
  FROM ENV TO N WITH (L3);
ENDCHANNEL E_N;

CHANNEL M_E
  FROM M TO ENV WITH (L3);
  FROM ENV TO M WITH (L2);
ENDCHANNEL M_E;

CHANNEL L_M
  FROM L TO M WITH S7,sb;
  FROM M TO L WITH (L2),sa;
ENDCHANNEL L_M;

CHANNEL N_M
  FROM N TO M WITH S8;
ENDCHANNEL N_M;

CHANNEL N_L
  FROM N TO L WITH S7,sc;
ENDCHANNEL N_L;

CONNECT B1 AND L_E,E_N;
CONNECT B2 AND M_E;

BLOCK L REFERENCED;
BLOCK M REFERENCED;
BLOCK N REFERENCED;

ENDSUBSTRUCTURE C2;

```

FIGURE D-4.5.2

**LDS/PR correspondant à l'exemple de la figure D-4.5.1**

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

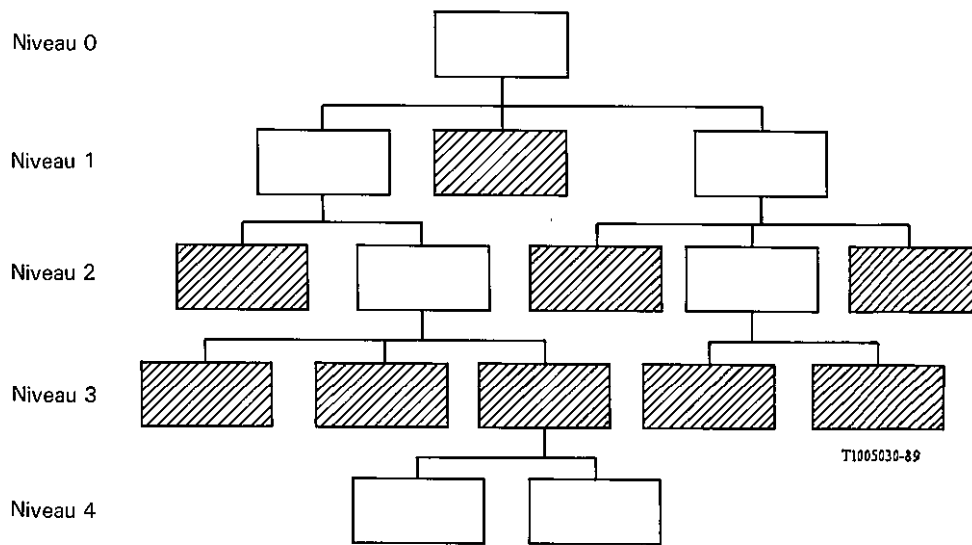
En premier lieu, l'arborescence a toujours une et seulement une racine, qui est le système. Il en est ainsi de même lorsque le système représenté se compose de plusieurs blocs depuis le début. Dans l'arborescence, ces blocs sont représentés au niveau 1. Le nom du système peut suffire pour constituer 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 système est représenté à plusieurs niveaux d'abstraction, nous pouvons représenter le système au niveau de notre choix. Le choix d'un niveau donné exige que les blocs soient examinés à ce même niveau, que leurs processus associés et tous les blocs feuille soient examinés à des niveaux supérieurs avec leurs processus associés (voir la figure D-4.6.1).

Il est souvent pratique de choisir différents niveaux à différentes fins, par exemple un niveau «d'aperçu» pour la présentation et un niveau plus détaillé pour la mise en œuvre.



*Remarque* – Le niveau de représentation 3 est indiqué au moyen de cases hachurées.

FIGURE D-4.6.1  
Représentation de niveau 3

La représentation d'un système à un certain niveau peut être incomplète dans la mesure où certains des blocs de ce niveau n'ont pas de processus associés.

On parle de niveaux de représentation et de sélection d'un certain niveau de représentation pour les raisons suivantes:

- il se peut que notre représentation ait atteint un certain «niveau» de précision, représenté par ce niveau de représentation (dans ce cas, les blocs de ce niveau sont des blocs feuille; ils ne sont pas complets car ils demandent un supplément de travail!);
- nous voulons considérer la représentation du système à un certain niveau de précision; nous choisissons donc le niveau de représentation qui correspond au mieux aux abstractions que nous cherchons. On notera que dans certains cas un niveau donné de représentation peut être constitué de documents à différents niveaux d'abstraction. La représentation d'une partie du système au niveau 2 peut être très détaillée, tandis que celle d'une autre partie au niveau 4 peut demeurer abstraite. Cela signifie qu'en choisissant une représentation au niveau 3, l'on peut obtenir une représentation très détaillée de certaines parties et une simple vue d'ensemble d'autres parties;
- la méthodologie de représentation et conception peut permettre à chaque niveau d'avoir une signification précise. Ainsi le niveau 1 correspond à la spécification, le niveau 2 fournit la structure générale du système, le niveau 3 la structure des modules («racks», fonctions de logiciel), le niveau 4 donne la structure détaillée (planches imprimées, procédures, modules de logiciel). Dans ce cas, le lecteur choisit un niveau donné en fonction de son propre besoin, et l'on notera que la méthode permet d'éviter les différences entre les niveaux de précision des parties qui constituent un niveau donné de représentation.

#### 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 spécification de 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.



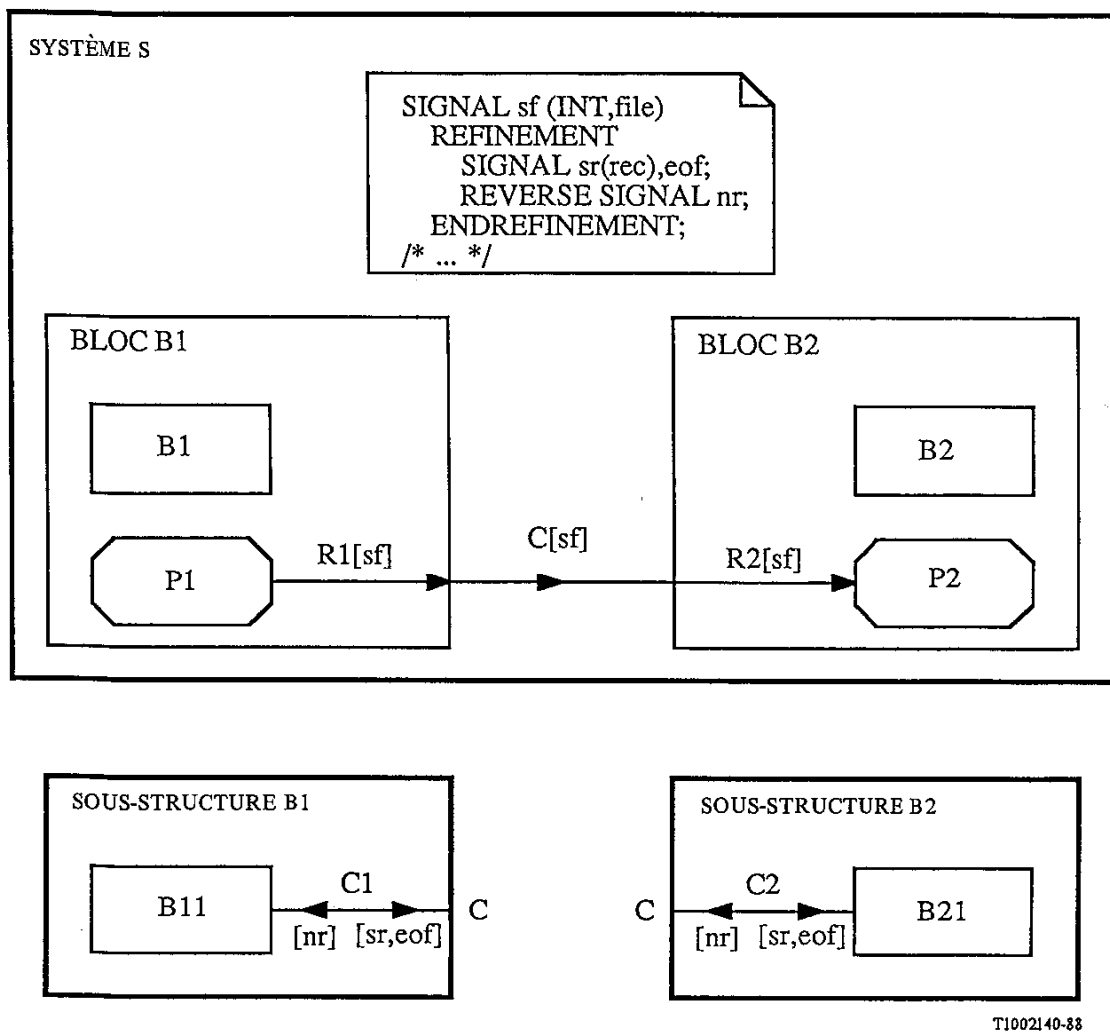


FIGURE D-4.7.1

Exemple d'affinage

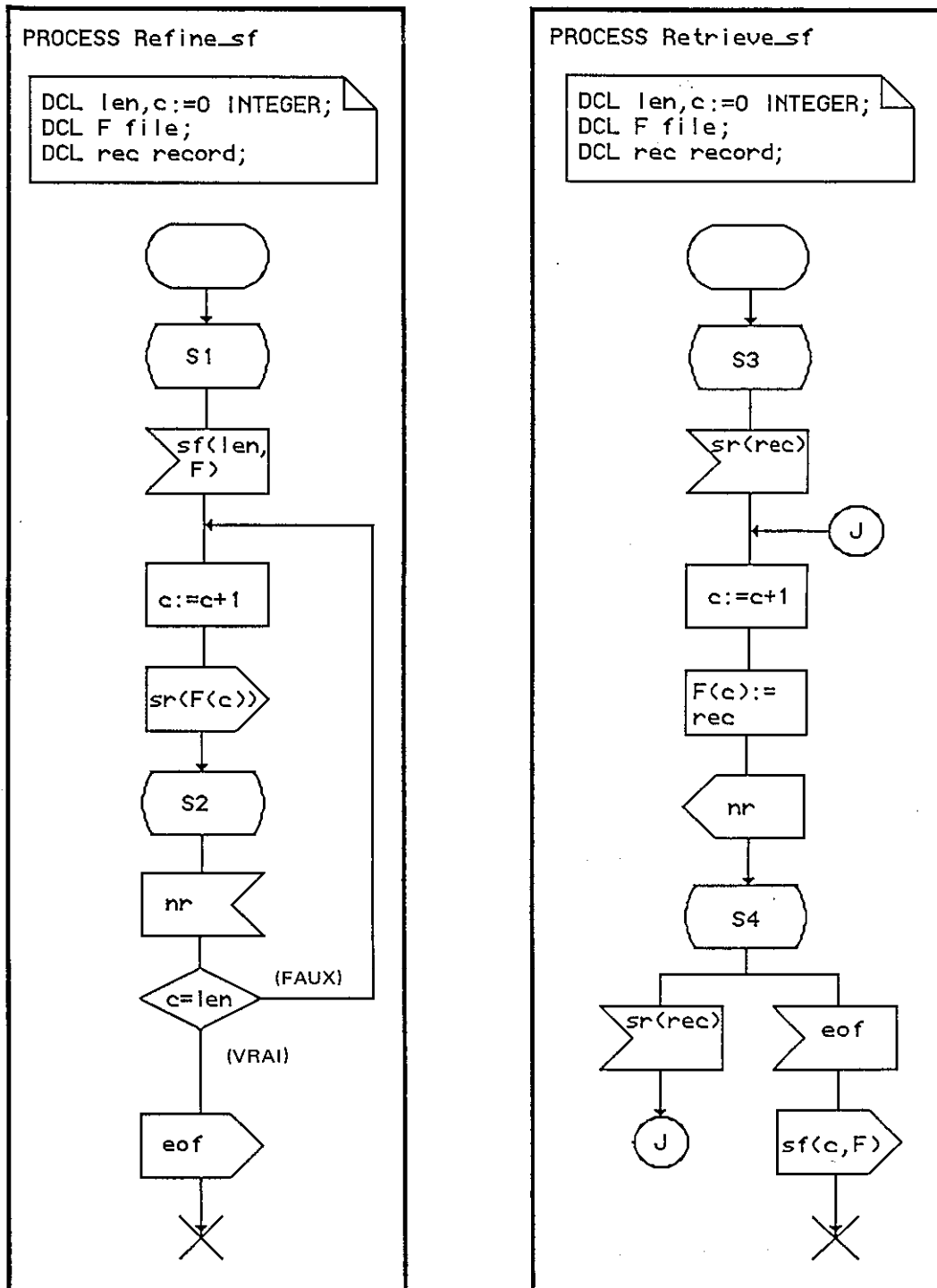
#### D.4.7.1 Sous-ensemble d'affinage cohérent

Lorsque l'affinage est appliqué à une spécification de système, la notion de sous-ensemble de subdivision cohérent est limitée de manière à éviter des communications avec différents signaux entre différents niveaux d'affinage. Dans ce cas, on dit que la définition de système contient plusieurs sous-ensembles d'affinage cohérents. Si un sous-ensemble d'affinage cohérent contient un processus communiquant avec des sous-signaux, ce processus ne peut communiquer avec le signal ascendant et l'autre extrémité de la liaison de communication doit communiquer avec les mêmes sous-signaux.

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

L'utilisateur peut avoir besoin de décrire la transformation entre différents niveaux d'affichage à des fins de simulation ou pour contrôler le comportement du système indifférent au niveau de précision. Il peut le faire de manière informelle à l'aide de processus LDS supplémentaire décrivant 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.



T1002150-88

FIGURE D-4.7.2

Exemple de spécification d'une transformation de signal

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.

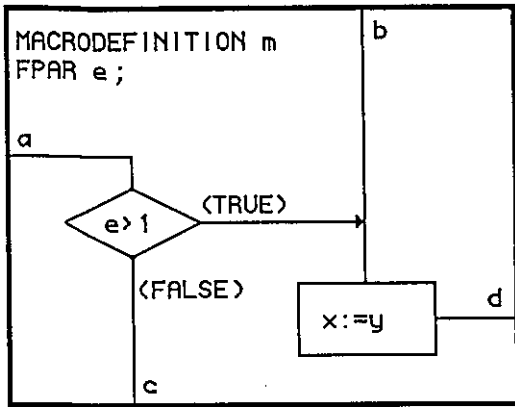
En LDS/GR, il y a deux moyens différents de représenter les accès d'entrée et de sortie d'une définition de macro. L'utilisateur peut soit tracer un cadre pour la macro et relier à celui-ci les accès d'entrée et de sortie, soit utiliser explicitement des symboles d'accès d'entrée et d'accès de sortie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



T1002160-88

LDS/GR

```

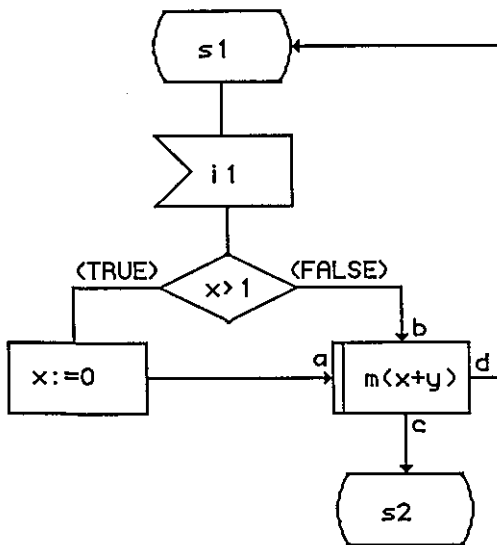
MACRODEFINITION m
  FPAR e,a,b,c,d;
  a: DECISION e>1;
    <TRUE>: JOIN b;
    <FALSE>: JOIN c;
  ENDDECISION;
  b: TASK x:=y;
  JOIN d;
ENDMACRO m;

```

LDS/PR

FIGURE D-5.1.1

Exemple d'une définition de macro



T1002170-88

LDS/GR

```

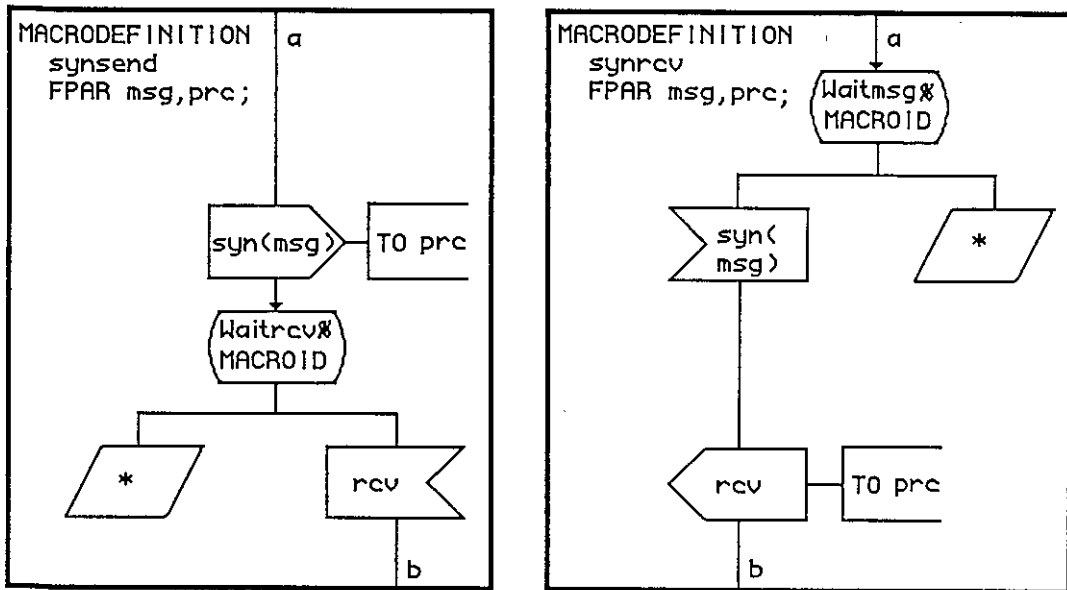
...
STATE s1;
INPUT i1;
DECISION x>1;
<FALSE>: JOIN BB;
<TRUE>: TASK x:=0;
        JOIN AA;
ENDDECISION;
MACRO m (x+y,AA,BB,CC,DD);
CC: NEXTSTATE s2;
DD: NEXTSTATE -;
...

```

LDS/PR

FIGURE D-5.1.2

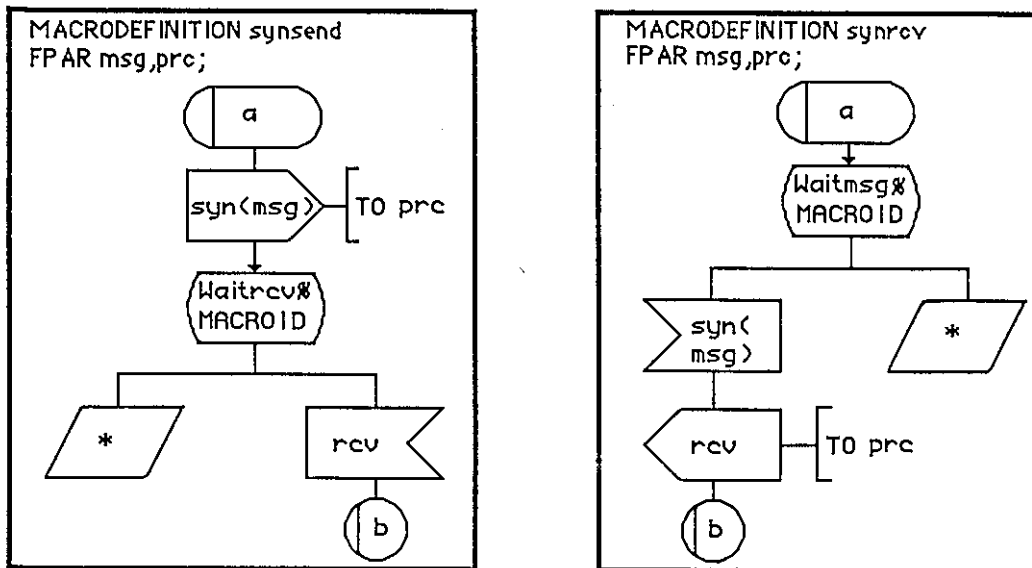
Exemple d'un appel de macro



T1002181-89

FIGURE D-5.1.3

Exemple de deux macros utilisant le paramètre MACROID



T1002190-89

FIGURE D-5.1.4

Exemple de l'emploi de symboles d'accès d'entrée et de sortie

### 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.

```

SYSTEM s;
...
  SYN inst.numb INTEGER = EXTERNAL;
  SYN rate.incr REAL = EXTERNAL;
...
  BLOCK b;
    PROCESS p (inst.numb); /* parametric instance number */
    ...
    val:=val+rate.incr; /* parametric increment */
    ...
  ENDPROCESS p;
ENDBLOCK b;
ENDSYSTEM s;

```

FIGURE D-5.2.1

**Exemple de l'emploi de paramètres de système**

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

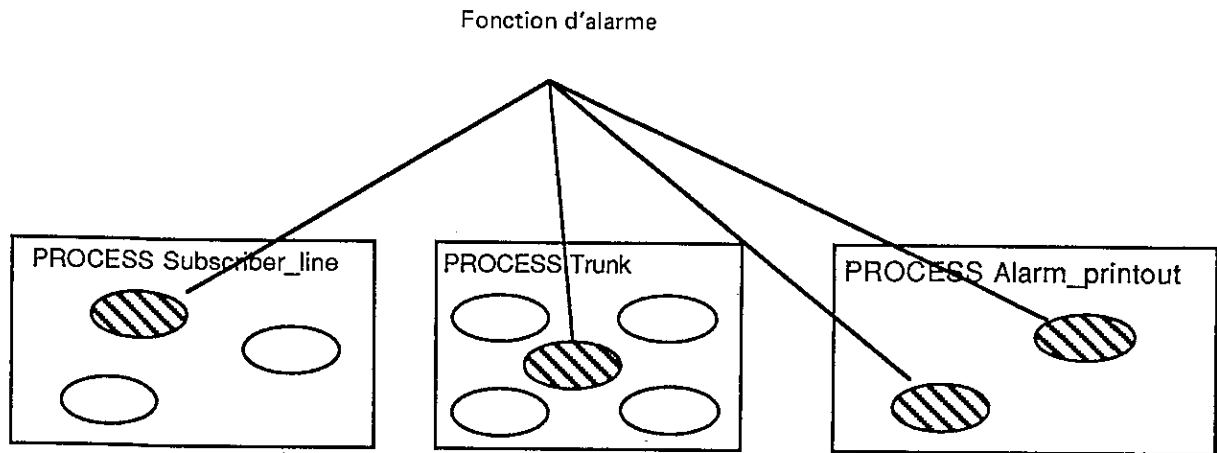
- construction SELECT: sélection conditionnelle d'une partie de spécification. La condition est spécifiée par une expression booléenne que l'on doit pouvoir évaluer statiquement, avant l'interprétation du système. Elle est choisie lorsque l'expression est VRAIE;
- construction ALTERNATIVE: permet la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



T1002200-88

FIGURE D-5.3.1

**Exemple informel illustrant la manière dont des services de différents processus peuvent composer une fonction supérieure du système**

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 influencer les uns sur les autres par suite de manipulations de données.

Le comportement d'un processus n'étant pas modifié lorsqu'il est subdivisé en services, les services représentent seulement une autre description du même comportement. On peut naturellement décider au 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:

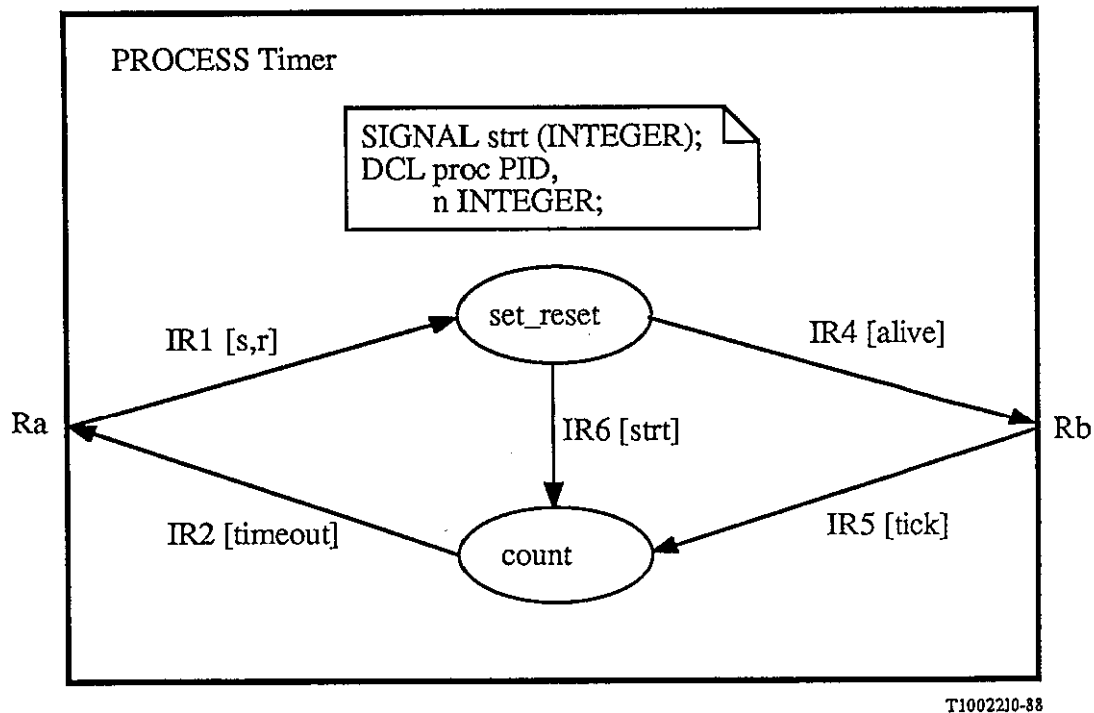
- nom de service;
- ensemble de signaux d'entrée valides: liste d'identificateurs de signaux définissant des signaux qui peuvent être reçus par le service;
- définitions de procédure: spécification des procédures qui sont locales au service. On peut aussi utiliser une référence de procédure;
- définitions 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 variables 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;
- définitions de visibilité: déclaration 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 être spécifiée;
- définitions d'import: spécification d'identificateurs de variables appartenant à d'autres processus, que le service désire importer. Pour chaque identificateur, la sorte de variable doit être spécifiée;

- 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, les 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).



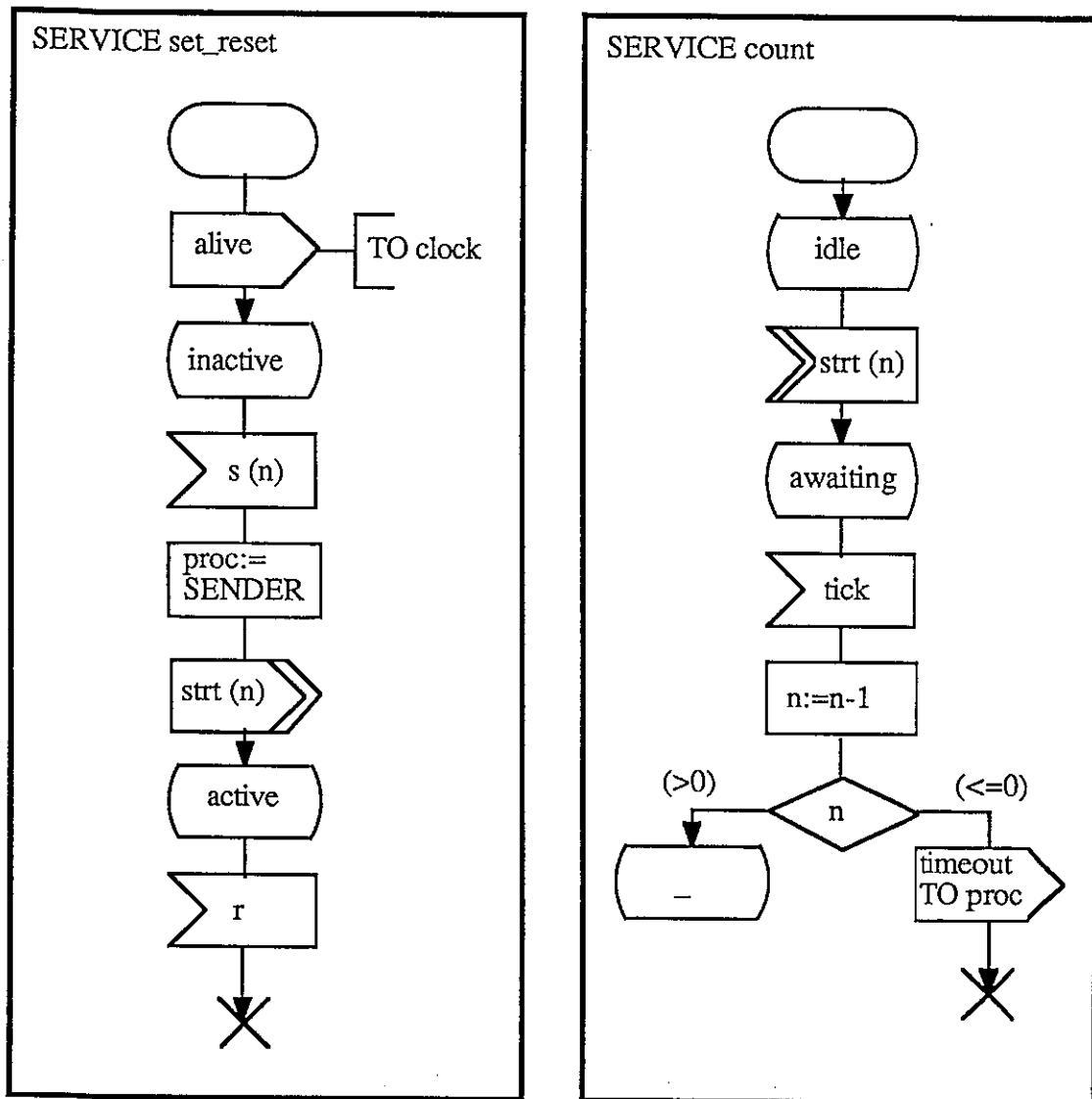
T1002210-88

FIGURE D-5.3.2

Diagramme de processus avec services référencés

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





T1002220-88

FIGURE D-5.3.3  
Diagrammes de service

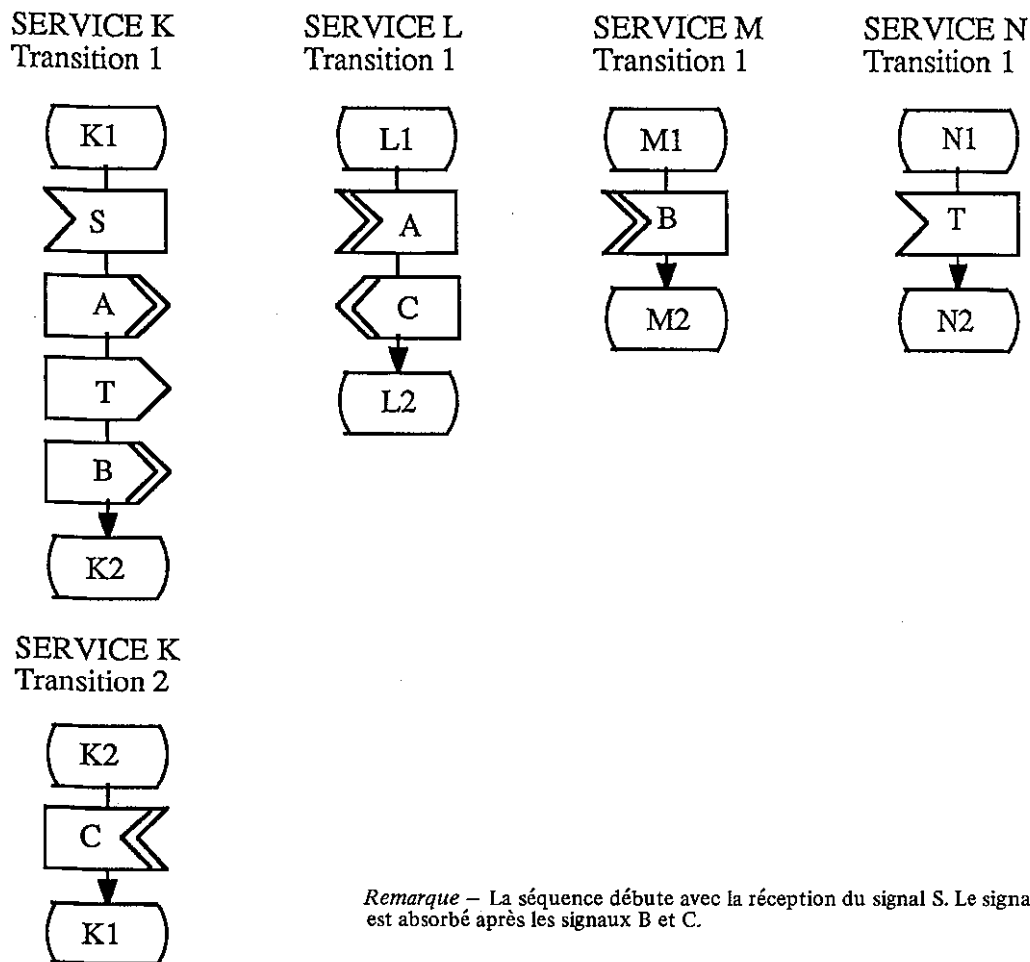
Il convient de noter que, les actions (sorties) étant comprises dans la transition de départ dans le premier service, aucune action n'est autorisée dans la transition de départ du second service.

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

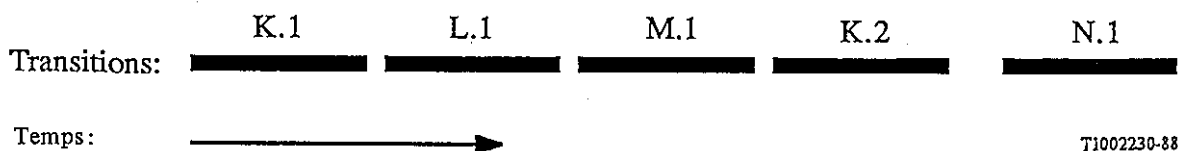
#### D.5.3.2 Signaux prioritaires

Un signal prioritaire est employé pour la communication entre deux transitions dans des services différents d'un processus lorsqu'aucun signal extérieur (provenant d'autres processus) ne peut être absorbé dans le laps de temps entre les transitions. Ainsi, les transitions sont «concaténées».

La figure D-5.3.4 est une illustration de la concaténation des transitions lorsque des signaux prioritaires sont utilisés.



Remarque - La séquence débute avec la réception du signal S. Le signal ordinaire T est absorbé après les signaux B et C.



T1002230-88

FIGURE D-5.3.4

**Concaténation de transitions dans différents services d'un processus (LDS informel)**

La construction permettant d'obtenir des signaux prioritaires utilisant la file d'attente d'entrée ordinaire est décrite dans la Recommandation. Les signaux prioritaires, émis vers un état préliminaire, peuvent être traités comme des signaux ordinaires et provoquent des transitions vers l'état principal (voir aussi le § D.5.3.3.1).

L'exemple suivant est une illustration des conséquences de l'emploi de signaux prioritaires. L'exemple est expliqué sans l'emploi du modèle «état préliminaire principal».

Le diagramme de séquençage (voir la figure D-5.3.5) est obtenu de l'interaction entre les services du processus «SUBSCRIBER\_LINE» décrit au § D.10.2.

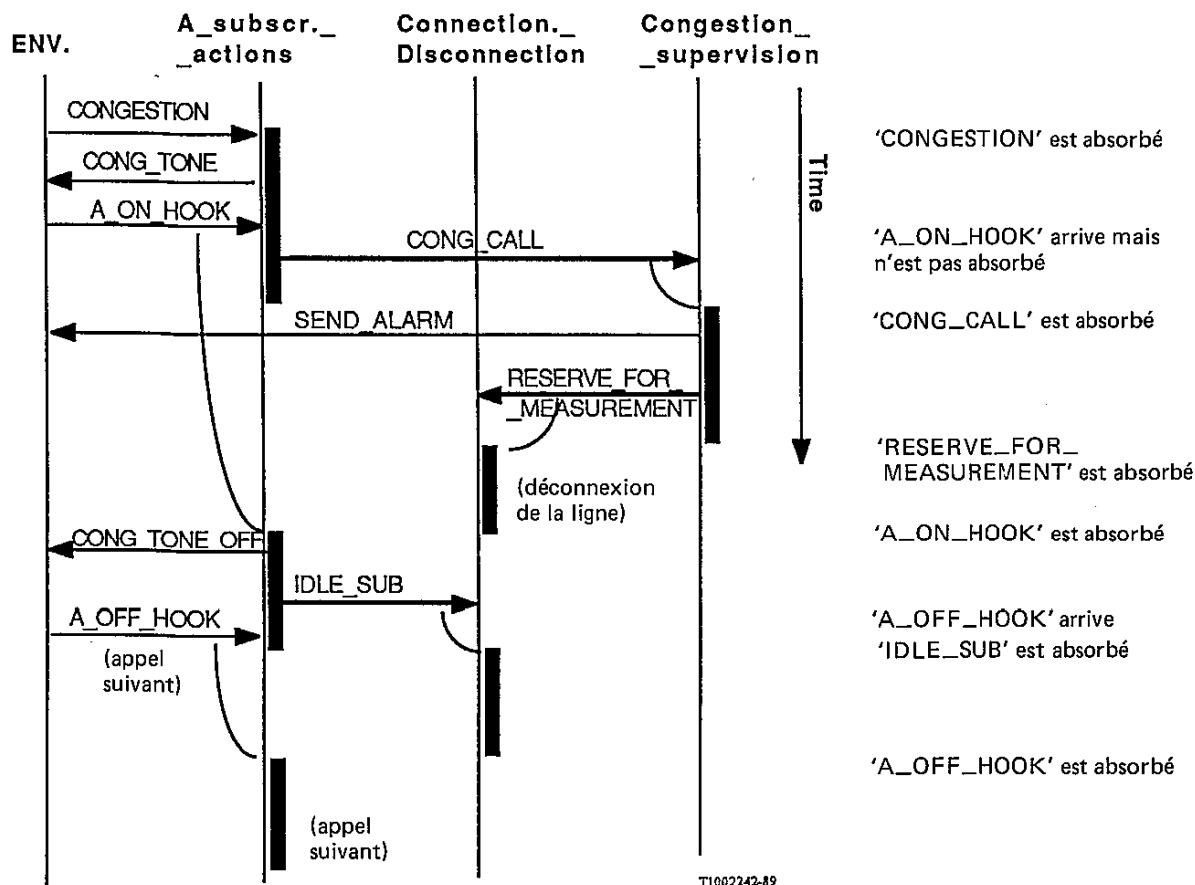


FIGURE D-5.3.5

Diagramme de séquençage montrant comment des signaux prioritaires peuvent modifier l'ordre des transitions. Chaque trait vertical épais indique l'exécution d'une transition

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 flèches de signaux (de haut en bas) indique comment les signaux sont placés dans la file d'attente. Les lignes verticales en traits épais indiquent comment l'exécution des transitions s'ordonne dans le temps.

La séquence débute avec le signal «CONGESTION» en provenance du registre, ce qui signifie que l'appel doit être rejeté. Cela signifie aussi le rejet immédiat de l'appel suivant.

La figure montre qu'une transition, activée 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», malgré l'ordre inverse de celui de sa mise en file d'attente. La transition activée par le signal «RESERVE\_FOR\_MEASUREMENT» déconnecte la ligne d'abonné, ce qui signifie que l'appel suivant (signal «A\_OFF\_HOOK») sera immédiatement rejeté.

Dans le diagramme de séquençage 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.

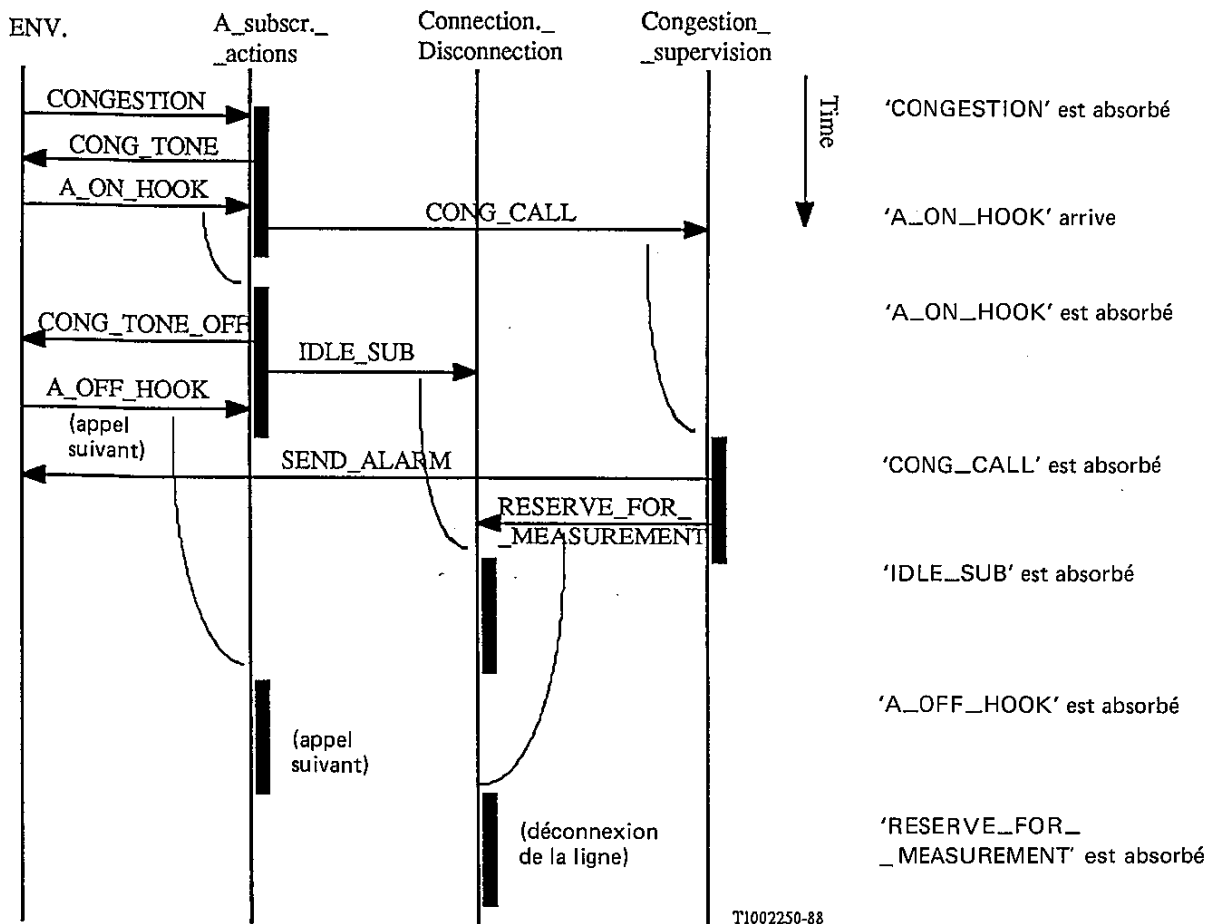


FIGURE D-5.3.6

Diagramme de séquençage avec signaux ordinaires et ordre normal des transitions

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'états

La Recommandation donne des règles spécifiques pour la transformation d'états. Le résultat de ces règles est que le nombre d'états du processus est le produit du nombre d'états qui se trouvent dans les services.

En outre, l'emploi de signaux prioritaires doublera ce produit car chaque état transformé est divisé en un état préliminaire et un état principal.

Ce doublement du nombre des états est dû aux signaux prioritaires qui ne sont pas traités dans l'exemple suivant de transformation d'états.

Dans bien des cas, de nombreux états du processus «transformé» ne sont pas pertinents et l'espace d'état peut être réduit. Cela est traité dans l'exemple suivant, tiré 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'états est de 10, 5, 3 et , c'est-à-dire 20 au total.

Appliquant à ces services les règles pour la transformation d'états, nous avons 300 états ( $10*5*3*2$ ) dans le processus, ce qui signifie 300 multiplats de noms. La dimension du multiplat est de 4 (4 services). Les positions du multiplat sont arrangées selon la figure D-5.3.7.

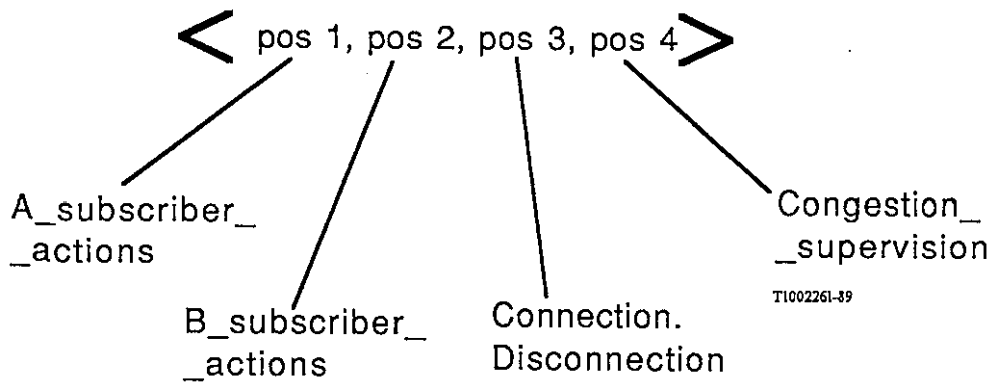


FIGURE D-5.3.7

**Arrangement de positions dans un multiplat de noms**

Tous les noms d'état possibles dans les différentes positions donnent 300 combinaisons.

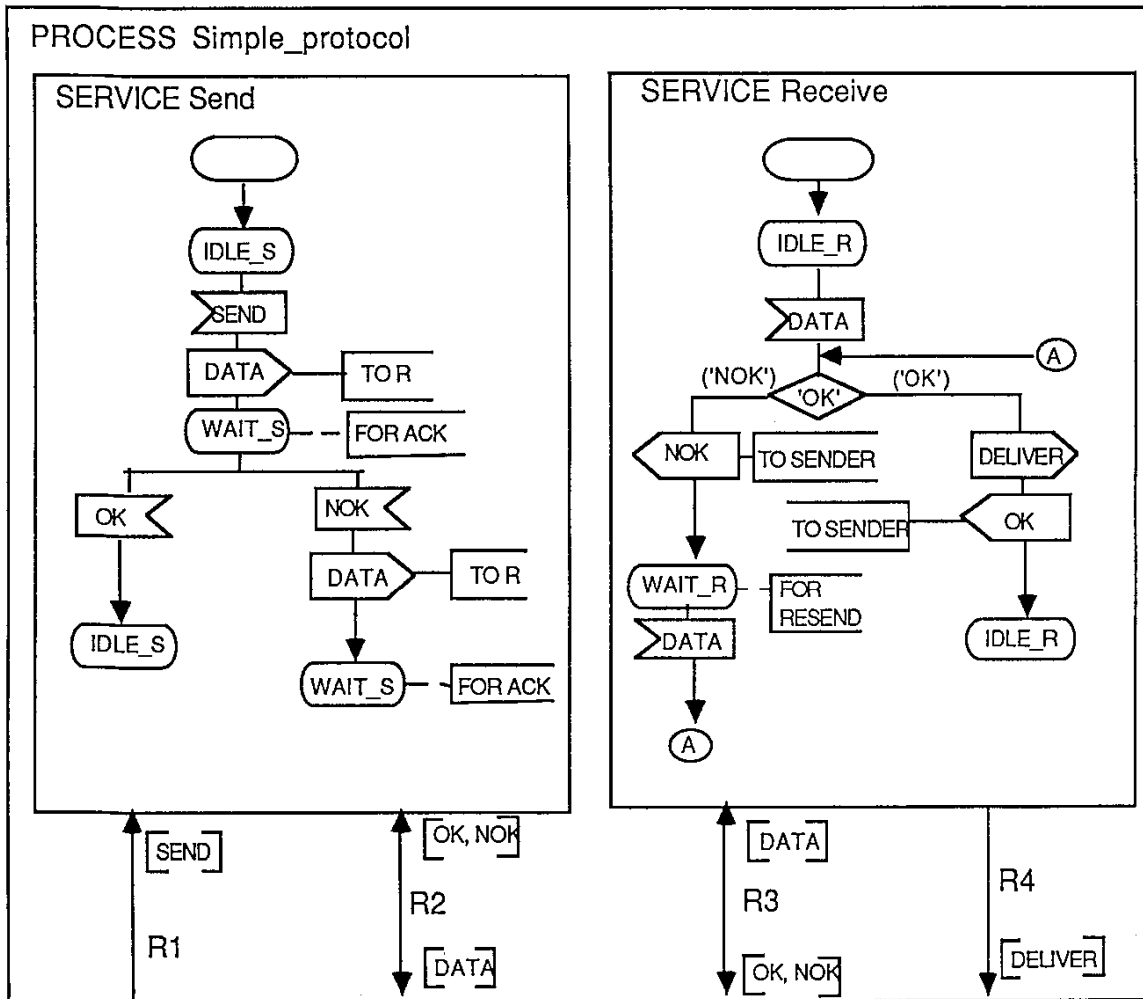
- Ex.<A\_IDLE, B\_IDLE, CONNECTED, NO\_ALARM>
- <AWAIT\_CONN, B\_IDLE, CONNECTED, NO\_ALARM>
- <AWAIT\_FIRST\_DIGIT, B\_IDLE, CONNECTED, NO\_ALARM>
- ....
- <AWAIT\_A\_ON\_HOOK\_2, B\_IDLE, CONNECTED, NO\_ALARM>
- <A\_IDLE, B\_RINGING, CONNECTED, NO\_ALARM>
- <A\_IDLE, B\_CONVERSATION, CONNECTED, NO\_ALARM>
- ....
- <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'abonné ne pourra jamais être utilisée à la fois pour l'abonné A et l'abonné B au cours du même appel. Cela signifie que les 10 états des `A\_subscriber actions` ne peuvent être combinés qu'avec un seul état des `B\_subscriber actions` c'est-à-dire `B\_IDLE`. Cela signifie en outre que les 5 états de `B\_subscriber actions` ne peuvent être combinés qu'avec un seul état de `A\_subscriber actions` (A\_IDLE). Le nombre d'états pertinents est donc de  $(10*1*3*2) + (1*5*3*2) = 90$ .

Une réduction de l'espace d'état, comme dans l'exemple ci-dessus, dépend du cas dont il s'agit et ne peut faire l'objet de règles générales formelles, applicables en vue d'une transformation automatique. Cependant, si la transformation est exécutée manuellement, il est probable que l'espace d'état peut être réduit. Ainsi, lorsque l'on compare la complexité d'un processus conçu sans services à celle du même processus subdivisé en services, il convient d'utiliser le processus à espace d'état réduit pour obtenir une bonne base de comparaison. Dans la plupart des cas, le recours aux services permettra de réduire considérablement le nombre des états.

**D.5.3.3.2 Transformation de transitions**

La Recommandation contient des règles spécifiques applicables à la transformation de transitions. L'exemple qui suit illustre les modalités de transformation. Le processus de cet exemple est une spécification d'un protocole simple. Le processus est subdivisé en deux services, émission (send) et réception (receive).



T1002271-89

FIGURE D-5.3.8

Diagramme de processus comprenant deux diagrammes de service

La figure D-5.3.9 est un diagramme synoptique d'état pour les deux services. Les transitions sont numérotées de 1 à 7.

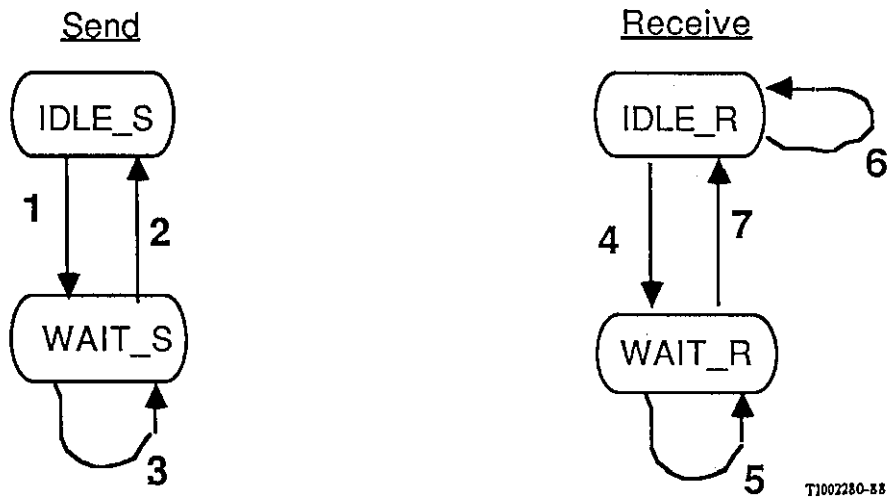


FIGURE D-5.3.9

Diagrammes synoptiques d'état avec transitions numérotées

Si l'on applique les règles formelles de transformation d'états et de transitions, on obtient pour le processus le diagramme synoptique d'état ci-après. Aucun signal prioritaire n'est utilisé, de sorte qu'il n'est pas nécessaire de diviser plus avant les états en états préliminaires et états principaux et cela n'est donc pas indiqué.

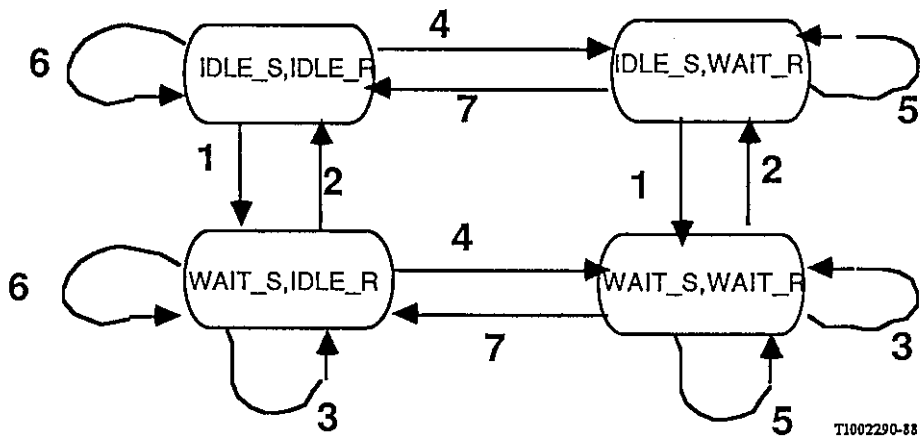


FIGURE D-5.3.10

Diagramme synoptique d'état pour les services transformés

Le nombre d'états ne peut être réduit et le graphe de processus est représenté dans la figure D-5.3.11. Les noms d'états 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.

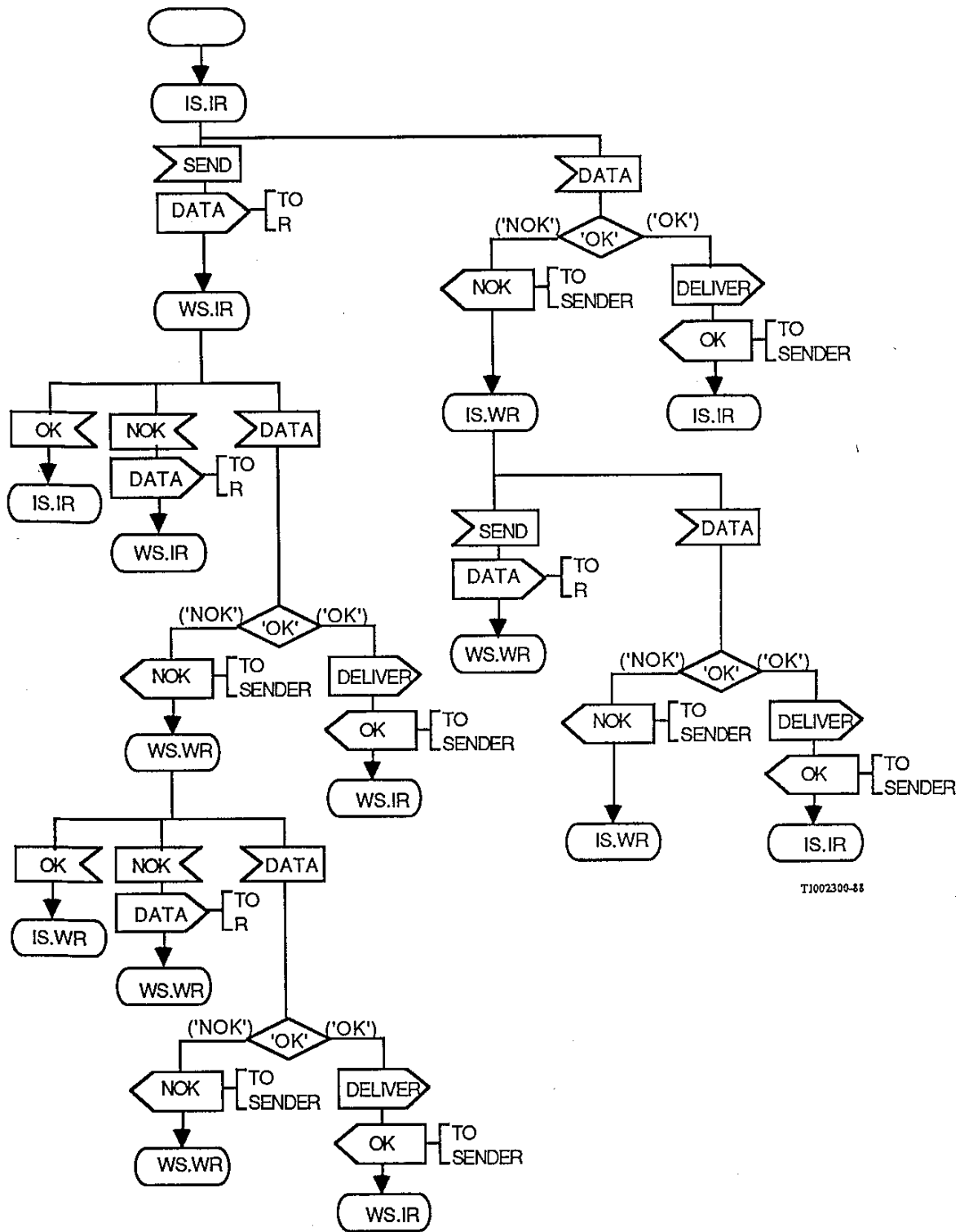


FIGURE D-5.3.11

Grphe de processus transformé à partir des deux graphes de service

D.5.4 Directives applicables à la représentation en fonction des états et aux éléments graphiques

Le présent paragraphe décrit la représentation en fonction des états du LDS/GR et des éléments graphiques utilisés.

Le § D.5.4.1 contient des observations d'ordre général sur la représentation en fonction des états: caractéristiques, applications pertinentes et variantes.



Le § D.5.4.2 explique la description d'états au moyen d'éléments graphiques.

#### D.5.4.1 *Observations d'ordre général sur la représentation en fonction des états*

Le LDS/GR offre trois versions différentes pour décrire un diagramme de processus.

La première est appelée version du LDS/GR en fonction des transitions, dans laquelle les transitions sont décrites exclusivement par des symboles d'action explicites.

La seconde est appelée version du LDS/GR orientée vers les états ou extension graphique du LDS orientée vers les états; les états d'un processus y sont décrits à l'aide d'illustrations d'états et les transitions ne sont données qu'implicitement, par les différences entre les états de départ et d'arrivée.

La dernière est appelée version mixte; il s'agit d'une combinaison des deux versions précédentes.

On trouvera des exemples de ces trois versions dans l'annexe E du présent fascicule.

La version orientée vers les transitions convient lorsque la séquence des actions présente plus d'importance que la description détaillée des états.

La version orientée vers les états décrit en détail les états par des énoncés; elle convient donc au cas où un état de processus présente plus d'importance que la séquence des actions à l'intérieur de chaque transition, alors que l'explication graphique intuitive est souhaitable et qu'il est intéressant de connaître les ressources ainsi que leurs relations avec les états.

Les illustrations d'état sont généralement exprimées par des éléments graphiques indiquant les ressources pertinentes de l'état en cours du processus. Cette version convient à des applications dans lesquelles sont définis des éléments graphiques appropriés; l'utilisateur peut donc employer cette représentation pour n'importe quelle application en définissant des éléments graphiques appropriés selon les besoins.

La version mixte convient lorsqu'il faut connaître à la fois la séquence des actions intérieures à chaque transition et les descriptions détaillées des états.

#### D.5.4.2 *Illustration d'état et élément graphique*

##### D.5.4.2.1 *Élément graphique et texte qualificatif*

Si l'on a choisi l'option illustration d'état, celle-ci se compose d'éléments graphiques et d'un texte qualificatif, comme indiqué dans la figure D-5.4.1 a) à D-5.4.1 d).

Cette combinaison rend compréhensibles les illustrations d'état. A titre d'exemple, la figure D-5.4.1 a) donne la signification d'un récepteur à cadran manipulé par le processus, l'exemple b) celle d'un émetteur de tonalité de numérotation émettant un signal permanent vers l'environnement.

A noter que les signaux de sortie (signaux non permanents) et les ressources pertinentes ne sont pas décrits dans les illustrations d'état; les signaux de sortie peuvent être décrits dans un diagramme de transition.

L'exemple c) montre un temporisateur dont l'expiration est toujours représentée par une entrée. A noter que l'illustration graphique recommandée pour le temporisateur comporte le signal d'entrée pertinent t1.

Le dernier exemple, d), signifie qu'un enregistreur de messages vocaux est en cours de fonctionnement.

L'identité de la ressource peut être considérablement abrégée et devrait, si possible, être placée à l'intérieur de l'illustration graphique appropriée. Ainsi, les éléments graphiques qui sont qualifiés sont tout à fait évidents.

##### D.5.4.2.2 *Illustrations d'état complètes*

Chaque illustration d'état doit comporter un nombre suffisant d'éléments graphiques afin de montrer:

- a) quelles ressources le processus met en œuvre au cours de l'état représenté. Exemples: trajets de commutation, récepteurs de signalisation, émetteurs de signaux permanents et modules de commutation;
- b) s'il y a en ce moment un ou plusieurs temporisateurs qui contrôlent le processus;
- c) dans le cas où le processus concerne le traitement des appels, si la taxation est ou non actuellement en cours et quels abonnés sont taxés au cours de cette phase de l'appel;
- d) quels objets appartenant effectivement à un autre processus (environnement) sont considérés comme en relation avec des ressources du processus pendant l'état en cours;
- e) les signaux permanents de sortie qui sont émis dans cet état;
- f) la relation entre les signaux et ressources existants dans l'état;

g) l'inventaire des ressources concernant l'état en cours du processus.

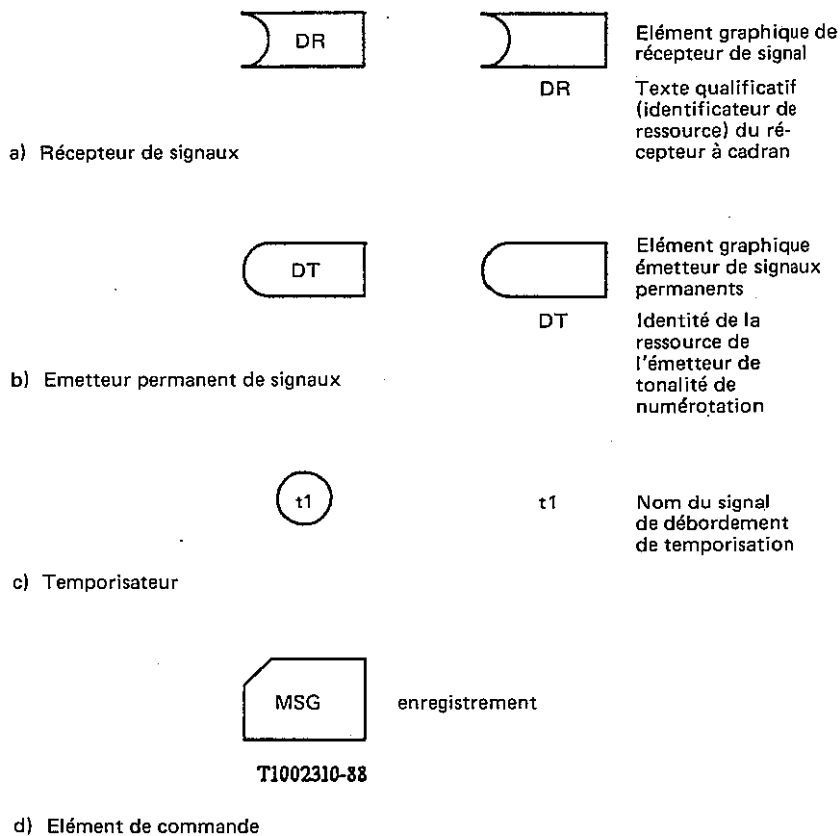


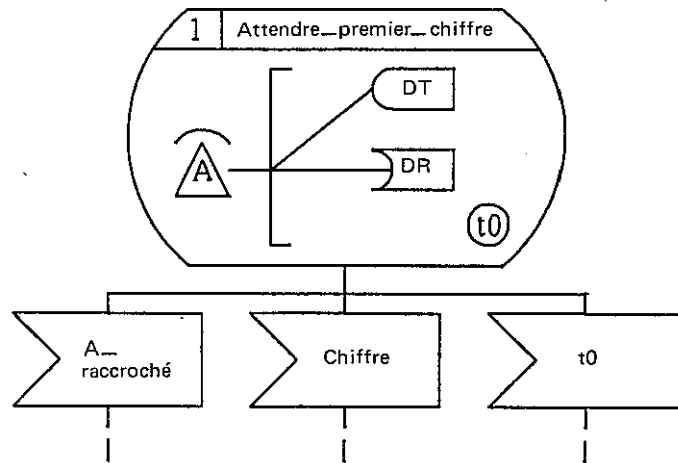
FIGURE D-5.4.1

Exemples d'éléments graphiques avec textes qualificatifs

#### D.5.4.2.3 Exemple

A titre d'exemple d'application des principes exposés ci-dessus, considérons l'état de la figure D-5.4.2. On peut voir que, dans cet état:

- les ressources affectées aux processus sont: un récepteur de chiffres à cadran, un émetteur de tonalités de numérotation, un poste d'abonné appartenant à l'environnement et des trajets de communication reliant ces organes;
- un temporisateur t0 surveille le processus;
- aucune taxation n'est en cours;
- l'abonné est identifié comme l'abonné A mais aucune autre information de catégorie n'est prise en considération;
- les signaux d'entrée suivants sont attendus: Anook (A rattaché), digit (chiffre) (chiffre numéroté) et t0 (le temporisateur de supervision t0 fonctionne);
- le signal permanent de sortie DT (tonalité de numérotation) a été mis avant cet état et pendant celui-ci.



T1002320-88

FIGURE D-5.4.2

Exemple d'un état d'un processus de traitement d'appel

#### D.5.4.2.4 Vérification de la cohérence de diagrammes LDS avec éléments graphiques

On constate que l'illustration d'état est plus compacte et que, d'une certaine manière, elle offre au lecteur plus d'informations; cependant, l'identification de la série exacte d'opérations accomplies au cours de la transition exige un examen très attentif des ressources.

En outre, une simple observation de l'illustration d'état ne permet pas de déterminer l'ordre des actions dans la transition.

Les éléments graphiques représentés à l'extérieur du bloc sont des éléments qui ne sont pas directement commandés par le processus donné: ceux qui sont représentés à l'intérieur du symbole «limites du bloc» sont directement commandés par ce processus. Par exemple, le processus d'appel partiellement spécifié dans la figure D-5.4.3 peut allouer ou libérer l'émetteur de tonalité d'appel, l'émetteur de sonnerie et les trajets de conversation; il peut également déclencher ou arrêter le temporisateur t4, mais il ne peut commander directement la condition du combiné de l'abonné.

En concevant la logique à partir d'une spécification LDS avec éléments graphiques, seuls les éléments graphiques représentés à l'intérieur des limites du bloc ont une influence sur les actions exécutées pendant les séquences de la transition. Les éléments graphiques complexes représentés à l'intérieur des limites de ce bloc sont normalement inclus dans une illustration d'état:

- a) soit parce qu'ils indiquent les ressources et l'état de l'environnement concernés par le signal d'entrée du processus pendant l'état donné;
- b) soit pour améliorer l'intelligibilité du diagramme.

#### D.5.4.2.5 Utilisation du symbole «temporisateur»

Que l'on emploie ou non des éléments graphiques, l'expiration d'un délai de temporisation est toujours représentée par une entrée.

La présence d'un symbole de temporisateur dans une illustration d'état implique qu'un temporisateur fonctionne pendant cet état.

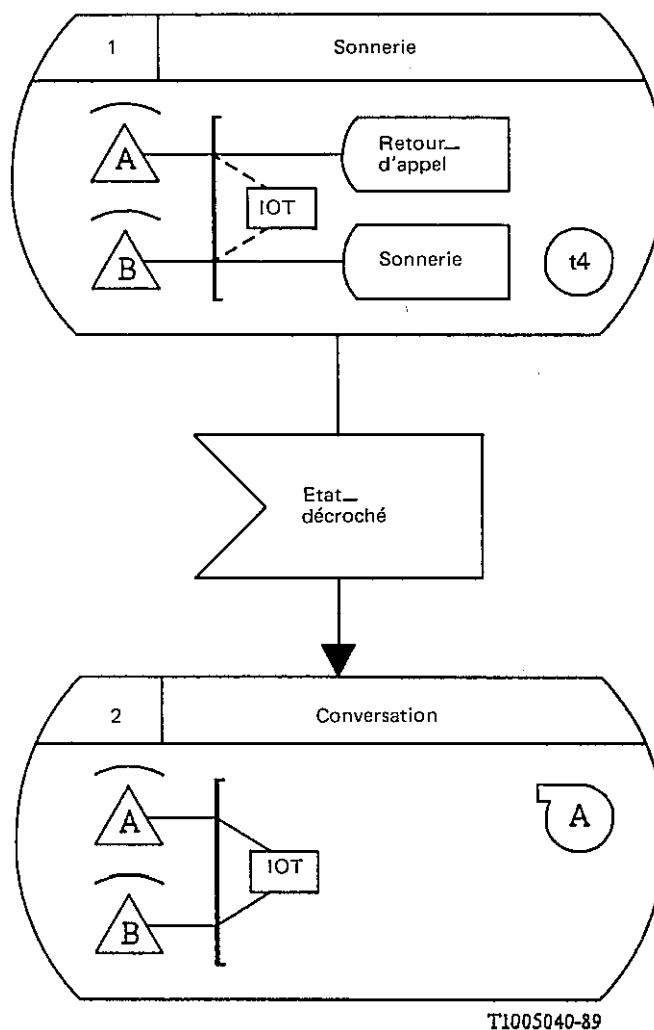
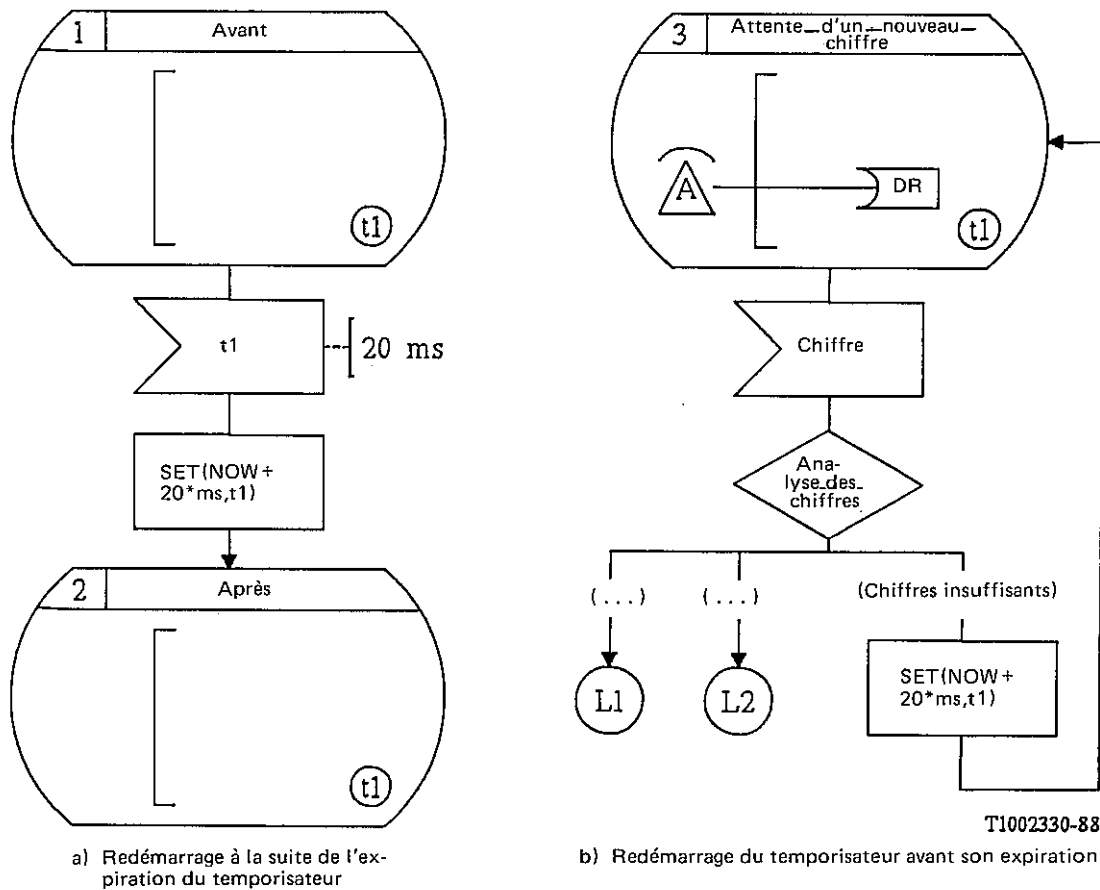


FIGURE D-5.4.3

Exemple d'une transition entre deux états dans lequel toutes les actions de traitement découlent implicitement les différences entre les illustrations d'état

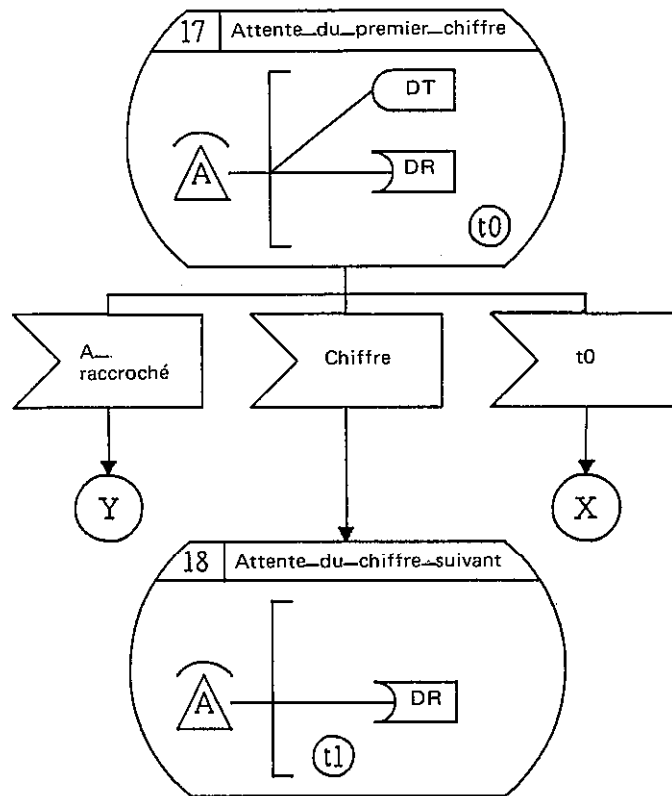
Conformément au principe général exposé dans les Recommandations, le démarrage, l'arrêt, le redémarrage et l'expiration du temporisateur sont représentés à l'aide d'éléments graphiques de la manière suivante:

- pour montrer qu'une temporisation commence au cours d'une transition donnée, le symbole temporisateur doit apparaître sur l'illustration d'état qui correspond à la fin de cette transition et non sur celle qui correspond à son début;
- inversement, pour montrer qu'une temporisation s'arrête au cours d'une transition, le symbole temporisateur doit apparaître sur l'illustration d'état qui correspond au début de cette transition et non sur celle qui correspond à sa fin;
- pour montrer qu'une temporisation est relancée au cours d'une transition un symbole explicite de tâche doit être représenté dans cette transition (on en voit deux exemples sur la figure D-5.4.4);
- l'expiration du délai d'une temporisation donnée est représentée par un symbole d'entrée associé à un état dont l'illustration porte le symbole «temporisateur» correspondant. Il peut naturellement arriver que plusieurs temporisateurs surveillent à la fois le même processus (voir la figure D-5.4.5).



Remarque – Chaque temporisateur  $t1$  a deux états qui s'excluent mutuellement: actif et inactif.

FIGURE D-5.4.4  
Redémarrage d'un temporisateur



T1002340-88

*Remarque* – t0 surveille l'arrivée du premier chiffre, à la suite de quoi la tonalité de numérotation est éliminée et t0 arrêté. t1 continue à surveiller l'arrivée d'un nombre de chiffres suffisant pour que l'appel soit convenablement acheminé.

FIGURE D-5.4.5

**Exemple d'utilisation de deux temporisateurs de surveillance dans le même état**

D.5.5 *Diagrammes auxiliaires*

Pour faciliter la lecture et la compréhension de diagrammes de processus de grande taille et/ou complexes, l'auteur peut y ajouter des diagrammes auxiliaires informels. De tels documents ont pour but de donner une description synoptique ou simplifiée du comportement du processus (ou d'une partie de celui-ci). Les documents auxiliaires ne remplacent pas les documents en LDS mais constituent une introduction à ceux-ci.

On trouvera dans la présente section des exemples de certains diagrammes auxiliaires couramment utilisés, notamment des diagrammes synoptiques d'état, des matrices état/signal, et des diagrammes de séquençement. (Le diagramme en arbre de bloc décrit au § D.4.4 est également un diagramme auxiliaire.)

D.5.5.1 *Diagramme synoptique d'état*

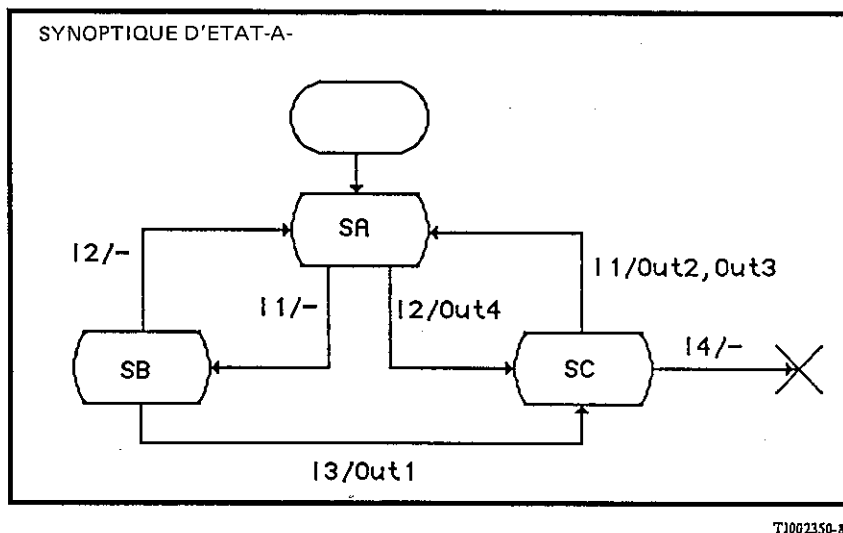
Son objectif est de donner une vue d'ensemble des états d'un processus, et d'indiquer quelles transitions sont possibles entre eux. Etant donné qu'il s'agit de donner un aperçu, l'on peut négliger les états ou les transitions de peu d'importance.

Les diagrammes se composent de symboles d'état, de flèches représentant des transitions et, éventuellement, de symboles de début et d'arrêt.

Le symbole d'état doit indiquer le nom de l'état référencé. Plusieurs noms d'état peuvent être inscrits dans le symbole, et il est possible d'employer un astérisque (\*) pour désigner tous les états.

Chacune des flèches peut, de même que chacune des sorties possibles pendant la transition, être associée au nom du signal ou de l'ensemble de signaux qui déclenchent la transition.

On trouvera dans la figure D-5.5.1 un exemple de diagramme synoptique d'état.

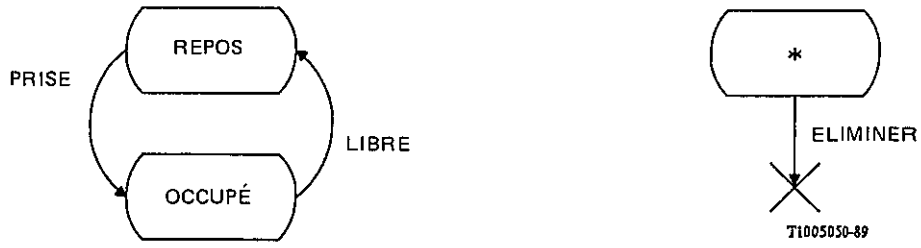


T1002350-88

FIGURE D-5.5.1

Exemple de diagramme synoptique d'état

Il est possible de répartir sur plusieurs diagrammes le diagramme synoptique d'état d'un processus; chacun des diagrammes obtenus porte alors sur un aspect particulier, comme «cas normal», traitement en cas de défaillance, etc.



T1005050-89

FIGURE D-5.5.2

Exemple de diagramme synoptique d'état réparti

D.5.5.2 Matrice état/signal

La matrice état/signal doit servir de document «préliminaire» à un diagramme de processus important. Elle indique les endroits où existent des combinaisons entre un état et un signal dans le diagramme.

Le diagramme se compose d'une matrice bidimensionnelle; celle-ci présente sur un axe tous les états d'un processus, et sur l'autre tous les signaux d'entrée valides d'un processus. L'état suivant est donné pour chaque élément de matrice, de même que les sorties possibles au cours de la transition. Une référence peut être indiquée pour permettre de trouver la combinaison donnée par les indices, si elle existe.

L'élément correspondant à l'état fictif ou «START» et au signal fictif «CREATE» sert à indiquer l'état initial du processus.

ETAT/SIGNAL A				
STATE \ SIGNAL	"START"	SA	SB	SC
"CREATE"	SA/-	-	-	-
I 1	-	SB/-	-	SA/Out2,Out3
I 2	-	SC/Out4	SA/-	-
I 3	-	-	SC/Out1	-
I 4	-	-	-	"STOP" /-

FIGURE D-5.5.3

Exemple de matrice état/signal

Il est possible de fractionner la matrice en sous-parties réparties sur plusieurs pages. Les références sont celles qu'emploie normalement l'utilisateur dans la documentation.

Les signaux et les états doivent être de préférence regroupés de façon que chaque partie de la matrice porte sur un aspect particulier du comportement du processus.

#### D.5.5.3 Diagramme de séquençement

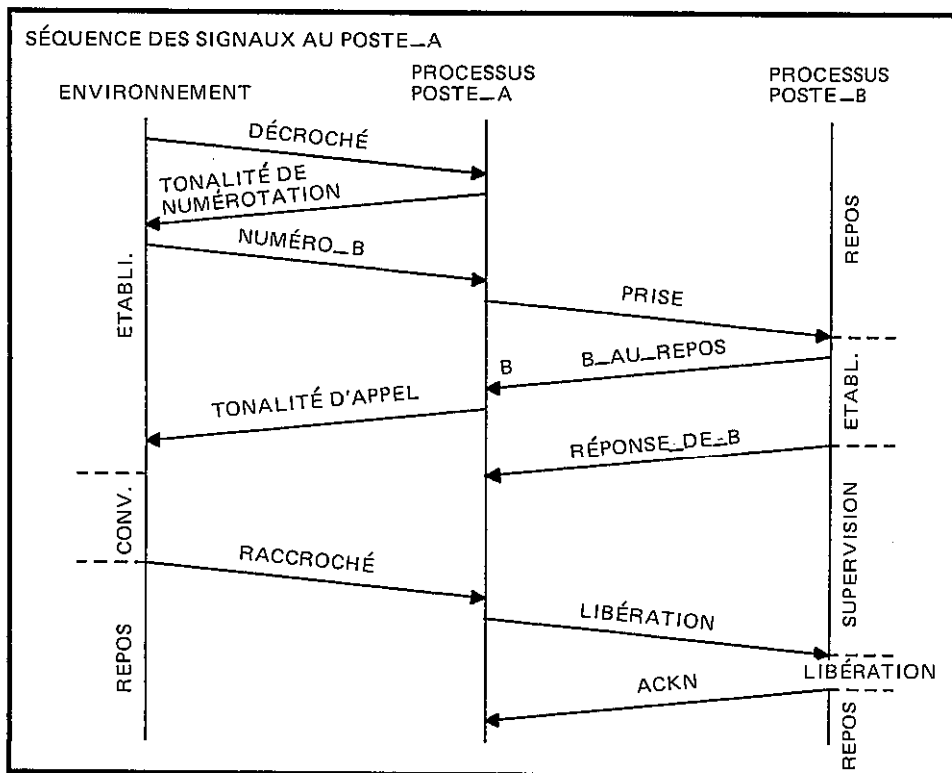
Le diagramme de séquençement peut servir à montrer l'échange des séquences de signaux autorisées entre un ou plusieurs processus et leur environnement.

Ce diagramme doit donner une vue d'ensemble de l'échange de signaux entre les parties du système. Ce diagramme peut représenter l'ensemble ou une partie de l'échange de signaux, en fonction des aspects à mettre en évidence.

Les colonnes du diagramme indiquent les entités en communication (services, processus, blocs ou l'environnement).

Leurs interactions sont illustrées par un ensemble de lignes fléchées, dont chacune représente un signal ou un ensemble de signaux.





T1002361-89

FIGURE D-5.5.4

Exemple de diagramme de séquençement

On peut annoter chaque séquence afin de faire apparaître clairement l'ensemble d'informations échangées. Chaque ligne est accompagnée d'une annotation qui donne les renseignements requis (noms des signaux ou de procédures).

On peut placer un symbole de décision dans les colonnes pour indiquer que la séquence suivante est valide si la condition indiquée est vraie. Dans ce cas, le symbole de décision apparaît généralement plusieurs fois; il indique les différentes séquences produites par chacune des valeurs de la condition.

Ce diagramme peut représenter la totalité ou seulement un sous-ensemble significatif des séquences de signaux échangés.

La représentation de l'interaction réciproque des services résultant de la subdivision d'un processus représente une application utile de ce type de diagramme.

Les diagrammes de séquençement ne comprennent généralement pas toutes les séquences possibles; ils constituent souvent un préalable à la définition complète.

## D.6 Définition des données en LDS

### D.6.1 Directives applicables aux données en LDS

On trouvera dans la présente section des renseignements complémentaires sur les concepts définis au § 5 de la Recommandation concernant le LDS. La principale différence entre la présente section et l'ancienne Recommandation Z.104 est que cette dernière a fait l'objet d'une révision à des fins de clarification et d'harmonisation avec l'ISO. Certains des mots clés ont été modifiés et des adjonctions ont été faites mais la cohérence de la sémantique a été assurée avec le Livre rouge. L'emploi des données décrites aux § 2, 3 et 4 de la nouvelle Recommandation (auparavant Z.101-Z.103) est resté tel quel.

#### D.6.1.1 Introduction générale

Les types de données du LDS sont fondés sur l'approche de «types de données abstraits»: on ne décrit pas la manière dont un type doit être mis en œuvre, mais on se borne à dire quel sera le résultat des opérateurs appliqué aux valeurs.

Lorsque l'on définit des données abstraites, chaque segment de la définition, appelée «définition de type partielle» est introduit par le mot-clé NEWTYPE. Chaque définition de type partielle a une incidence sur les autres, de sorte que toutes les définitions de type partielles au niveau du système constituent une définition de type de données unique. Si plusieurs définitions de type partielles sont introduites à un niveau inférieur (niveau de bloc, par exemple), leur ensemble constitue, avec les définitions de niveau supérieur, une définition de type de données. Cela signifie qu'en un point quelconque de la spécification, il n'existe qu'une seule définition de type de données.

En substance, la définition de type de données comprend trois parties:

- a) définitions de sortes;
- b) définitions d'opérateurs;
- c) équations.

Chacune de ces parties fait l'objet d'explications dans les paragraphes qui suivent. La définition de type de données est structurée en définitions de type de données partielles, chacune introduisant une sorte. Les définitions d'opérateurs et les équations recouvrent les définitions de type de données partielles.

#### D.6.1.2 Sortes

Une sorte est un ensemble de valeurs qui peut avoir un nombre fini ou infini d'éléments mais ne peut être vide.

*Exemples:*

- a) l'ensemble de valeurs de la sorte booléenne est {True (vrai), False (faux)};
- b) l'ensemble de valeurs de la sorte Natural (naturel) est l'ensemble infini des nombres naturels {0, 1, 2, . . .};
- c) l'ensemble de valeurs de la sorte Couleur\$\$-primaire est {Vert, Rouge, Bleu}.

Tous les éléments d'une sorte ne doivent pas être directement fournis par l'utilisateur (cela exigerait un temps infini dans le cas des nombres naturels), mais le nom de la sorte doit être indiqué. Dans la syntaxe concrète, le mot-clé NEWTYPE est directement suivi par le nom de la sorte (certaines autres possibilités seront indiquées plus loin). Ce nom est surtout utilisé dans les définitions d'opérateurs, comme expliqué au § D.6.1.3, et dans les déclarations de variables.

#### D.6.1.3 Opérateurs, littéraux et termes

Les valeurs d'une sorte peuvent être définies de trois manières:

- a) par une énumération: les valeurs sont définies dans la section des littéraux;
- b) par des opérateurs: les valeurs sont données comme les résultats d'«applications» d'opérateurs;
- c) par une combinaison d'énumérations et d'opérateurs.

La combinaison de littéraux et d'opérateurs donne des termes. Les relations entre les termes sont exprimées à l'aide d'équations. Les paragraphes qui suivent traitent des littéraux, des opérateurs et des termes; le § D.6.1.4 traite des équations.

##### D.6.1.3.1 Littéraux

Les littéraux sont des valeurs énumérées d'une sorte. Une définition de type partiel ne doit pas nécessairement comporter de littéraux: tous les éléments de la sorte peuvent être définis au moyen d'opérateurs. Les littéraux peuvent être considérés comme des opérateurs dépourvus d'arguments. Une relation entre les littéraux peut être exprimée dans des équations. Dans la syntaxe concrète, les littéraux sont introduits après le mot-clé LITERALS.

*Exemples:*

- a) La définition de la sorte booléenne contient deux littéraux, à savoir True (vrai) et False (faux). Ainsi, la définition du type booléen se présente comme suit:

```
EWTYPE Boolean
  LITERALS True, False
  . . .
ENDNEWTYPE Boolean;
```

- b) la sorte naturelle peut être définie à l'aide d'un littéral, le zéro. Les autres valeurs peuvent être générées au moyen de ce littéral et d'opérateurs;
- c) les valeurs de la sorte couleur\$\$-primaire peuvent être définies de la même manière que les littéraux booléens;

```

NEWTYPE Primary$$-colour
  LITERALS Red, Green, Blue
  ...
ENDNEWTYPE Primary$$-colour;

```

d) au § D.6.1.3.2, le troisième exemple c) présente une définition de type partielle sans littéraux.

#### D.6.1.3.2 *Opérateurs*

Un opérateur est une fonction mathématique qui met en concordance une ou plusieurs valeurs (éventuellement de sortes différentes) avec une valeur résultante. Les opérateurs sont introduits après le mot-clé OPERATORS, la ou les sortes de leur ou leurs arguments et la sorte de résultat sont également indiquées (on parle de signature des opérateurs).

*Exemples:*

a) dans le type booléen, un opérateur appelé «not» peut être défini; il aura un argument de sorte booléen et également un résultat de sorte booléen. Dans la définition du type booléen, il se présente comme suit:

```

NEWTYPE Boolean
  LITERALS True, False
  OPERATORS «Not»: Boolean->Boolean;
  ...
ENDNEWTYPE Boolean;

```

b) l'opérateur susmentionné nécessaire pour construire tous les nombres naturels est Next. Cet opérateur prend un argument de sorte naturelle et donne une valeur naturelle (la valeur supérieure suivante) en tant que résultat;

c) il est possible de définir pour des couleurs un nouveau type qui n'a pas de littéraux mais utilise des littéraux de la sorte couleur primaire et certains opérateurs:

```

NEWTYPE Color
  OPERATORS
    Take: Primary_colour->Colour;
    Mix: Primary_colour, Colour->Colour;
  ...
ENDNEWTYPE Colour;

```

Il s'agissait de prendre une couleur primaire et de la considérer comme une couleur puis de commencer à mélanger plusieurs couleurs primaires afin d'obtenir d'autres couleurs.

#### D.6.1.3.3 *Termes*

A l'aide de littéraux et d'opérateurs, il est possible de construire comme suit l'ensemble des termes:

- 1) Rassembler tous les littéraux dans un ensemble de la sorte dans laquelle ils sont définis - chaque littéral est un terme.
- 2) Un nouvel ensemble de termes est créé pour chaque opérateur lorsque l'opérateur est appliqué à toutes les combinaisons possibles de termes de la sorte correcte créés précédemment:
  - a) pour la sorte «Boolean», l'ensemble de littéraux est {True (vrai), False (faux)}. Le résultat de cette étape est {Not (True), Not (False)} parce que nous avons seulement l'opérateur Not (non);
  - b) pour la sorte «Natural», le résultat de cette étape est {Next(0)};
  - c) pour la sorte «Colour» l'ensemble de littéraux est vide mais le résultat de cette étape est {Take(Red), Take(green), Take(Blue)}.
- 3) Les termes des ensembles créés au cours de l'étape précédente sont tous la sorte du résultat de l'opérateur appliqué; par exemple, tous les résultats de l'opérateur Not sont de la sorte booléenne. Alors, on réunit tous les ensembles de la même sorte, aussi bien des ensemble initiaux que les ensembles nouvellement créés:
  - a) pour la sorte booléenne, on obtient l'ensemble {True, False, Not(True), Not(False)};
  - b) pour les nombres naturels cette étape donne l'ensemble {0, Next(0)}.

- 4) Les deux dernières étapes sont répétées à maintes reprises, et définissent en général un ensemble infini de termes:
- l'ensemble de termes booléens générés par les littéraux True et False et l'opérateur Not est {True, False, Not(True), Not(False), Not(Not(True)), Not(Not(False)), Not(Not(Not(True))), . . . }
  - l'ensemble des termes naturels générés par le littéral 0 et l'opérateur Next est {0, Next(0), Next(Next(0)), Next(Next(Next(0))), . . . }.
  - l'ensemble des termes de couleur générés par les littéraux Red, Green et Blue de la sorte Primary\_colour et les opérateurs Take et Mix est {Take(Red), Take(Green), Take(Blue), Mix(Red, Take(Red)), Mix(Red, Take(Green)), Mix(Red, Take(Blue)), Mix(Green, Take(Red)), Mix(Green, Take(Green)), Mix(Green, Take(Blue)), Mix(Blue, Take(Red)), Mix(Blue, Take(Green)), Mix(Blue, Take(Blue)), . . . }.

#### D.6.1.4 Equations et axiomes

D'une manière générale, le nombre de termes générés au paragraphe précédent est supérieur au nombre de valeurs de la sorte. A titre d'exemple, il existe deux valeurs booléennes mais l'ensemble de termes booléens a un nombre infini d'éléments. Il existe cependant une possibilité d'établir des règles spécifiant quels sont les termes considérés comme désignant la même valeur. Ces règles sont appelées équations et font l'objet du paragraphe suivant. Deux types particuliers d'équations, les axiomes et les équations conditionnelles, font l'objet des § D.6.1.4.2 et D.6.1.4.3.

Les équations, les axiomes et les équations conditionnelles sont donnés, dans la syntaxe concrète après le mot-clé AXIOMS. Ce mot-clé a été conservé pour des raisons d'ordre historique.

##### D.6.1.4.1 Equations

Une équation indique quels sont les termes considérés comme désignant la même valeur. Une équation relie deux termes séparés par le symbole d'équivalence ==.

Par exemple «Not(True) == False» indique que les termes Not(True) et False sont équivalents; chaque fois que l'on trouve Not(True), on peut le remplacer par False sans changement de sens et vice versa.

Dans certaines équations, l'ensemble de termes est divisé en sous-ensembles discontinus de termes qui désignent la même valeur. Ces sous-ensembles sont appelés classes d'équivalence. Dans le langage courant, les classes d'équivalence sont identifiées aux valeurs.

*Exemples:*

- a) L'ensemble de termes de la sorte booléenne est divisé en deux classes d'équivalence de termes par les deux axiomes suivants:

$$\text{Not(True)} \quad == \text{False};$$

$$\text{Not(False)} \quad == \text{True};$$

Les classes d'équivalence qui en résultent sont:

$$\{\text{True}, \text{Not(False)}, \text{Not(Not(True))}, \text{Not(Not(Not(False)))}, \\ \text{Not(Not(Not(Not(True))))}, \text{Not(Not(Not(Not(Not(False))))}, \dots\}$$

et

$$\{\text{False}, \text{Not(True)}, \text{Not(Not(False))}, \text{Not(Not(Not(True)))}, \\ \text{Not(Not(Not(Not(False))))}, \text{Not(Not(Not(Not(Not(True))))}, \dots\}$$

Ces deux classes d'équivalence sont identifiées aux valeurs True (Vrai) et False (Faux);

- b) Dans le cas de couleurs, on peut désirer spécifier que le mélange d'une couleur primaire avec une couleur qui contient cette couleur primaire ne fait pas de différence. De plus, l'ordre dans lequel les primaires sont mélangés est sans importance. Cela peut être énoncé dans les équations suivantes:

$$\text{Mix(Red, Take(Red))} \quad == \text{Take(Red)};$$

$$\text{Mix(Red, Mix(Red, Take(Green)))} \quad == \text{Mix(Red, Take(Green))};$$

$$\text{Mix(Red, Mix(Red, Take(Blue)))} \quad == \text{Mix(Red, Take(Blue))};$$

$$\text{Mix(Red, Mix(Green, Take(Red)))} \quad == \text{Mix(Green, Take(Red))};$$

$$\text{Mix(Red, Mix(Blue, Take(Red)))} \quad == \text{Mix(Blue, Take(Red))}; \text{ etc.}$$

Cela demande beaucoup de travail car des équations similaires apparaissent pour toutes les permutations de Red, Green et Blue. C'est pourquoi le LDS comporte la construction FOR-ALL qui introduit les noms de valeur représentant une classe d'équivalence arbitraire (ou la valeur associée à cette classe d'équivalence). Cela peut être très utile dans la situation ci-dessus; toutes les équations susmentionnées et celles qui sont indiquées par etc. peuvent être décrites en quelques lignes sous la forme suivante:

```

FOR ALL p1, p2 IN Primary_colour
/*1*/ Mix(p1, Take(p1)) == Take(p1);
/*2*/ Mix(p1, Mix(p1, Take(p2))) == Mix(p1, Take(p2));
/*3*/ Mix(p1, Mix(p2, Take(p1))) == Mix(p2, Take(p1));
/*4*/ Mix(p1, Take(p2)) == Mix(p2, Take(p1));
    FOR ALL c IN Colour
/*5*/ ( Mix(p1, Mix(p2, c)) == Mix(p2, Mix(p1, c)));
/*6*/ ( Mix(p1, Mix(p2, c)) == Mix(Mix(p1, Take(p2)) c)
    )

```

Dans les équations ci-dessus, il y a chevauchement mais cela ne pose pas de problèmes pour autant que les équations ne se contredisent pas mutuellement.

Les équations susmentionnées créent 7 classes d'équivalence dans l'ensemble de termes de la sorte Colour, de sorte qu'avec ces équations, il y a sept valeurs de couleurs. Les termes suivants sont dans les classes d'équivalence différentes:

```

Take(Red), Take(Green), Take(Blue),
Take(Red, Take(Green)),
Mix(Green, Take(Blue)),
Mix(Blue, Take(Red)),
Mix(Blue, Mix(Green, Take(Red))).

```

Tous les autres termes de la sorte Colour sont équivalents à l'un des termes ci-dessus.

Dans les exemples d'équations comportant la construction FOR-ALL, appelées équations explicitement quantifiées, l'information que p1 et p2 sont des identificateurs de valeur de sorte Primary\_colour est redondante; l'argument de l'opérateur Take et le premier argument de l'opérateur Mix ne peuvent être de la sorte Primary\_colour. En général, les équations explicitement quantifiées deviennent plus lisibles mais il est possible d'omettre la quantification si la sorte des identificateurs de valeur peut être déduite du contexte. Dans ce cas, on dit que l'équation est implicitement quantifiée.

*Exemple:*

Les équations 4 et 5 ci-dessus sont les mêmes que:

```

Mix(p1, Take(p2)) , c)) == Mix(p2, Take(p1));
Mix(p1, Mix(p2, c)) == Mix(p2, Mix(p1, c));

```

#### D.6.1.4.2 Axiomes

Les axiomes sont simplement des espèces particulières d'équation, introduites parce que dans des situations pratiques, de nombreuses équations se rapportent à des booléens. Dans ce cas, les équations tendent à prendre la forme ... == True (vrai), c'est-à-dire qu'elles indiquent qu'un terme donné est équivalent à True.

*Exemple:*

Admettons qu'un opérateur soit défini pour une couleur type: Contains: Colour, Primary\_colour-> Boolean; ce qui doit donner la réponse True (vrai) si la couleur primaire est contenue dans la couleur et False (faux) dans le cas contraire. Voici un exemple des équations dont il s'agit:

```

FOR ALL p IN Primary_colour
( Contains(Take(p), p) == True;
FOR ALL c IN Colour

```

```
( Contains(Mix(p,c), p)    == True)
)
```

La partie «== True» de ces équations peut être omise et les résultats sont appelés axiomes. Des axiomes peuvent se reconnaître à l'absence du symbole d'équivalence ==; ils désignent des termes qui sont équivalents à la valeur True (vrai) de la sorte booléenne.

La construction de la seconde équation peut paraître quelque peu forcée. Une meilleure manière d'écrire ces équations est indiquée après l'introduction de certaines constructions utiles.

#### D.6.1.4.3 Equations conditionnelles

Les équations conditionnelles constituent un moyen d'écrire des équations qui ne sont valables que dans certaines conditions. Ces conditions sont désignées par la même syntaxe que les équations non conditionnelles et sont séparées par un symbole ==> de l'équation qui est valable si la condition est remplie.

*Exemple:*

L'exemple type d'une équation conditionnelle est la définition de la division en type réel où:

```
FOR ALL x, z IN Real
( z/= 0 == True ==> (x/z) * z == x)
```

indique que, si la condition «z n'est pas égale à 0» est valable, la division par z suivie de la multiplication par z donne la valeur originale. Cette équation conditionnelle n'indique rien quant à ce qui devrait arriver si une valeur de la sorte Real était divisée par 0. Si l'on veut spécifier ce qui se passe en cas de division par zéro, il faudrait créer une équation conditionnelle de la forme suivante:

```
FOR ALL x, z IN Real
( z = 0 == True ==> (x/z) * z == . . .).
```

En pareil cas, cependant, il est recommandé de placer un «terme conditionnel» du côté droit, pour faciliter la lecture. Dans le cas ci-dessus l'équation deviendrait:

```
FOR ALL x, z IN Real
( (x/z) * z == IF z/=0
                THEN x
                ELSE . . .
                FI
)
```

#### D.6.1.5 Informations complémentaires concernant les équations et les axiomes

Les deux sections qui suivent traitent de certaines difficultés que l'on peut rencontrer lorsque des opérateurs donnent des résultats appartenant à une sorte déjà définie. Le § D.6.1.5.3 explique la notion d'erreur en tant que terme d'une équation.

##### D.6.1.5.1 Cohérence de la hiérarchie

En un point quelconque d'une spécification en LDS, il existe une seule et unique définition de type de données. Cette définition de type de données contient les sortes, opérateurs et équations prédéfinis et l'ensemble des sortes, opérateurs et équations définis par l'utilisateur dans les définitions de type partielles visibles en ce point. (C'est la raison pour laquelle un texte NEWTYPE . . . ENDNEWTYPE est appelé définition de type *partielle*.)

Il en résulte certaines conséquences pour les définitions de type aux niveaux inférieurs. Cette influence sur le type pourrait être peu souhaitable. A titre d'exemple, on pourrait spécifier de façon erronée que deux termes sont équivalents, les rendant ainsi équivalents alors qu'ils ne sont pas dans une portée englobante.

Il n'est pas admis de donner des équations telles que:

- a) les valeurs d'une sorte qui sont différentes dans une portée à un niveau supérieur soient rendues équivalentes;
- b) de nouvelles valeurs soient ajoutées à une sorte définie dans une portée de niveau supérieur.

Cela signifie par exemple que, dans un bloc au niveau du système, des définitions de type partielles spécifiées par l'utilisateur contenant un opérateur ayant un résultat prédéfini doivent rapporter tous les termes produits par cet opérateur à des valeurs de cette sorte de résultat.

*Exemples:*

- a) Si, pour une raison quelconque, on donne l'axiome:

FOR ALL n, m IN Integer  
 ((Fact(n) = Fact(m)) => (n = m))

afin de spécifier que si les résultats de l'opérateur Fact sont les mêmes, les arguments sont alors les mêmes. (A noter que => est l'implication booléenne; cela a peu de relation avec le signe d'équation conditionnelle ==>). Ainsi, par accident, les valeurs sont unifiées. Des équations des exemples précédents on peut déduire que Fact(0) = Fact(1) et cette dernière équation indique que 0 et 1 sont des notations différentes de la même valeur. Sur la base de cette dernière équation, on peut prouver que les nombres d'éléments de la sorte Integer sont réduits à un.

Avec l'aide d'une équation conditionnelle, on peut indiquer que, pourvu que n et m ne soient pas égaux à 0, le même résultat de l'opérateur Fact sur n et m implique n = m. En LDS:

FOR ALL n, m IN Integer  
 (n/=0, m/=0 ==> (Fact(n) = Fact(m)) => (n = m))

A noter que cette dernière équation n'ajoute rien à la sémantique des nombres entiers; c'est un théorème qui peut être déduit des autres équations. Par ailleurs, l'adjonction d'une équation prouvable ne présente pas d'inconvénient.

- b) Admettons que l'on découvre la nécessité d'un opérateur pour les factorielles lors de la spécification d'un type donné. Dans la définition de type partielle de ce type, l'opérateur Fact est introduit:

Fact: Integer -> Integer;

et les équations suivantes sont données pour définir cet opérateur:

Fact(0) == 1;  
 FOR ALL n IN Integer  
 (n > 0 ==> Fact(n) == n \* Fact(n-1))

Ces équations ne définissent pas Fact(-1) et ainsi, c'est un terme de la sorte Integer qui n'a pas de relation avec d'autres termes de cette sorte. En conséquence, Fact(-1) est une nouvelle valeur de la sorte entier (et il en va de même pour Fact(-2), Fact(-3), etc). Cela n'est pas admis. L'exemple b) du § D.6.1.5.3 donne une définition correcte de fact.

#### D.6.1.5.2 Egalité et inégalité

Dans chaque type, les opérateurs d'égalité et d'inégalité sont implicites. Ainsi, si une définition de type partielle introduit une sorte S, il y a alors des définitions implicites d'opérateur:

"=": S, S -> Boolean;  
 "/=": S, S -> Boolean;

(Remarque – Les guillemets spécifient que = et/= sont utilisés comme opérateurs infixes.)

L'opérateur d'égalité présente les propriétés prévisibles:

- a = a,
- a = b => b = a,
- a = b AND b = c => a = c,
- a = b => a == b,
- a = b => op(a) = op(b) for all operators op.

Ces propriétés ne sont pas écrites en syntaxe LDS et ne doivent pas être énoncées dans des axiomes ou des équations car elles sont implicites. La valeur booléenne obtenue lorsque cet opérateur est appliqué est True (vrai) si les termes de la partie gauche et de la partie droite sont de même classe d'équivalence; sinon la valeur obtenue est False (faux). S'il n'est pas explicitement spécifié, que la valeur est True (vrai) ou False (faux), la spécification est incomplète.

Pour l'opérateur d'inégalité, c'est par une équation en LDS que l'on peut le mieux expliquer la sémantique:

```
FOR ALL a, b IN S
( a/= b == Not(a = b))
```

Il n'y a pas de différence entre l'égalité et l'équivalence. Deux termes qui sont équivalents désignent la même valeur et l'opérateur d'égalité entre eux donne le résultat True (vrai).

#### D.6.1.5.3 Erreur

On a jugé nécessaire, pour les exemples qui précèdent, de spécifier que l'application de l'opérateur à certaines valeurs est considérée comme une erreur. Le LDS a un moyen de le spécifier formellement: l'élément ERROR. L'erreur devrait servir à exprimer:

«l'application de cet opérateur à cette valeur n'est pas admise et lorsqu'il se présente, le comportement futur est indéfini».

Dans la syntaxe concrète, cela est indiqué par le terme Error!, qui ne peut être utilisé comme argument d'un opérateur.

Lorsque Error résulte de l'application d'un opérateur et que cette application est un argument d'un autre opérateur, l'application d'opérateur extérieur porte également Error dans son résultat (propagation-d'erreurs). Dans un terme conditionnel, la partie THEN ou la partie ELSE est évaluée, de sorte que l'une d'elle peut être une erreur sans que celle-ci soit évaluée (étant donnée que l'autre partie de l'alternative est évaluée).

*Exemples:*

a) Dans l'exemple de division de valeurs de sorte Real, les poids peuvent être remplacés comme suit:

```
FOR ALL x, z IN Real
    IF z/=0
    THEN    x
    ELSE    Error!
    FI
)
```

Pour plus de clarté, on pourrait ajouter:

```
FOR ALL x IN Real
( x/0 == Error!)
```

b) Dans l'exemple comportant l'opérateur Fact, on pourrait spécifier que l'application de cet opérateur sur des entiers négatifs est considérée comme une erreur (Error). Cela permet d'éviter que Fact(-1), Fact(-2), . . . ne deviennent de nouvelles valeurs de la sorte Integer (entier). Il conviendrait de définir l'opérateur Fact comme suit:

```
n < 0 ==> Fact(n) == Error!;
          Fact(0) == 1;
n > 0 ==> Fact(n) == Fact(n-1) * n;
```

Ces trois lignes sont beaucoup plus claires que l'équation du style programmation indiquée ci-après. En général, le terme conditionnel devrait être utilisé s'il y a deux cas complémentaires; l'emboîtement de termes conditionnels rend l'équation illisible, comme on peut le voir ci-dessous:

```
Fact(n) == IF n > 0
            THEN Fact(n-1) * n
            ELSE IF n = 0
            ELSE THEN 1
            ELSE ELSE Error!
            ELSE FI
            FI
```



## D.6.2 Générateurs et héritage

Le présent paragraphe traite de deux constructions qui peuvent être utilisées pour spécifier des types ayant des parties communes. Le générateur spécifie non un type mais un schéma, qui devient un type lorsque les sortes, opérateurs, littéraux et constantes formels sont remplacés par des termes réels.

L'héritage offre la possibilité de créer un nouveau type en partant d'un type déjà existant. Les noms de littéraux et d'opérateurs peuvent être renommés et il est possible de spécifier des littéraux, des opérateurs et des équations supplémentaires.

### D.6.2.1 Générateurs

Une définition de générateur définit un schéma paramétré par des noms formels de sortes, de littéraux, de constantes et d'opérateurs. Les générateurs sont destinés à des types qui représentent des «variations sur un thème», tels que des ensembles d'éléments, des chaînes d'éléments, des fichiers d'enregistrement, des tables de consultation, des tableaux.

On peut l'expliquer à l'aide d'un exemple pour lequel des variations peuvent être envisagées. Admettons qu'il soit nécessaire qu'un type ressemble à la notion mathématique d'un ensemble d'entiers. La partie suivante du texte fait partie de la définition de type de cet ensemble d'entiers.

```
NEWTYPE Int__set
  LITERALS empty__int__set
  OPERATORS
    Add : Int__set, Integer -> Int__set;
    Delete : Int__set, Integer -> Int__set;
    Is_in : Int__set, Integer -> Boolean
  AXIOMS
/* 1 */  Delete(empty__int__set, int)
         == empty__int__set;
/* 2 */  Is_in(set,int1) = false == >
         Delete(Add(set, int1), int2)
         == IF int1 = int2
            THEN set
            ELSE Add(Delete(set, int2), int1)
            FI;
/* 3 */  Is_in(empty__int__set, int)
         == False;
/* 4 */  Is_in(Add(set, int1), int2)
         == int1 = int2 OR Is_in(set, int2);
/* 5 */  Add(Add(set, int1), int2)
         == IF int1 = int2
            THEN Add(set, int1)
            ELSE Add(Add(set, int2), int1)
            FI
ENDNEWTYPE Int__set;
```

FIGURE D-6.2.1

#### Int\_\_set de Newtypes

Toutes les équations ont une quantification implicite. La première équation indique que la suppression d'un élément de l'ensemble vide donne l'ensemble vide pour résultat. La seconde équation indique que la suppression après insertion du même élément donne pour résultat l'ensemble tel qu'il était avant l'insertion (à condition que l'ensemble ne contienne pas l'élément); sinon l'ordre d'insertion et de suppression peut être interchangeable. La troisième équation indique que l'élément vide ne contient pas d'éléments. La quatrième équation indique qu'un élément se trouve dans un ensemble s'il est le dernier élément ajouté ou s'il se trouvait dans l'ensemble avant l'adjonction du dernier élément. La dernière équation indique que l'ordre d'addition des éléments ne fait aucune différence.

Dans l'exemple de la figure D-6.2.1, Int\_\_set (ensemble\_d'entiers) n'est qu'un exemple d'un ensemble et si l'on a également besoin d'un PID\_\_set, d'un Subscriber\_\_set (ensemble\_d'abonnés) et d'un Exchange\_name\_\_set (ensemble\_de\_noms\_de\_central) dans la spécification, personne ne sera surpris qu'il contienne tous les opérateurs Add

(ajouter), Delete (supprimer) et Is\_in (est\_dans) et un littéral pour l'ensemble vide. Les équations données pour ces opérateurs facilitent la généralisation à d'autres ensembles.

Tel est le cas où la notion de générateur se révèle utile; le texte commun peut être donné une fois et être utilisé plusieurs fois. La figure D-6.2.2 présente le générateur. (A noter que les noms de sortes formels sont introduits par le mot clé TYPE, cela uniquement pour des raisons d'ordre historique.)

Au lieu d'utiliser Integer (entier), on utilise le type formel Item et, pour pouvoir donner différents noms à l'ensemble entier vide et aux ensembles vides dans d'autres types, on fait également de ce littéral un paramètre formel.

```

GENERATOR Set (TYPE Item, LITERAL empty_set)
LITERALS empty_set
OPERATORS
  Add : Set, Item -> Set;
  Delete : Set, Item -> Set;
  Is_in : Set, Item -> Boolean
AXIOMS
/* 1 */   Delete(empty_set, itm)
          == empty_set;
/* 2 */   Is_in(st,itm1) = false == >
          Delete(Add(st, itm1), itm2)
          == IF itm1 = itm2
             THEN st
             ELSE Add(Delete(st, itm2), itm1)
             FI;
/* 3 */   Is_in(empty_set, itm)
          == False;
/* 4 */   Is_in(Add(st, itm1), itm2)
          == itm1 = itm2 OR Is_in(st, itm2);
/* 5 */   Add(Add(st, itm1), itm2)
          == IF itm1 = itm2
             THEN Add(st, itm1)
             ELSE Add(Add(st, itm2), itm1)
             FI
ENDGENERATOR Set;

```

FIGURE D-6.2.2  
Générateur "SET"

Avec ce générateur, le type Int\_set peut être construit comme suit:

```

NEWTYPE Int_set Set (Integer, empty_int_set)
ENDNEWTYPE Int_set;

```

Si l'on compare les figures D-6.2.1 et D-6.2.2, on constate que:

- GENERATOR et ENDGENERATOR sont remplacés par NEWTYPE et ENDNEWTYPE respectivement;
- les paramètres formels de générateur (le texte entre parenthèses après le nom de générateur) sont supprimés;
- Set, Item et empty\_set sont remplacés dans tout le générateur par Int\_set, Integer et empty\_int\_set, respectivement.

Ainsi, il n'y a en fait aucune différence entre cet Int\_set et celui de la figure D-6.2.1, mais . . .

- si l'on a besoin d'un ensemble de valeurs Pid, on peut créer le type à l'aide de:

```

NEWTYPE PId-set Set(PId, empty_pid_set)
ENDNEWTYPE PId_set;

```

- si l'on a besoin d'un ensemble d'abonnés, dans lequel les abonnés sont représentés par un type introduisant la sorte Subscr, l'ensemble d'abonnés peut être créé à l'aide de:

```

NEWTYPE Subscr_set Set(Subscr, empty_subscr_set)
ENDNEWTYPE Subscr_set;

```

Cela permet d'économiser du papier, de plus, le travail est facilité parce qu'il suffit de penser une fois aux ensembles et que ce travail peut être délégué à des spécialistes qualifiés sur les types de données abstraites.

*Exemple:*

Cet exemple montre un générateur utilisant une sorte, un opérateur, un littéral et une constante formels. Il décrit une rangée d'éléments ayant une longueur maximale max\_length. La sorte comprend un littéral désignant la rangée vide et les opérateurs pour insertion et suppression d'éléments dans une rangée ou de celle-ci, l'enchaînement de rangées, le choix d'une sous-rangée et la détermination de la longueur d'une rangée. Ce dernier opérateur est rendu formel ce qui permet de la nommer à nouveau.

```

GENERATOR Row (TYPE Item, OPERATOR Length, LITERAL Empty,
    CONSTANT max_length)
LITERALS Empty
OPERATORS
    Length: Row      -> Integer;
    Insert: Row, Item, Integer  -> Row;
    Delete: Row, Integer, Integer  -> Row;
    "///": Row, Row  -> Row;
    Select: Row, Integer, Integer-> Row
    /* and other operators relevant for rows of items */
AXIOMS
    /* The equations for the operators above, among *
    /* which the following two (or equivalents) */
    Length(r) = max_length ==> Insert(r, itm, int) == Error!;
    Length(r1) + Length(r2) > max_length ==> r1//r2 == Error!
ENDGENERATOR Row;

```

A noter que l'opérateur formel Length (longueur) et le littéral Empty (vide) sont donnés une fois de plus dans le corps du générateur parce qu'ils sont renommés lors de leur instantiation. Dans le cas de l'opérateur, les arguments et la sorte de résultat ne sont donnés que dans le corps. Le générateur Row (rangée) peut servir à faire des lignes, des pages et des livres, comme suit:

```

NEWTYPE Line Row(Character, Width, Empty_line, 80)
ENDNEWTYPE Line;

NEWTYPE Page Row(Line, Length, Empty_page, 66)
ENDNEWTYPE Page;

NEWTYPE Book Row(Page, Nrf_pages, Empty_book, 10000)
ENDNEWTYPE Book;

```

#### D.6.2.2 Héritage

L'héritage constitue un moyen d'établir toutes les valeurs de la sorte dite parente, certains ou tous les opérateurs du type parent et toutes les équations du type parent. Pour les littéraux et les opérateurs, il existe une possibilité de les nommer à nouveau. En général, c'est une méthode satisfaisante parce que, dans ce cas, le lecteur peut déduire du contexte qu'il s'agit d'un autre type, même si les littéraux sont les mêmes.

Si un opérateur n'est pas hérité, on lui attribue systématiquement un autre nom inaccessible à l'utilisateur. Le fait que les opérateurs sont encore présents signifie que toutes les équations du type parent sont encore présentes (avec des opérateurs portant un autre nom). Cela garantit que les valeurs parentes sont héritées.

Avec la possibilité d'empêcher l'utilisation d'un opérateur (lorsqu'il n'est pas hérité), on assure la possibilité d'ajouter de nouveaux opérateurs. Après le mot clé ADDING, on peut donner des littéraux, des opérateurs et des équations comme dans un type ordinaire. Toutefois, il faut faire attention aux nouveaux littéraux et aux confusions possibles entre opérateurs hérités et ajoutés.

Lorsque les littéraux sont ajoutés, le résultat des opérateurs hérités, appliqués à des nouveaux littéraux, doit être défini (par des équations). Lorsque des opérateurs sont ajoutés, il ne faut pas oublier les opérateurs nommés à nouveau de manière invisible et les équations associées. Les équations de définition des opérateurs ajoutés devraient être compatibles avec les équations comportant des opérateurs hérités et non hérités.

Après cette liste d'avertissements, prenons quelques exemples:

- a) Supposons que le newtype couleur est complet et disponible. Ce type est fondé sur le choix et le mélange de faisceaux de lumière de couleur primaire. Il faudrait de longues réflexions et un long texte et/ou copie pour définir quelque chose de semblable pour le choix et le mélange de peinture.

Une solution commode à ce problème consiste à faire du newtype Colour (couleur) un générateur en effectuant uniquement deux remplacements:

- 1) la première ligne

```
NEWTYPER Colour
```

devient

```
GENERATOR Colour (TYPE Primary_colour)
```

- 2) le mot-clé ENDNEWTYPER devient ENDGENERATOR.

On peut maintenant nommer à nouveau le générateur lorsqu'il est instancié. Supposons que la sorte antérieure Primary\_colour soit appelée Light\_primary, et que la sorte Paint\_primary soit définie comme:

```
NEWTYPER Paint_primary
```

```
LITERALS Red, Yellow, Blue
```

```
ENDNEWTYPER Paint_primary;
```

Il est maintenant très facile de définir deux types similaires, un pour la lumière et un pour la peinture:

```
NEWTYPER Light_colours Colour (Light_primary) ENDNEWTYPER;
```

```
NEWTYPER Paint_colours Colour (Paint_primary) ENDNEWTYPER;
```

Il n'y a pas de problème jusqu'ici, mais comment peut-on voir la différence entre Take (Red) de Light\_colour et celui de Paint\_colour avec la même syntaxe? S'il est nécessaire de distinguer entre ces deux termes, on peut avoir recours à l'héritage. Au lieu de Light\_colours et Paint\_colours, les types Light (lumière) et Palette sont définis par héritage et le nom de l'opérateur Take (prendre) est modifié:

```
NEWTYPER Light
```

```
INHERITS Light_colours
```

```
OPERATORS (Beam=Take, Mix, Contains)
```

```
ADDING
```

```
LITERALS White
```

```
AXIOMS
```

```
White == Mix (Red, Mix (Yellow, Beam (Blue)))
```

```
ENDNEWTYPER Light;
```

Maintenant le newtype Light (lumière) a les littéraux de Light\_colours et le littéral White (blanc). Light\_colours n'a pas de littéraux qui lui soient propres (car il utilise les littéraux de Light\_primary), de sorte que White est le seul littéral de Light. Les opérateurs et les équations de Light sont les mêmes que ceux de Light\_colours, à l'exception du fait que le nom d'opérateur Take est remplacé par Beam (faisceau) et que l'équation pour White a été ajoutée. L'axiome ajouté indique que ce littéral ajouté devient un élément de l'ensemble de termes dans lequel les trois couleurs primaires sont mélangées.

Le newtype Palette a les littéraux de Paint\_colours et l'opérateur Take est remplacé par Paint (peinture):

```
NEWTYPED Palette
  INHERITS Paint_colours
  OPERATORS (Paint_Take, Mix, Contains)
ENDNEWTYPED Palette;
```

- b) Admettons que l'on veuille étendre l'ensemble de types entiers (sorte Int\_set), introduit dans la section précédente, par un opérateur qui trouve le plus petit entier de l'ensemble. Tout d'abord, il faut se demander si cet opérateur peut être introduit dans la définition de générateur pour le rendre disponible à tous les ensembles et autres choses.

S'il est vrai que cela peut être fait, cela limiterait l'élément à la définition de > et <. Cela ne convient pas à tous les éléments (Pid par exemple) et il peut être préférable de créer un newtype ayant la sorte New\_int\_set comportant un opérateur Min.

```
NEWTYPED New_int_set
  INHERITS Int_set
  OPERATORS ALL
  ADDING
  OPERATORS
    Min: New_int_set-> Integer
  AXIOMS
    Min(Empty_int_set) == Error!;
    Min(Add(Empty_int_set, x)) == x;
    Min(Add(Add(nis,x),y)) ==
      IF y < Min(Add(nis,x))
      THEN y
      ELSE Min(Add(nis,x))
    FI
ENDNEWTYPED New_int_set;
```

Etant donné que l'adjonction après une instantiation de générateur est relativement courante, le texte commençant par ADDING et se terminant juste avant le ENDNEWTYPED peut être donné à l'intérieur de l'instanciation de générateur. On en trouvera un exemple au § 5.4.1.12 de la Recommandation.

### D.6.3 *Observations relatives aux équations*

Lorsque l'on introduit un nouveau type de données, il est essentiel d'introduire suffisamment d'équations. Dans les paragraphes qui suivent, trois observations sont présentées concernant les équations qui en faciliteront l'établissement.

#### D.6.3.1 *Conditions générales*

De quelque manière que l'on construise les équations, il faut tenir compte des faits suivants:

- chaque opérateur apparaît au moins une fois dans l'ensemble des équations (sauf pour les cas «pathologiques»);
- tous les énoncés vrais peuvent être tirés des équations. Ils sont soit indiqués comme des axiomes, soit déduits par substitution de termes équivalents dans les équations;
- aucune incohérence doit être décelée, c'est-à-dire que l'on ne peut déduire des équations que Vrai = Faux.

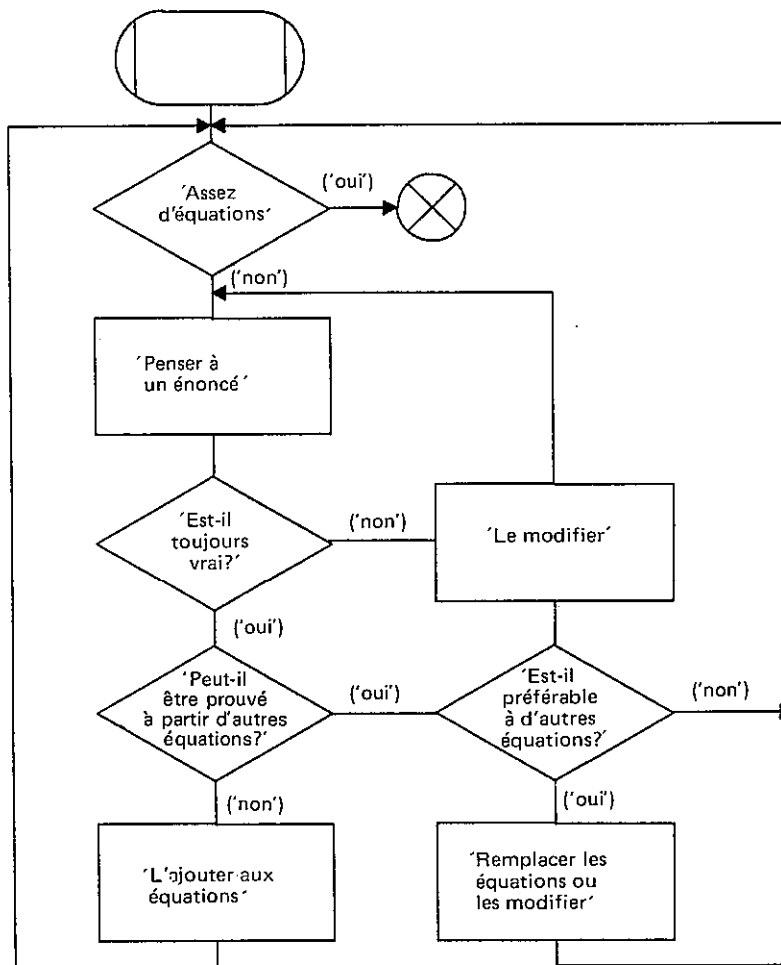
Une procédure permettant de trouver des équations peut être exprimée en LDS informel, comme indiqué dans la figure D-6.3.1.

### D.6.3.2 Application de fonctions aux constructeurs

D'une manière générale, l'ensemble d'opérateurs possède un sous-ensemble d'opérateurs appelés «constructeurs» et «fonctions». Les constructeurs peuvent servir à générer toutes les valeurs (classes d'équivalence) de la sorte. Dans cette approche, les littéraux sont considérés comme des opérateurs sans argument.

Exemples:

- le type booléen a ses littéraux pour constructeurs;
- le type naturel a le littéral 0 et l'opérateur Next comme constructeurs; un naturel quelconque peut être construit seulement avec 0 et Next;
- le générateur pour les ensembles a le littéral ensemble\$\$-vide et l'opérateur\_additif comme constructeurs; un ensemble quelconque ne peut être construit qu'en utilisant Empty\_set (ensemble\_vide) et Add (ajouter);
- le type entier peut être construit au moyen des littéraux 0 et 1, des opérateurs + et moins unaires.



T1002370-88

FIGURE D-6.3.1

Procédures permettant de trouver des équations en LDS informel

A noter qu'il y a parfois plusieurs choix possibles pour l'ensemble de constructeurs. Tout choix sera valable pour le reste de la présente section mais les petits ensembles sont généralement les meilleurs.

Ensuite, les fonctions sont traitées une par une. Pour chaque argument d'une fonction, seuls sont énumérés tous les termes possibles composés de constructeurs. Pour éviter le problème que posent les nombres infinis de termes, il faut appliquer la quantification.

*Exemples:*

- a) Pour les nombres naturels, cette liste peut être réduite à:

0

Next (n) où n est tout nombre naturel.

- b) Pour les ensembles, la liste éventuelle peut être:

empty\_set

Add (s,i) où s est tout ensemble et i tout élément (item).

Si, dans le terme de droite d'une équation ayant (s,i) du côté gauche, il y a une différence entre s étant vide ou n'étant pas vide, on peut réécrire la liste comme suit:

empty\_set

Add (empty\_set,i) où i est un élément (item) quelconque,

Add (Add (s,i),j) où s est un ensemble quelconque et i, j, un élément quelconque.

Après la création de cette liste, on obtient les parties de gauche des équations en appliquant chaque fonction à une combinaison d'arguments tirés de la liste. Les identificateurs de valeurs de différents arguments reçoivent des noms différents. La procédure indiquée ci-dessus pour les fonctions peut être appliquée aux constructeurs; dans ce cas, elle donne les relations entre des termes où les constructeurs sont utilisés dans différents ordres.

*Exemples:*

- a) Pour l'opérateur de multiplication de nombres naturels portant la signature

«\*»:Natural, Natural -> Natural

Cette procédure donne la partie de gauche des équations (incomplètes) suivantes. L'utilisateur devra ajouter la partie de droite.

(n)0\*0 == ...;

(n)0\*Next (n) == ...;

Next (n)\*0 == ...;

Next (n)\*Next (m) == ...;

- b) Pour les opérateurs Is\_in (est\_dans) et Delete (supprimer, dans le générateur Set (voir le § D.6.2.1), cette approche est déjà appliquée.

- c) Pour la sorte Colour, les constructeurs sont Take et Mix. Un opérateur analogue à Contains dans le § D.6.1.4.2 doit être défini pour les arguments.

Take(p) où p est toute couleur primaire

Mix(p,c) où p est toute couleur primaire et c une couleur quelconque.

Etant donné que l'on avait annoncé au § D.6.1.4.2 que les équations complètes seraient données pour cet opérateur:

Contains(Take(p),q) ==p=q;

Contains(Mix(p,c),q) ==(p=q) OR Contains(c,q);

Cette procédure de construction peut donner plus d'équations que cela n'est nécessaire, mais elle est très sûre. Dans l'exemple susmentionné de multiplication des nombres naturels, il est vraisemblable que la propriété de commutativité de la multiplication sera indiquée et qu'en conséquence, seule la dernière (ou la seconde) équation des trois premières sera nécessaire.

La procédure décrite dans cette section peut être appliquée en combinaison avec la procédure décrite dans la section précédente, où elle est utile pour la tâche «Penser\_à\_un\_énoncé».

### D.6.3.3 Spécification d'ensemble d'essai

On peut aussi considérer les équations du point de vue de la mise en œuvre. Si les opérateurs sont mis en œuvre sous la forme de fonctions dans un langage de programmation, les équations montrent comment ces fonctions doivent être testées.

Il convient d'évaluer les expressions correspondant à la partie de gauche d'une équation, de faire de même pour la partie droite de cette équation et de voir si elles sont équivalentes. C'est la construction FOR ALL qui pourrait poser certains problèmes. Ceux-ci peuvent souvent être résolus de manière pragmatique:

Au lieu de `tif de test` peut utiliser `FOR ALL i IN Integer`

le dispositif de test peut utiliser `FOR ALL i IN {-10,-1,0,1,10}` et procéder ainsi dans la plupart des cas.

Considérer des équations comme nécessaires à la mise en œuvre peut être utile pour ce qui est de la tâche «Penser\_à\_un\_énoncé» dans la procédure du § D.6.3.1.

#### D.6.4 *Caractéristiques*

La présente section décrit certains dispositifs du LDS qui sont rarement nécessaires dont on peut pratiquement se passer, mais qui rendent quelquefois la tâche plus facile.

##### D.6.4.1 *Opérateurs cachés*

Il arrive que l'ensemble des équations puisse être simplifié ou rendu plus lisible grâce à l'introduction d'un opérateur supplémentaire, mais cet opérateur ne devrait pas être utilisé dans les processus. Cela signifie que l'opérateur est visible de l'intérieur mais caché en dehors de la définition de type.

On peut atteindre ce résultat en définissant un «type caché», c'est-à-dire un type que l'utilisateur ne doit pas employer. A partir de ce type caché, l'utilisateur peut hériter de tous les opérateurs auxquels il peut avoir accès; c'est le type hérité qui doit être utilisé. On peut s'assurer qu'il est correctement employé en examinant toutes les déclarations de variables (aucune variable de sorte introduite par le type caché ne doit apparaître).

Ce qui caractérise les opérateurs cachés, c'est qu'ils peuvent être atteints par une restriction de leur visibilité aux seules équations. On peut le faire en plaçant un point d'exclamation après l'opérateur.

*Exemple:*

La manière courante de faire un ensemble d'un élément à partir du générateur Set est la suivante:

```
Add(empty_set,x)
```

et c'est ainsi que doit le faire chaque utilisateur. Dans les équations, le spécificateur peut utiliser un opérateur spécial, par exemple:

```
Mk_set!:Item->Set;
```

défini par l'équation:

```
Mk_set!(itm) == Add(empty_set,itm);
```

qui peut être utilisé dans des définitions de type partiel mais non dans le corps de processus en LDS.

##### D.6.4.2 *Relations d'ordre*

Lorsqu'il faut spécifier une relation d'ordre sur les éléments d'une sorte, cela signifie en général qu'il faut définir quatre opérateurs (<,<=,>,>=) et les propriétés mathématiques ordinaires (transitivité, etc.). S'il y a de nombreux littéraux, il faut également donner de nombreuses équations. Par exemple, le type de données prédéfinies Character est défini de cette manière.

Le LDS comporte une caractéristique qui permet d'abrégé ces définitions de type longues, peu lisibles et ennuyeuses: c'est l'abréviation ORDERING (relation d'ordre).

ORDERING est donné dans la liste d'opérateurs, de préférence au début ou à la fin de celle-ci. Cela permet d'introduire des opérateurs de relation d'ordre et les équations normales. Lorsque ORDERING est spécifié, il faut donner les littéraux, s'il en existe, dans l'ordre ascendant.

*Exemple:*

```
NEWTTYPE Even_decimal_digit
```

```
LITERALS 0,2,4,6,8
```

```
OPERATORS
```

```
ORDERING
```

```
ENDNEWTTYPE Even_decimal_digit;
```

Maintenant, l'ordre `0<2<4<6<8` est implicite.



Au § D.6.2.2 (Héritage), on soulignait qu'il fallait s'assurer que les littéraux étaient ajoutés à une sorte héritée. Il convient d'en indiquer ici la raison.

Admettons que l'on désire l'extension suivante de la sorte Even\_decimal\_digit:

```
NEWTYPE Decimal_digit
  INHERITS Even_decimal_digit
  OPERATORS ALL
  ADDING
  LITERALS 1,3,5,7,9
  AXIOMS
    0<1; 1<2;
    2<3; 3<4;
    4<5; 5<6;
    6<7; 7<8;
    8<9
  ENDNEWTYPE Decimal_digit;
```

Les axiomes donnés ici ne peuvent être omis. Sans ces axiomes, il ne peut y avoir qu'un ordre dit partiel:

0<2<4<6<8

et

1<3<5<7<9.

Avec les axiomes ci-dessus, on obtient un ordre complet:

0<1<2<3<4<5<6<7<8<9

mais avec l'axiome «9<0» au lieu de l'ensemble d'axiomes ci-dessus, l'ordre complet serait le suivant:

1<3<5<7<9<0<2<4<6<8.

#### D.6.4.3 Sortes avec champs

Comme indiqué au § 5.4.1.10 de la Recommandation, on peut définir une sorte structurée sans constructions spéciales, mais les sortes structurées sont à la fois courantes et utiles, ce qui justifie certaines constructions supplémentaires dans le langage.

Une sorte structurée devrait être utilisée lorsqu'une valeur d'objet est formée par l'association de valeurs de plusieurs sortes. Chaque valeur de cette association est caractérisée par un nom, appelé nom de champ. La sorte d'un champ est fixe.

*Exemples:*

```
NEWTYPE Subscriber
  STRUCT numbers Number_key;
         name Name_key;
         admin Administrative;
  ENDNEWTYPE Subscriber;
NEWTYPE Name_key
  STRUCT name,
         street Charstring;
         number Integer;
         city Charstring;
  ENDNEWTYPE Name_Key;
```

Avec une sorte structurée, certains opérateurs sont définis implicitement:

- a) l'opérateur constructeur, «(.«avant, et».)» après les valeurs de champ;
- b) les opérateurs de sélection de champ, les variables de la sorte structurée suivies par un ! et le nom de champ, ou suivies par le nom de champ entre parenthèses. Il ne faut pas confondre la variable suivie d'un ! avec l'opérateur caché (§ D.6.4.1).

On trouvera un exemple dans la figure D-6.4.1.

```

PROCESS Some_process;
DCL na, st, where Charstring,
    nu          Integer,
    nk          Name_key;
/* ... Text where values are assigned to the variables na and st */
TASK nk := (. na, st, 5, 'London' .);
/* ... Text where no assignments to nk take place */
TASK where := nk!city;
/* Now where = 'London' holds */
ENDPROCESS Some_process;

```

FIGURE D-6.4.1

**Exemple d'emploi d'opérateurs de sorte à structure implicite**

D.6.4.4 *Sortes indexées*

Une sorte indexée est une sorte pour laquelle le type a pour nom d'opérateur Extract! (extraction). Dans les types de données prédéfinies, le générateur Array est un tel type. Array est l'un des exemples les plus courants de type indexé.

Pour l'opérateur caché Extract!, il existe une syntaxe concrète spéciale qui doit être appliquée en dehors des définitions de type.

On peut penser que le type Index dans le générateur prédéfini Array doit être un type «simple» comme Integer, Natural ou Character. Toutefois, il n'y a pas de raison pour qu'une structure comme Name\_key ne puisse pas être utilisée comme Index.

*Exemple:*

```

NEWTYPENAME Subsc_data_base
    Array (Name_key, Subscriber)
ENDNEWTYPENAME Subsc_data_base;

```

Les sortes Name\_key et Subscriber sont celles qui ont été définies dans la section précédente. Supposons qu'il existe une procédure Bill comportant un paramètre de sorte Subscriber et que cette procédure soit définie dans un processus qui comporte aussi une variable Sub\_db de la sorte Subsc\_data\_base. Dans ce processus, l'appel suivant pourrait apparaître.

```
CALL Bill (Sub_db ((. 'P.M.', 'Downingstreet', 10, 'London'.)));
```

D.6.4.5 *Valeur par défaut de variables*

Comme indiqué dans la section concernant la déclaration de variables (§ D.3.10.1), il est possible d'affecter des valeurs à une variable immédiatement après la déclaration. Cependant, certains types ont une valeur qui sera (presque) toujours la valeur initiale d'une variable. Il existe une caractéristique qui permet d'éviter d'écrire la valeur initiale pour chaque déclaration: la clause DEFAULT.

A titre d'exemple, on peut considérer l'ensemble. Il est très probable que presque toutes les variables, de tout ensemble imaginaire, seront initialisées avec empty\_set.

La notation:

```
DEFAULT empty_set
```

après la liste d'équations indique que chaque variable de chaque instantiation de ce générateur sera initialisée à la valeur empty\_set de cette instantiation, sauf s'il y a une initialisation explicite (voir le § D.3.10.1.)

S'il n'est pas sûr que la valeur initiale de toutes les variables d'une sorte soit la même, il ne faut pas utiliser la clause DEFAULT, sinon il est difficile d'éviter des surprises.

#### D.6.4.6 *Opérateurs actifs*

Les utilisateurs qui connaissent la Recommandation Z.104 de 1984 concernant le LDS pourraient se demander ce qui est arrivé aux opérateurs dits actifs. En fait, cette caractéristique a été supprimée, pour les raisons suivantes:

- a) elle n'est pas nécessaire car les opérateurs courants et les procédures et/ou macros offrent la même capacité d'expression;
- b) elle compromet la lisibilité des équations;
- c) de nombreux utilisateurs ont eu des difficultés à l'utiliser correctement;
- d) elle ne s'intègre pas au modèle de type de données abstrait fondé sur les algèbres initiales qui constituent le modèle choisi comme base mathématique de cette partie du LDS.

#### D.7 *Directives supplémentaires pour le dessin et l'écriture*

La Recommandation spécifie la manière de représenter chacun des concepts du LDS. Par exemple, le concept de tâche est représenté en LDS/GR au moyen d'un rectangle et du mot-clé TASK en LDS/PR.

Les Directives pour les usagers complètent la Recommandation en spécifiant des règles de dessin et d'écriture applicables à tous les concepts, afin de souligner ce qui peut être considéré comme approprié, incorrect ou peu pratique.

##### D.7.1 *Directives pour le LDS/GR*

###### D.7.1.1 *Considérations générales*

Les directives générales applicables à l'établissement de diagrammes sont les suivantes:

- la séquence de lecture doit se présenter de haut en bas et de gauche à droite;
- les diagrammes ne doivent pas contenir trop d'informations au même niveau. Il est souvent souhaitable de subdiviser de grands diagrammes en sous-diagrammes traitant des différentes parties ou aspects, par exemple en utilisant le mécanisme de référence.

###### D.7.1.2 *Points d'entrée et de sortie*

Les points d'entrée et de sortie vers ou d'un symbole doivent être tracés verticalement (ce qui correspond à la lecture de haut en bas) et c'est seulement lorsque cela n'est pas pratique que l'on peut utiliser des points d'entrée et de sortie horizontaux.

Les exceptions à cette règle générale sont les suivants:

- les décisions comportant deux ou trois points de sortie sont généralement représentées avec deux points de sortie horizontaux (plus un point vertical);
- les appels de macro utilisent des connexions horizontales et verticales;
- les connecteurs d'entrée et de sortie ont généralement des points d'entrée et de sortie horizontaux.

Des lignes diagonales ne devraient se présenter que dans des cas exceptionnels (par exemple pour des canaux et des acheminements de signaux).

Les points d'entrée et de sortie verticaux devraient diviser le symbole en deux parties ayant la même longueur horizontale.

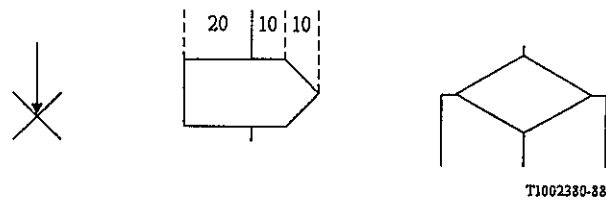


FIGURE D-7.1.1

Points d'entrée et de sortie dessinés correctement

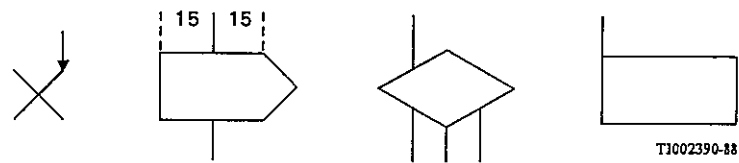


FIGURE D-7.1.2

Points d'entrée et de sortie dessinés incorrectement

### D.7.1.3 Symboles

- Les symboles devraient être tracés de telle sorte que les axes vertical et horizontal coïncident avec les deux axes du papier.
- La symétrie verticale des symboles est seulement autorisée pour les symboles d'entrée, de sortie, d'extension de texte et de commentaire (voir la figure D-7.1.3).
- Le rapport général entre la hauteur et la longueur de tous les symboles dans les graphiques ainsi que pour les symboles de référence est de 1 : 2.

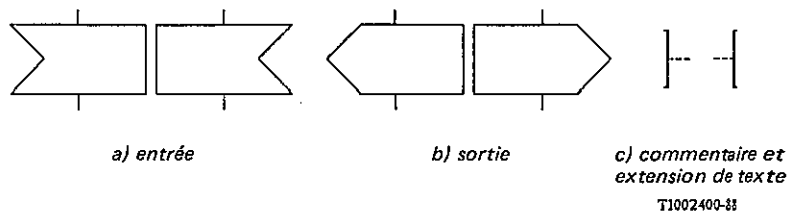
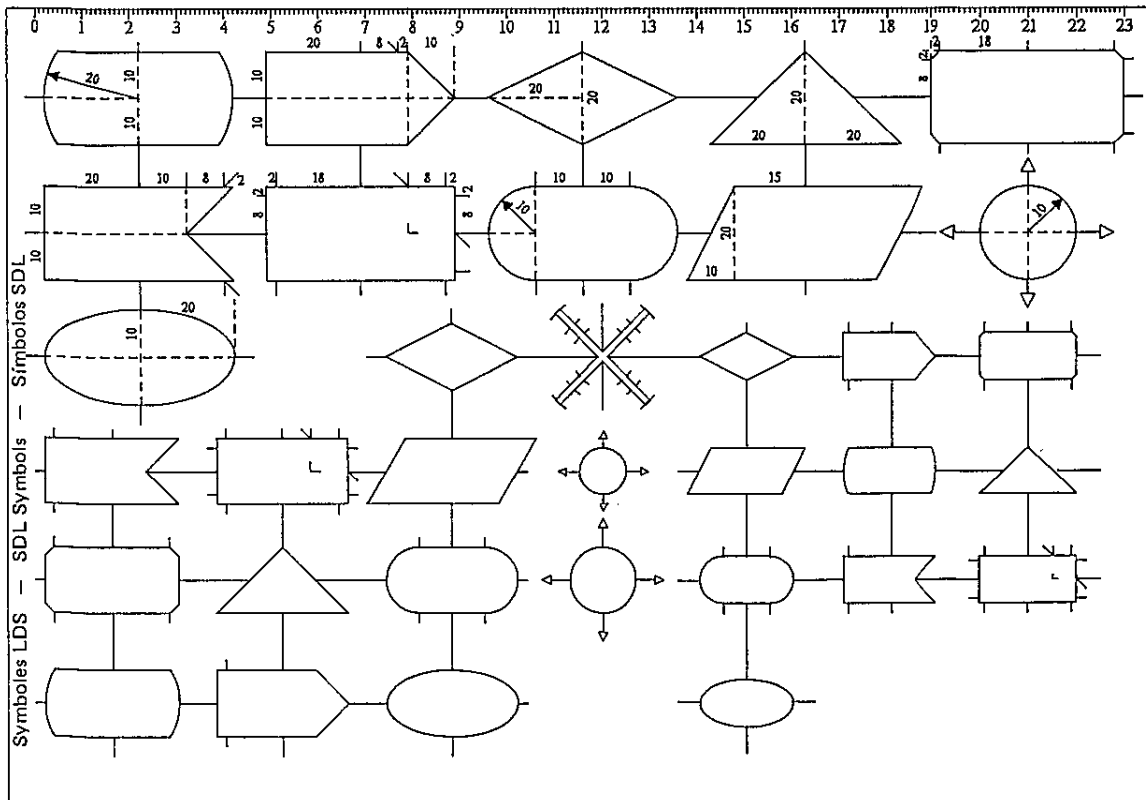


FIGURE D-7.1.3

Les quatre symboles symétriques autorisés

### D.7.1.4 Gabarit

On trouvera à l'intérieur de la couverture arrière du présent fascicule un gabarit permettant le tracé graphique des symboles du LDS/GR. Une représentation schématique de ce gabarit est donnée dans la figure D-7.1.4.



T1002410-88

FIGURE D-7.1.4

Représentation schématique du gabarit

Cette figure reproduit directement et en trois formats ( $20 \times 40$  mm,  $20/\sqrt{2} \times 40/\sqrt{2}$  et  $10 \times 20$  mm) les symboles suivants: Entrée, Sortie, Décision, Option, Processus, Début, Tâche, Etat, Mise en réserve, Référence de service, Connecteur et Arrêt. Les cotes intérieures des symboles de grand format sont indiquées.

On peut réaliser les symboles correspondant à l'Appel de Procédure, Macro et Créer à partir du symbole Tâche en traçant la(les) ligne(s) horizontale(s) ou verticale(s) supplémentaire(s) indiquée(s) dans la figure.

Il est possible de construire le symbole de début de procédure à partir du symbole de début de processus en traçant les lignes verticales supplémentaires indiquées.

Le symbole de retour est une combinaison des symboles de connecteur et d'arrêt.

Les symboles de commentaire, d'extension de texte et de liste de signaux sont tracés à l'aide du symbole Tâche.

Les symboles d'entrée et de sortie prioritaires peuvent être construits à partir des symboles d'entrée et de sortie avec l'adjonction de la ligne supplémentaire indiquée.

Le symbole de référence de procédure peut être construit à partir du symbole de référence de processus avec l'adjonction de deux traits verticaux supplémentaires comme indiqué.

Les symboles de condition de validation et de signal continu peuvent être tracés à l'aide du symbole d'arrêt.

Les canaux, les acheminements de signaux et les lignes vers le symbole d'extension de texte sont tracés en traits pleins.

La ligne vers un commentaire de symbole est tracée en trait discontinu avec un rapport 1 : 1.

Le symbole de texte est tracé à l'aide du symbole de Tâche, l'angle supérieur droit étant plié comme indiqué.

Tous les symboles recommandés sont définis dans la Recommandation. On trouvera un aperçu des symboles recommandés dans l'annexe C – Résumé de la syntaxe graphique. Les trois dimensions indiquées sont à préférer. Si d'autres dimensions sont utilisées, le rapport devrait être le même (c'est-à-dire 1 : 2). Les tailles indiquées, c'est-à-dire une longueur de 40 mm, 28 mm et 20 mm permettent la réduction photographique du format A3 au format A4 avec des tailles de symbole compatibles (car  $40 \text{ mm}/\sqrt{2} = 28 \text{ mm}$  et  $28 \text{ mm}/\sqrt{2} = 20 \text{ mm}$ ).

## D.7.2 Directives applicables au LDS/PR

Les directives générales applicables à l'écriture de LDS textuel sont les suivantes:

- la séquence de lecture devrait se présenter de haut en bas et de gauche à droite;
- le texte devrait être divisé en parties traitant de différents aspects;
- les commentaires sur le niveau des instructions devraient commencer dans la même colonne;
- les lignes devraient être en retrait. L'indentation peut suivre la hiérarchie courante des concepts du LDS comme indiqué dans l'exemple de la figure D-7.1.5.

```
SYSTEM A;  
  SIGNAL S1, S2;  
  BLOCK B;  
    PROCESS P;  
      START;  
      NEXSTATE ST1;  
      STATE F;  
        INPUT G: . . .  
        INPUT F: . . .  
        . . .  
      ENDSTATE F;  
    ENDPROCESS P;  
  ENBLOCK B;  
ENDSYSTEM A;
```

FIGURE D-7.1.5

**Exemples d'indentation textuelle en LDS**

## D.8 Documentation

### D.8.1 Introduction

L'ISO définit un document comme étant «une quantité limitée et cohérente d'informations stockées sur un support et que l'on peut rechercher». On peut donc le considérer comme une unité logique strictement délimitée. Les documents servent à véhiculer toutes les informations relatives à un système spécifié ou décrit en LDS.

Lorsque le papier sert de support physique au stockage d'un document, on applique souvent improprement le terme «document» aux feuilles de papier plutôt qu'à leur contenu logique. L'emploi des supports de stockage magnétiques se répandant, ce terme retrouve petit à petit sa signification initiale.

La présente section traite des aspects relatifs à la documentation. Elle aborde plutôt l'organisation logique des documents que leur organisation physique. Ce dernier point est laissé à la discrétion de l'utilisateur. Du fait de la ressemblance entre les conditions exigées par l'organisation logique et par l'organisation physique des documents, les conseils contenus dans le texte ci-après peuvent utilement aider un usager à mettre en place une organisation physique de documents.

En subdivisant l'information en un nombre approprié de documents, on peut rendre le système plus lisible et d'un maniement plus commode.

Le langage ne recommande pas certains documents ni des structures de document. Toutefois, certaines propositions sont présentées afin d'aider l'utilisateur à traiter les documents.

### D.8.2 Types de représentation de système

Le résultat de la spécification d'un système en LDS est un ensemble de définitions en LDS/GR et/ou LDS/PR.

Ces définitions peuvent être emboîtées ou séquentielles, selon le type de représentation de système, hiérarchique ou à plat. Dans les figures D-8.2.1 et D-8.2.2, un système est décrit avec deux variantes de représentation en LDS/GR. Les deux figures ne sont pas des spécifications complètes de système étant donné que, pour des raisons de simplicité, seules les entrées et les sorties sont indiquées et que les définitions éventuelles de signaux et de données ont été omises.

Naturellement, il est possible de recourir à une combinaison de présentation pour spécifier un système. Lorsque les définitions sont séquentielles comme dans la figure D-8.2.1, elles sont «référéncées», mécanisme possible pour les définitions aussi bien en LDS/PR qu'en LDS/GR.

Si nous considérons la spécification de système comme un ensemble de définitions, un document peut être considéré comme contenant ces définitions.

Si le système est petit et hiérarchisé comme dans la figure D-8.2.2, un seul document suffit. Si la représentation à plat est utilisée comme dans la figure D-8.2.1 il faut utiliser plus d'un document, par exemple un document par définition.

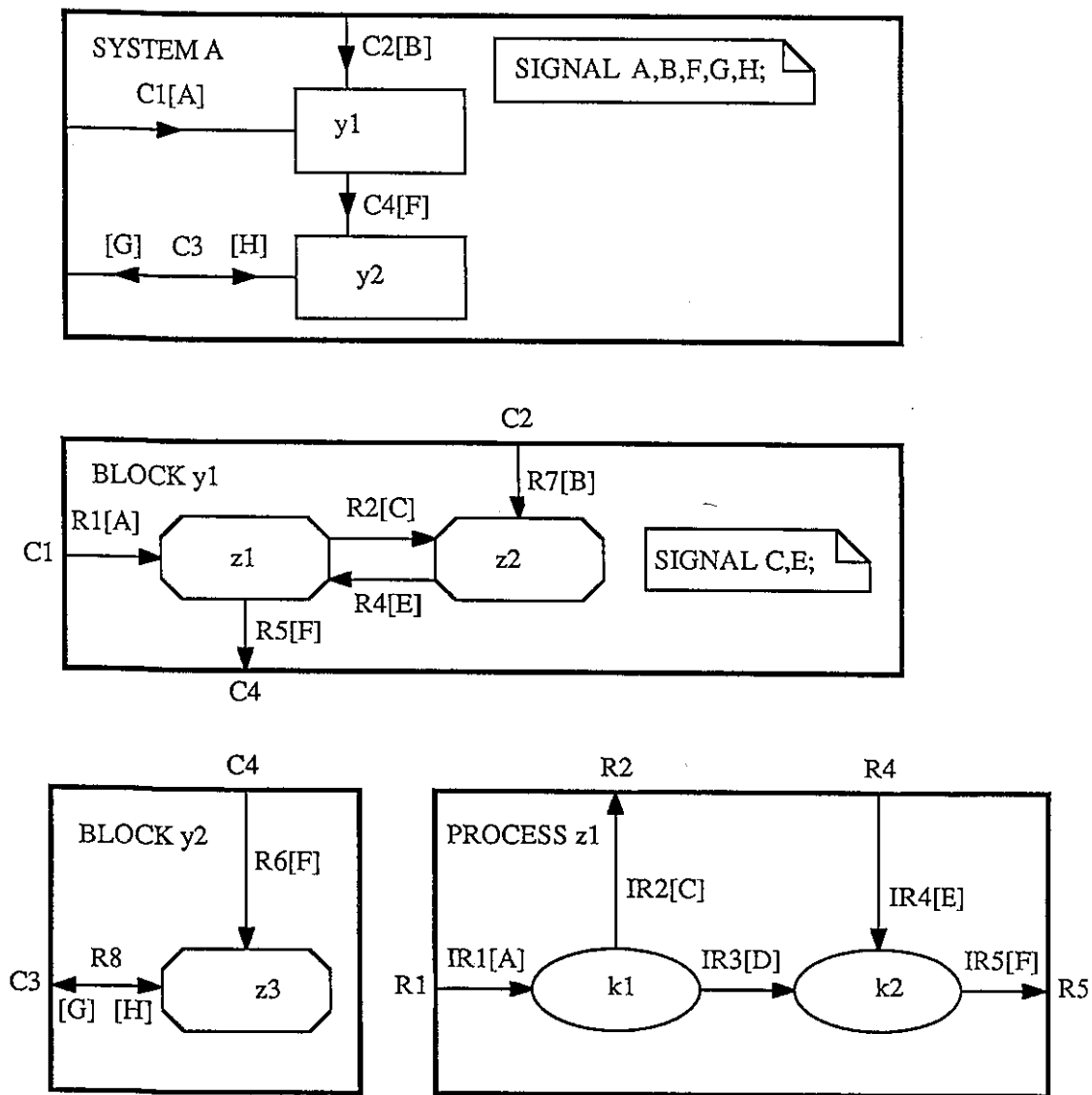
Lorsque l'on choisit le type de représentation de système, il faut considérer le type de document souhaité. Pour avoir un document par définition, il faut une représentation à plat. Si l'on désire disposer de l'ensemble de la spécification de système sur un seul document, il faut une représentation structurée.

Le cas normal est probablement une combinaison de ces représentations. Pour décider des proportions de cette combinaison, il faut appliquer les règles suivantes:

- 1) une définition ne doit pas être divisée en plusieurs documents;
- 2) si l'on veut faire figurer une définition dans un document séparé, elle doit être référencée et non emboîtée (voir la figure D-8.2.3);
- 3) lorsque l'on applique le concept logique de page pour diviser un diagramme en plusieurs pages de diagrammes, celles-ci devraient coïncider avec les pages réelles du document (voir la figure D-8.2.4);
- 4) si un diagramme a plus d'une page, il doit être référencé et non emboîté.

### D.8.3 *Structure des documents*

L'ensemble de documents traitant de toute la définition du système peut être structuré. Une structure de documents, lorsque ceux-ci se réfèrent à des sous-documents, peut être jointe à toute entité dans le système telle que sous-système, bloc ou processus (voir la figure D-8.3.1).



T1002420-88

FIGURE D-8.2.1 a)

a) Exemple de diagrammes séquentiels (référéncés) utilisés pour une représentation de système à plat



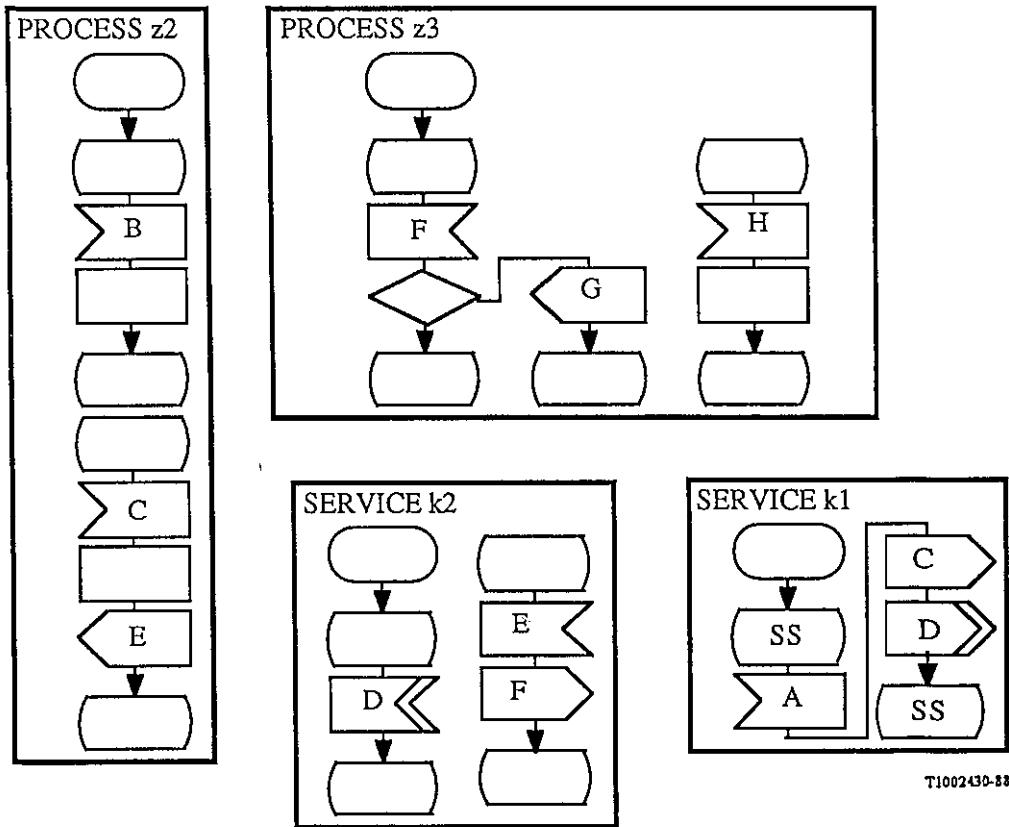
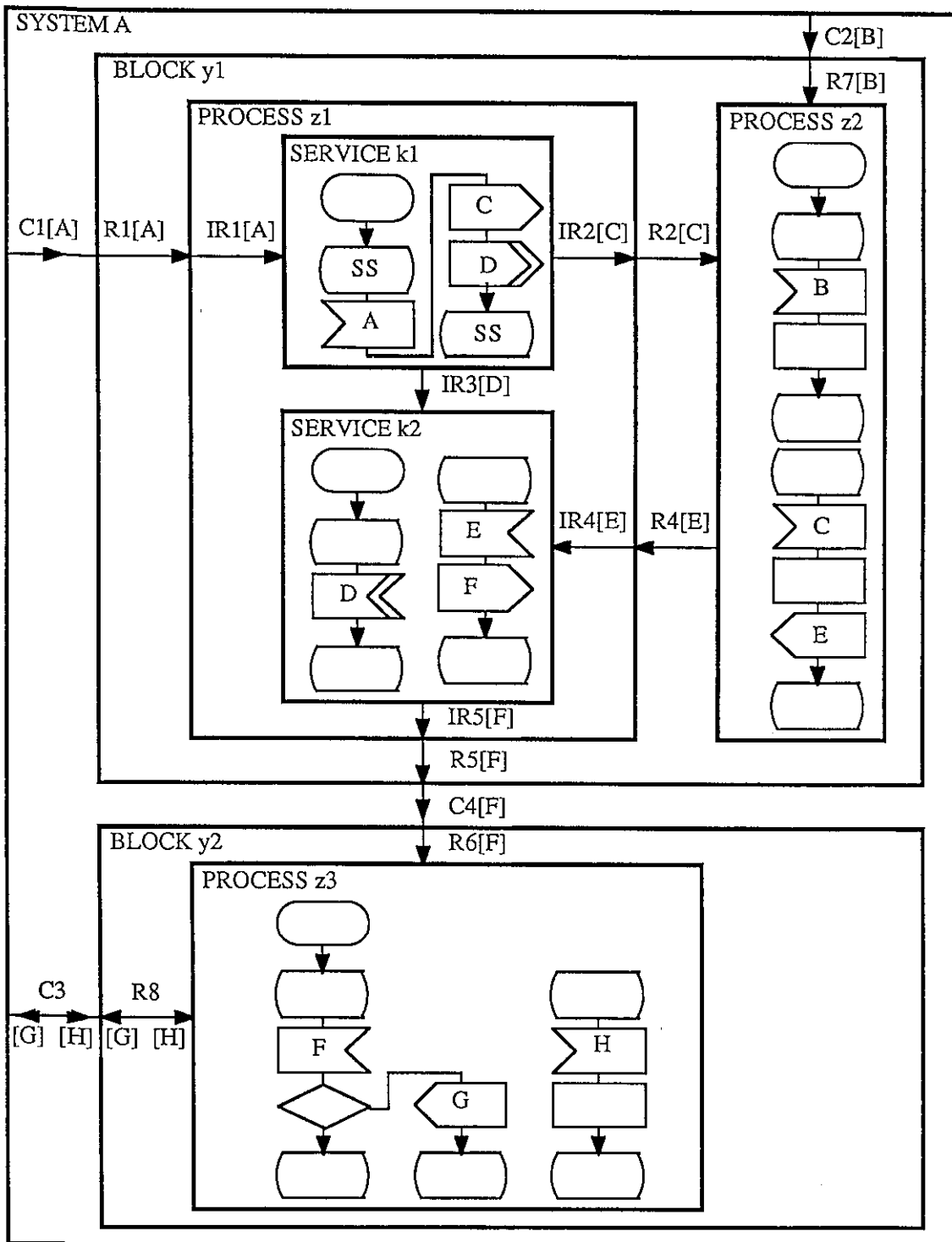


FIGURE D-8.2.1 b)

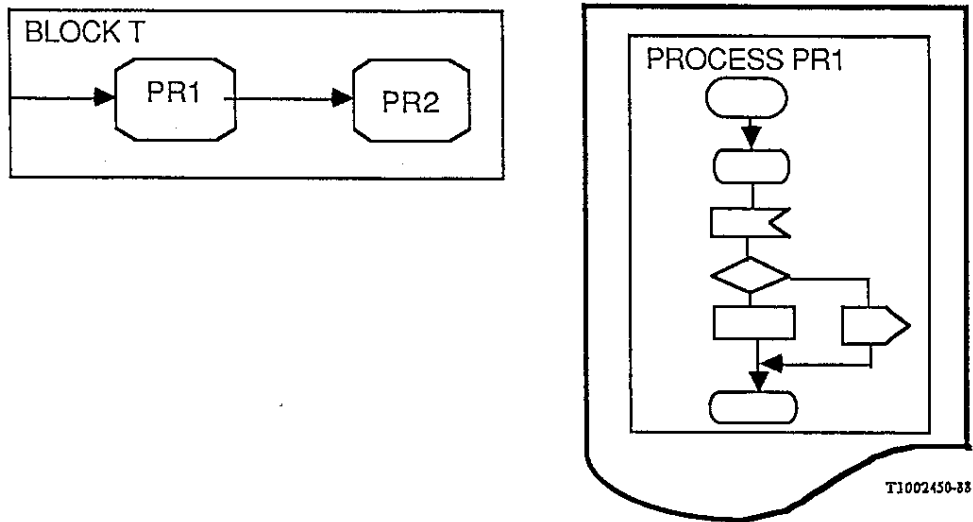
b) Exemple de diagrammes séquentiels (référencés) utilisés pour une représentation de système à plat



T1002440-88

FIGURE D-8.2.2

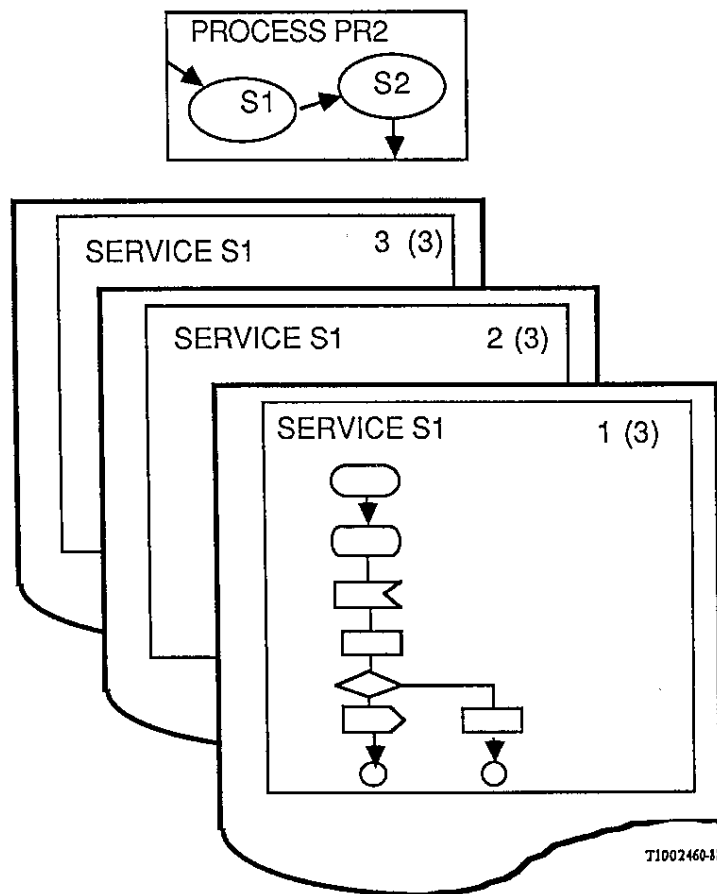
Exemple de définitions emboîtées utilisées pour la représentation hiérarchisée d'un système



T1002450-33

FIGURE D-8.2.3

Une définition référencée peut être placée dans un document séparé



T1002460-33

FIGURE D-8.2.4

Diagramme de service référencé subdivisé en trois pages

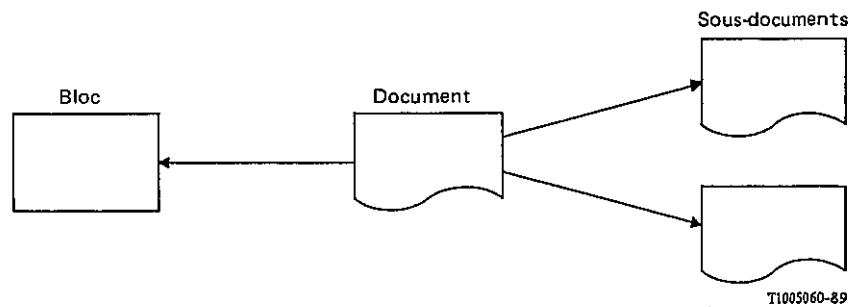


FIGURE D-8.3.1  
Structure d'un document

#### D.8.4 Mécanisme de référence

Le mécanisme de référence dans le langage, lorsque des noms de concept sont utilisés pour référence entre les concepts, peut aussi être utilisé pour des références entre documents. C'est une approche naturelle lorsque le document coïncide avec une définition.

#### D.8.5 Classification des documents

Les documents peuvent être classés conformément aux types de définitions qu'ils contiennent.

Dans cette classification, il convient de placer au moins dans des documents séparés les définitions de processus ou de service décrivant le comportement dynamique du système. Ces documents peuvent aussi comporter des définitions de variables.

On trouvera dans la figure D-8.5.1 un exemple de structure de document pour un système.

Dans cet exemple, les définitions de canal et d'acheminement de signal sont comprises dans le document pour les définitions de système, de bloc et de processus. Les définitions de système, les définitions de données et les définitions de listes de signaux sont placées dans des documents séparés et il a été admis que toutes les définitions de données se trouvent au niveau du système.

Les définitions de procédures, les définitions de macros et les définitions de service forment des sous-documents du document de processus.

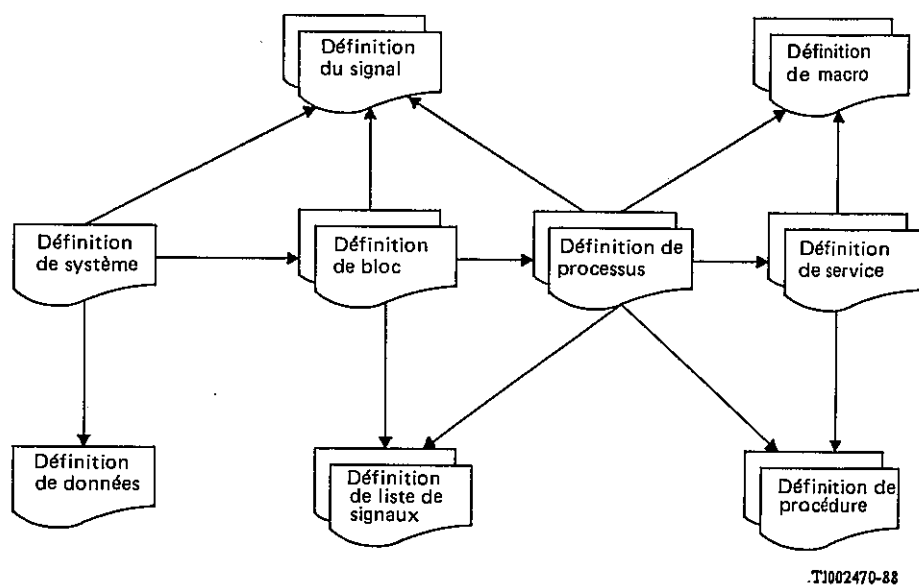


FIGURE D-8.5.1  
Exemple de structure de document pour un système

Si différents services forment ensemble une fonction de système, ils peuvent être décrits dans un document commun.

Les différentes définitions de service peuvent être placées l'une après l'autre dans un document de service mais il est également possible de les placer l'une à côté de l'autre sur la même page de document. Cette dernière présentation permet une compréhension satisfaisante de l'interaction entre les services. La figure D-8.5.2 constitue un exemple de page de document dans un document de service.

Pour de grandes spécifications de systèmes, il convient de prévoir une «table des matières» pour le système afin d'indiquer où l'on trouvera les états, les entrées, les sorties, etc. De plus, la table des matières devrait porter tous les concepts, c'est-à-dire indiquer l'emplacement des définitions et la manière dont elles sont utilisées. Cela s'applique notamment au système, aux blocs, aux canaux, aux signaux, aux processus, aux services, aux macros et aux procédures.

De telles tables des matières de système peuvent constituer des documents séparés.

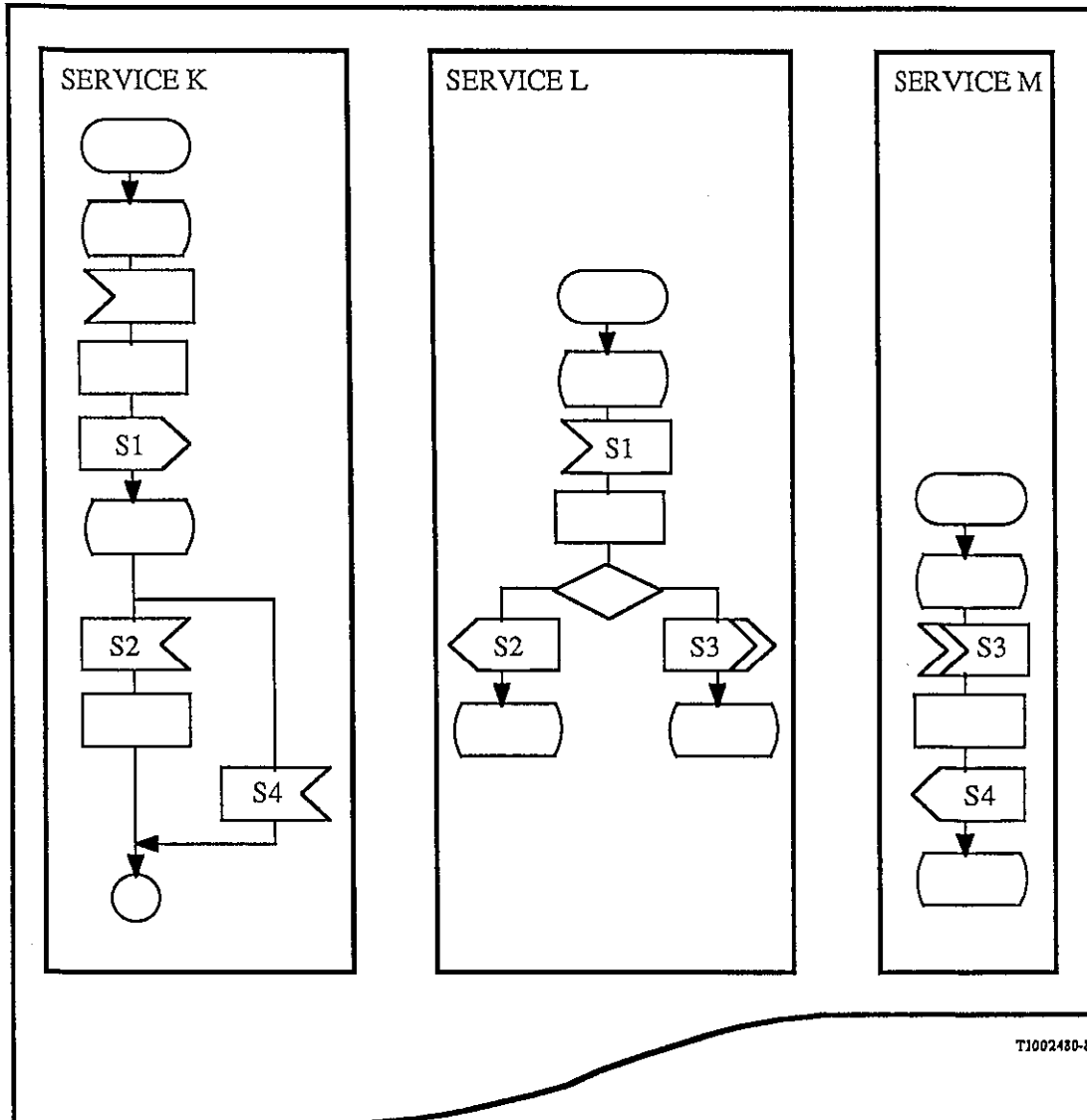


FIGURE D-8.5.2

Page d'un document de service

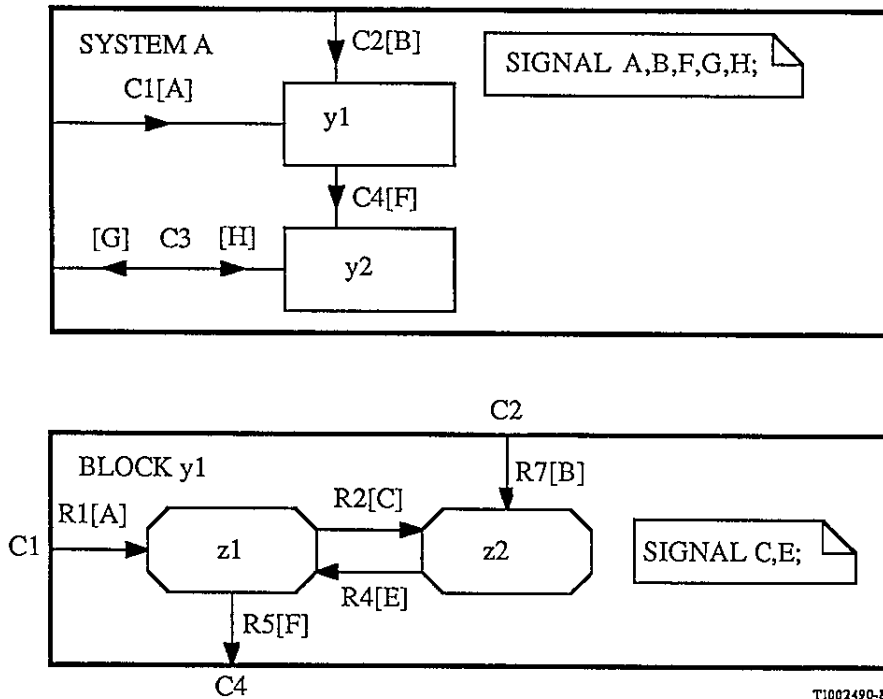
D.8.6 *Combinaison de LDS/GR et de LDS/PR*

Il est possible d'utiliser ensemble le LDS/GR et le LDS/PR pour spécifier un système.

Les concepts de système, de bloc, de processus, de service, de procédure, de macro, de canal, etc. peuvent être spécifiés soit dans l'une, soit dans l'autre de ces versions du LDS.

On passe du LDS/PR au LDS/GR ou vice versa en recourant au mécanisme de référence du langage. Un concept qui est référencé en LDS/PR peut être spécifié en LDS/GR et un concept référencé en LDS/GR peut être spécifié en LDS/PR.

La figure D-8.6.1 est une spécification de système «complète» utilisant une combinaison de LDS/PR et de LDS/GR. C'est le même système que dans les figures D-8.2.1 et D-8.2.2. Chaque définition de l'exemple peut être située dans un document séparé.



```

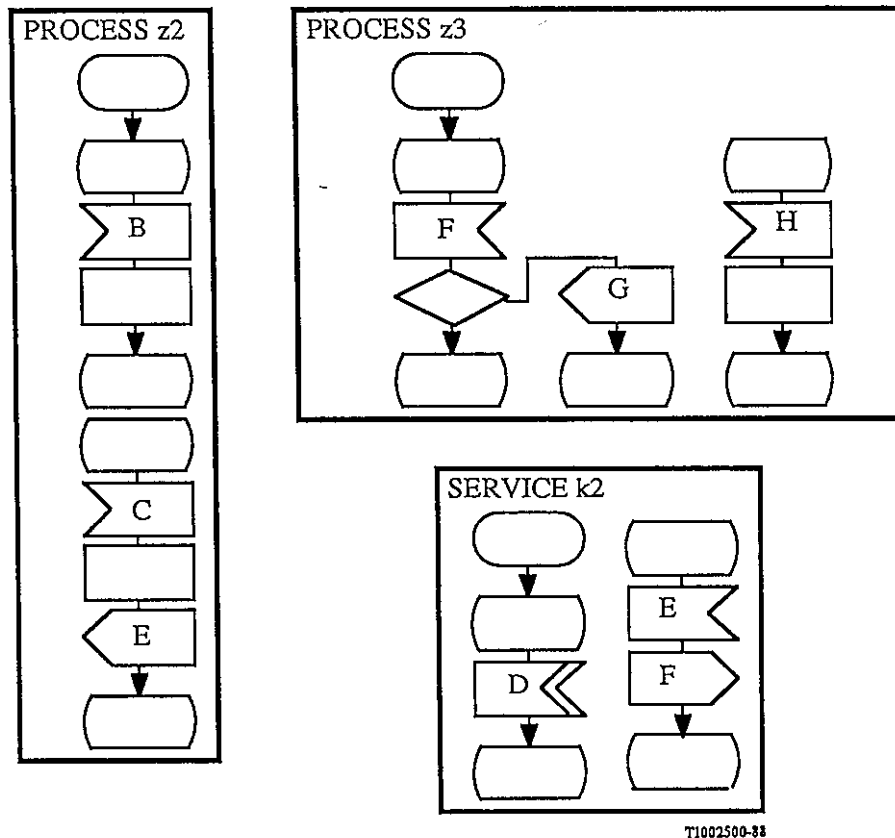
BLOCK y2;
  SIGNALROUTE R8 FROM ENV TO z3 WITH H; |
  FROM z3 TO ENV WITH G; |
  SIGNALROUTE R6 FROM ENV TO z3 WITH F; |
  CONNECT C3 AND R8; |
  CONNECT C4 AND R6; |
  PROCESS z3 REFERENCED; |
ENDBLOCK y2;

PROCESS z1;
  SIGNALROUTE IR1 FROM ENV TO k1 WITH A; |
  SIGNALROUTE IR2 FROM k1 TO ENV WITH C; |
  SIGNALROUTE IR3 FROM k1 TO k2 WITH D; |
  SIGNALROUTE IR4 FROM ENV TO k2 WITH E; |
  SIGNALROUTE IR5 FROM k2 TO ENV WITH F; |
  CONNECT R1 AND IR1; |
  CONNECT R2 AND IR2; |
  CONNECT R4 AND IR4; |
  CONNECT R5 AND IR5; |
  SERVICE k1; |
  START; |
  STATE SS; |
  INPUT A; |
  OUTPUT C; |
  PRIORITY OUTPUT D; |
  NEXSTATE SS; |
  ENDSERVICE k1; |
  SERVICE k2 REFERENCED; |
ENDPROCESS z1;

```

FIGURE D-8.6.1a)

a) Spécification de système avec combinaison possible de LDS/GR et de LDS/PR



T1002500-88

FIGURE D-8.6.1b)

b) Spécification de système avec combinaison possible de LDS/GR et de LDS/PR

## D.9 Mise en correspondance

La présente section décrit certains aspects de mise en correspondance du LDS et du CHILL (voir le § D.9.1), du LDS/GR et du LDS/PR (voir le § D.9.2).

### D.9.1 Mise en correspondance du LDS et du CHILL

Les paragraphes qui suivent décrivent diverses possibilités de mise en correspondance du LDS et du CHILL. Ces solutions sont données succinctement et ne sont pas exhaustives; dans la pratique, rien ne s'oppose à ce qu'on ait recours à d'autres méthodes.

L'examen de la mise en concordance doit porter non seulement sur le compilateur CHILL disponible et sur la machine cible, mais aussi sur des considérations d'ordre plus général. La mise en concordance est une activité intellectuelle très complexe et ce n'est que par l'expérience que les concepteurs/programmeurs pourront décider de la structure de programme CHILL à utiliser pour mettre en œuvre une représentation en LDS donnée. Cela est valable également pour la représentation des fonctions mises en œuvre par un programme CHILL. Une correspondance d'élément à élément (si elle est réalisable) n'est pas nécessairement la meilleure manière d'utiliser le LDS pour représenter les fonctions mises en œuvre par le CHILL.

Sur la base de cette approche, la structure globale d'un programme CHILL tiré d'un diagramme en LDS est présentée dans la figure D-9.1.1.

```

Déclaration: MODULE
    /* module CHILL contenant les signaux et leurs variables associées utilisés dans le diagramme LDS */
    GRANT
    /* octroi des signaux et des variables */
    SIGNALS
    /* définition des signaux */
    SYNMODE (ou NEWMODE)
    /* définition de types */
END Déclaration;

Bloc_fonctionnel: MODULE
    /* module contenant la partie procédurale des diagrammes LDS */
    SEIZE
    /* saisie de tous les signaux et de leurs variables que ce bloc fonctionnel peut recevoir ou émettre */
    /* définition et déclaration des données; ces données valent globalement pour tous les processus qui relèvent de ce module */
    Nom_du_processus: PROCESS ( );
        /* définition et déclaration des données locales */
        état_suivant:=...;
        liaison:=néant;
        DO FOR EVER;
            boucle_d'état: CASE état_suivant OF
                /* boucle sur la variable état_suivant indiquant l'état LDS */
                (étiquette_d'état): RECEIVE CASE
                    (nom_du_signal_1):
                        .
                        .
                        .
                        (nom_du_signal_n):
                ESAC boucle_d'état;
                DO WHILE liaison/=néant;
                    CASE liaison OF
                        (étiquette_de_liaison_1): liaison: = néant;
                        .
                        .
                        .
                        (étiquette_de_liaison_n): liaison: = néant;
                        .
                        .
                    ESAC;
                OD
            END nom_du_processus;
    END Bloc_fonctionnel;

```

FIGURE D-9.1.1

Structure globale d'un programme/CHILL tiré d'un diagramme en LDS

Des exemples de schémas de mise en correspondance de constructions des deux langages sont illustrés aux figures D-9.1.2 à D-9.1.5. Ils concernent les constructions LDS suivantes:

- état et réception ou mise en réserve des signaux; sélection d'un état suivant;
- sortie;



- branchement;
- décision.

Le module de déclaration contient à la fois la définition et la déclaration de tous les signaux employés dans le diagramme LDS qui sont transformés et de toutes les variables qui leur sont associées: toutes ces variables sont octroyées au module qui représente le bloc fonctionnel du diagramme LDS.

Le module du bloc fonctionnel représente le comportement (partie procédurale) des processus du LDS.

Dans ce schéma de traduction, chaque processus du LDS est représenté par une boucle infinie: une variable nommée «état\_suivant» indique l'état à examiner et une variable nommée «liaison» indique des points de branchement possibles qui déterminent des ensembles communs d'instructions.

On choisit au moyen de la construction CASE de CHILL, la valeur de l'état suivant; chaque entrée du CASE identifie un état du LDS. Pour chaque entrée, on choisit entre les signaux d'entrée possibles. Chaque signal d'entrée détermine la série d'actions à accomplir, («Le chemin de transition»).

Chaque chemin de transition se termine par une affectation soit de la variable «état\_suivant», déterminant ainsi directement l'état suivant à examiner, soit de la variable «liaison». Une autre boucle de sélection sur la valeur actuelle de la variable «liaison» permet à chaque transition de prendre fin, au sens du LDS, et à la fin, affecte une valeur à la variable état\_suivant.

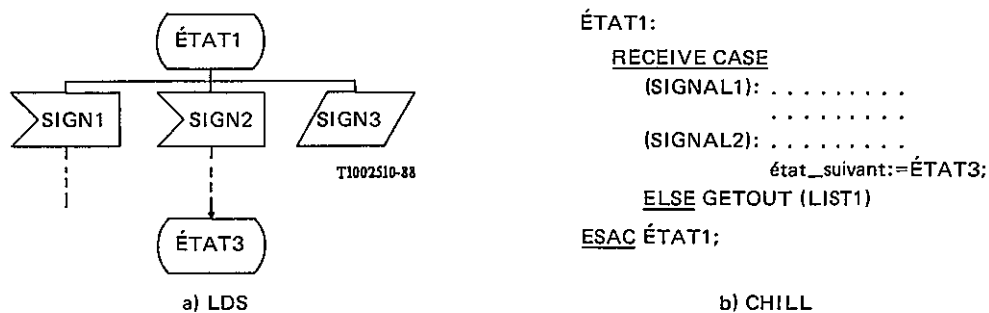
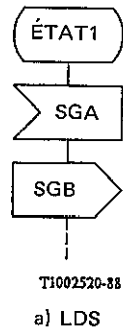


FIGURE D-9.1.2

Exemples de mise en correspondance des instructions STATE/INPUT/SAVE/NEXTSTATE

L'un des problèmes principaux dans la relation entre LDS et CHILL réside dans la sémantique différente de la réception des signaux: en fait, alors que le CHILL ne consomme pas (et par conséquent ne détruit pas) les signaux à moins qu'ils ne soient reçus (signaux persistants), le processus LDS consomme (et par conséquent détruit) tous les signaux reçus jusqu'à ce qu'il y ait concordance avec l'une des entrées énumérées pour cet état. La différence de sémantique a été résolue en introduisant la fonction prédéfinie: GETOUT comme une alternative (chemin ELSE) dans la construction CHILL RECEIVE CASE, comme indiqué à la figure D-9.1.2. La fonction prédéfinie GETOUT du CHILL qui connaît (par paramètres) la liste des signaux d'entrée et de mise en réserve, détruit les autres signaux disponibles au processus lorsque ce dernier est appelé.

Après l'exécution de la fonction GETOUT le sélecteur d'état est mis à 1 de façon à répéter la boucle pour cet état jusqu'à ce qu'un signal d'entrée valable soit sélectionné (ou arrive, s'il n'est pas déjà présent).



```
ÉTAT1 :
  RECEIVE CASE
    (SGA): pi:=get_instance_value();
           send SGB to pi;
           état_suivant:= . . . . .;
    ELSE  état_suivant:=état1;
  ESAC ÉTAT1;
```

b) CHILL

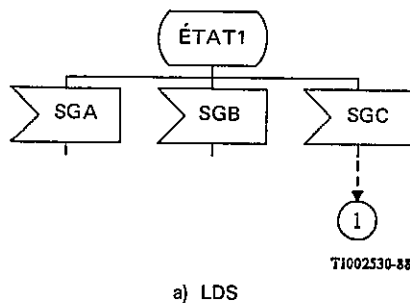
FIGURE D-9.1.3

Exemple de mise en correspondance des sorties

Par exemple, une fois que le signal de sortie SGA, à la figure D-9.1.3, a été reconnu, l'instance appropriée du processus destinataire pour le signal SGB est choisie et le signal SGB est envoyé.

Avant d'envoyer le signal SGB, il peut être nécessaire de remplir certains champs d'information qui doivent être acheminés par le signal. Cela peut être fait immédiatement avant ou, bien plus longtemps avant l'envoi du signal.

Quand il existe un point de branchement dans le diagramme (voir la figure D-9.1.4), on affecte à la variable «liaison» la valeur appropriée. Ainsi qu'il a été expliqué dans la figure D-9.1.1, une boucle sur la variable liaison est exécutée pour déterminer le prochain état à examiner. Dans l'optique du langage de programmation, on peut considérer un point de branchement comme une construction «goto» (aller à); la collecte de tous les points de branchement afin qu'ils puissent être examinés, permet d'écrire entièrement le squelette de programme sans faire intervenir de «goto», ce qui en facilite la lecture.



```
ÉTAT1 :
  RECEIVE CASE
    (S1 in m): case m.id of
      {SGA}: ... ;
      {SGB}: ... ;
      {SGC}: ... ; JOIN:=1;
    ELSE état_suivant:=état1;
  ESAC;
  ESAC ÉTAT1;
```

b) CHILL

FIGURE D-9.1.4

Exemple de branchements de mise en concordance

Une décision en LDS se traduit directement dans la construction CASE de CHILL, comme indiqué à la figure D-9.1.5.

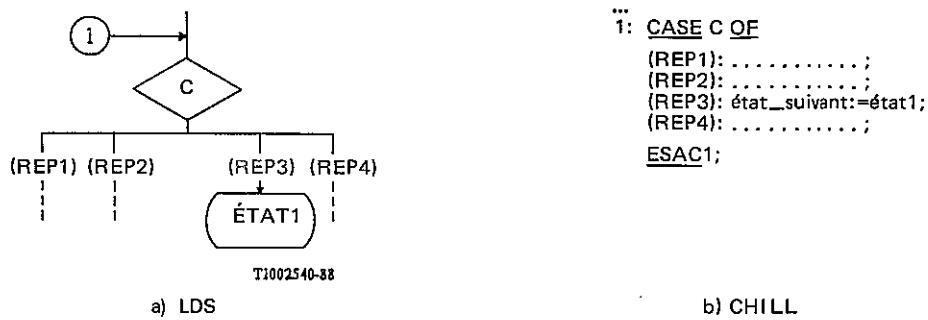


FIGURE D-9.1.5  
Exemple de mise en correspondance des DÉCISIONS

### D.9.2 Mise en correspondance de GR et PR

Compte tenu des restrictions susmentionnées relatives aux macros (§ D.5.1), la version en GR peut toujours être mise en correspondance avec le PR et vice versa.

On trouvera dans tout le présent texte des spécifications équivalentes exprimées sous ces deux formes.

### D.10 Exemples d'application

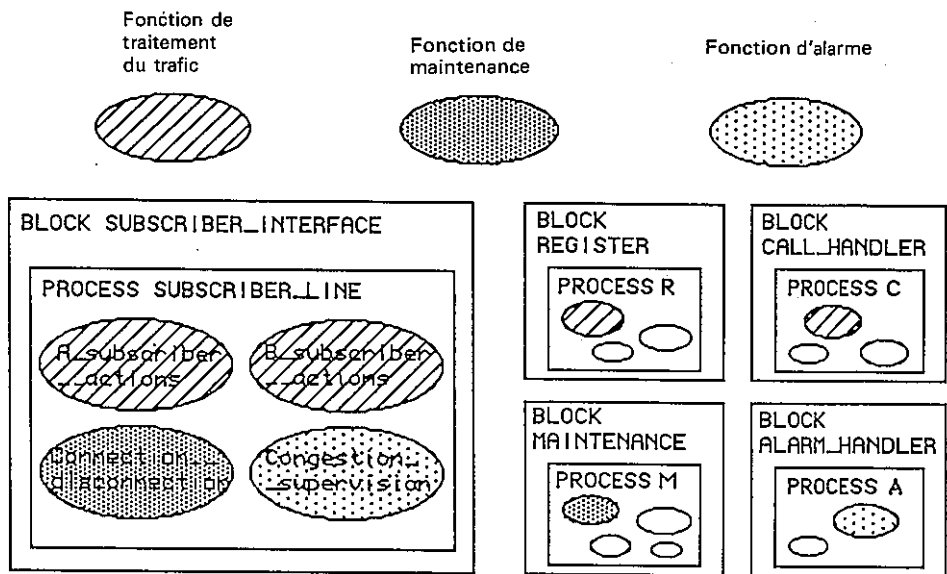
#### D.10.1 Introduction

Le § D.10 contient certains exemples d'utilisation du LDS. Les exemples sont tirés de domaines d'application des télécommunications et mettent en œuvre différents sous-ensembles du LDS. On s'est efforcé de prendre des exemples pratiques et de traiter autant de concepts LDS que possible.

Les exemples sont destinés à illustrer l'utilisation du LDS mais ne constituent pas des spécifications internationales.

#### D.10.2 Le concept de service

Le système présenté ci-après est une illustration du concept de service. Dans ce système, trois «fonctions de système» présentent un intérêt particulier dans cet exemple. Ce sont, une fonction d'écoulement du trafic, une fonction de maintenance et une fonction d'alarme. La figure qui suit, D-10.2.1, illustre la manière dont ces fonctions de système se composent de services subordonnés comportant cinq différents processus et cinq blocs.



T1002550-33

FIGURE D-10.2.1  
Composition de fonctions

La fonction d'écoulement du trafic comprend l'établissement et la terminaison d'un appel téléphonique.

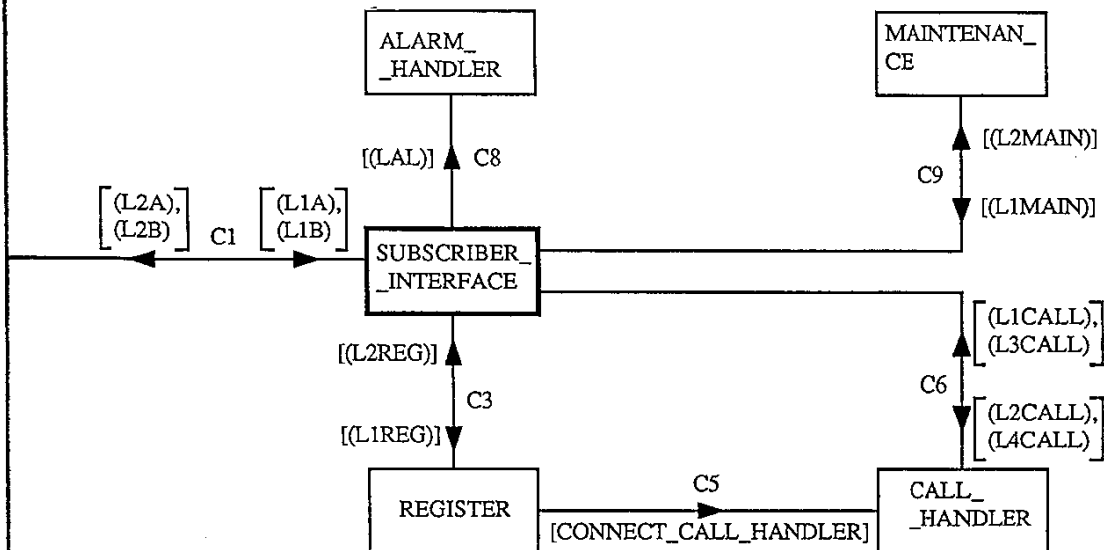
Cet exemple ne constitue pas une documentation complète des fonctions au niveau du système mais simplement une description des quatre services compris dans le processus «SUBSCRIBER\_LINE».

Les trois figures suivantes sont des diagrammes de système (figure D-10.2.2), le diagramme de bloc de «SUBSCRIBER\_INTERFACE» (figure D-10.2.3) et le diagramme de processus de «SUBSCRIBER\_LINE» (figure D-10.2.4). Les canaux, acheminements de signaux et signaux nécessaires sont représentés dans ces diagrammes.

/\*Ce système est un exemple illustrant le concept de service en LDS. Le bloc SUBSCRIBER\_INTERFACE a été choisi à cet effet. Le diagramme de système représente les blocs d'interfonctionnement et les canaux nécessaires aux services dans la SUBSCRIBER\_INTERFACE\*/

```
SIGNAL A_OFF_HOOK,A_ON_HOOK,DIALLED_DIGIT,B_ON_HOOK,B_OFF_HOOK,
DIAL_TONE,CONG_TONE,DIAL_TONE_OFF,CONG_TONE_OFF,RING_TONE_TO_A,
RING_TONE_A_OFF,RING_SIG_TO_B,RING_SIG_B_OFF,CONNECT_DIG_REC,DIGIT,
DISCONN_DIG_REC,CONNECTION,CONGESTION,FETCH_NEXT_DIGIT,
CONNECT_CALL_HANDLER,A_OFF,A_ON,LINE_CONNECTED,LINE_DISCONNECTED,
SEND_RING_TONE,B_ANSWER,DISC_A,RING_SIG,DISC_B,SEND_ALARM,
CEASE_ALARM,CONN_REQ_ACK1,CONN_REQ_ACK2,DISC_REQ_ACK1,
DISC_REQ_ACK2,CONN_REQ,DISC_REQ,B_ON,B_OFF;
```

```
SIGNALLIST L1A = A_OFF_HOOK,A_ON_HOOK,DIALLED_DIGIT;
SIGNALLIST L1B = B_ON_HOOK,B_OFF_HOOK;
SIGNALLIST L2A = DIAL_TONE,CONG_TONE,DIAL_TONE_OFF,CONG_TONE_OFF,
RING_TONE_TO_A,RING_TONE_A_OFF;
SIGNALLIST L2B = RING_SIG_TO_B,RING_SIG_B_OFF;
SIGNALLIST LAL = SEND_ALARM,CEASE_ALARM;
SIGNALLIST L1MAIN = CONN_REQ,DISC_REQ;
SIGNALLIST L2MAIN = CONN_REQ_ACK1,CONN_REQ_ACK2,DISC_REQ_ACK1,
DISC_REQ_ACK2;
SIGNALLIST L1REG = CONNECT_DIG_REC,DIGIT,DISCONN_DIG_REC;
SIGNALLIST L2REG = CONNECTION,CONGESTION,FETCH_NEXT_DIGIT;
SIGNALLIST L1CALL = SEND_RING_TONE,B_ANSWER,DISC_A;
SIGNALLIST L2CALL = A_OFF,A_ON;
SIGNALLIST L3CALL = RING_SIG,DISC_B;
SIGNALLIST L4CALL = LINE_CONNECTED,LINE_DISCONNECTED,B_ON,B_OFF;
```



T1002560-88

FIGURE D-10.2.2

Diagramme de système

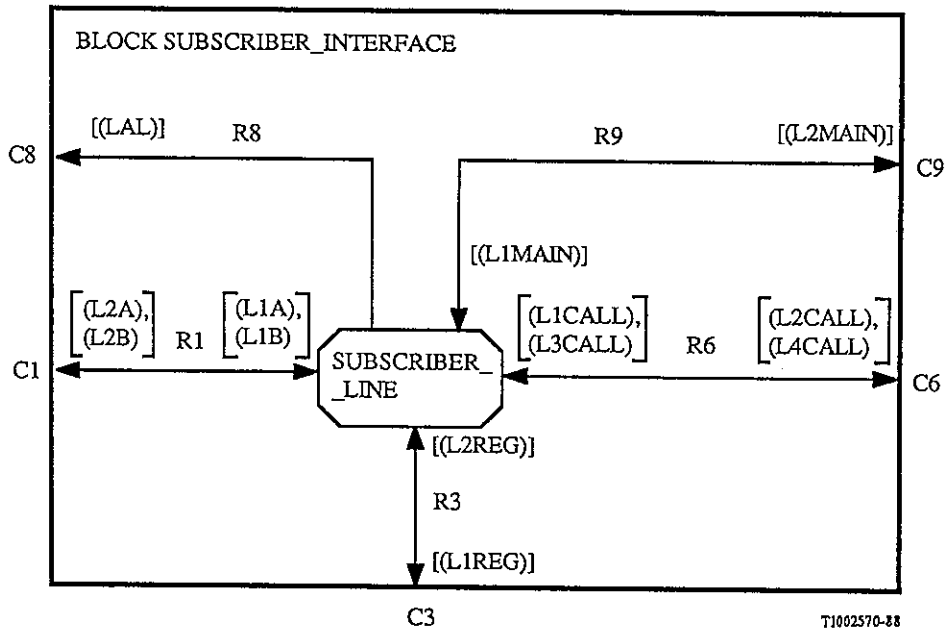
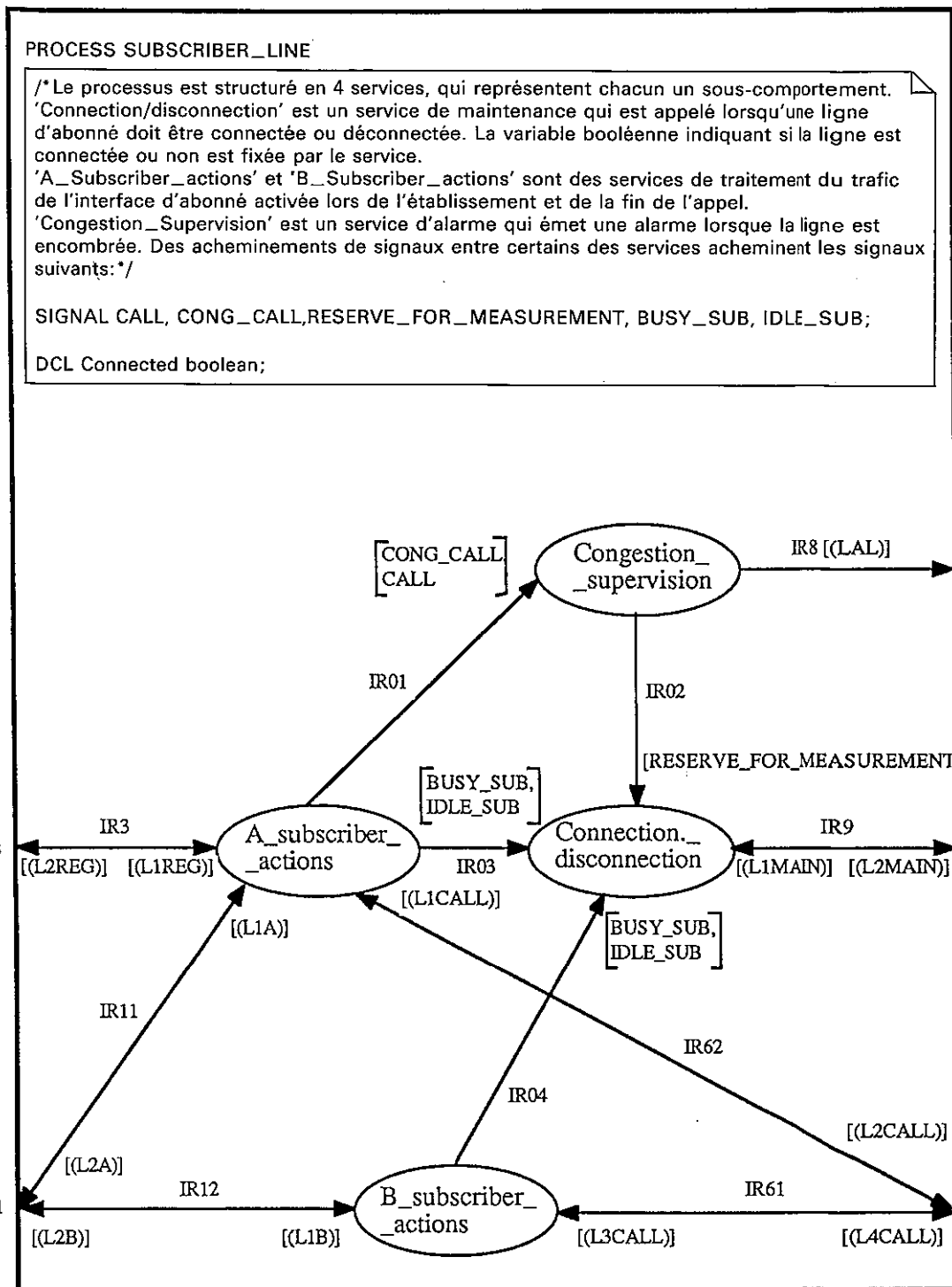


FIGURE D-10.2.3  
Diagramme de bloc

Comme on peut le voir dans le diagramme de processus (figure D-10.2.4), il y a interfonctionnement des services avec les signaux prioritaires par les acheminements de signaux IR01, IR02, IR03 et IR04. Il y a également une interaction entre les services, qui influent les uns sur les autres, au moyen de la variable «globale» «Connected» déclarée dans le processus.

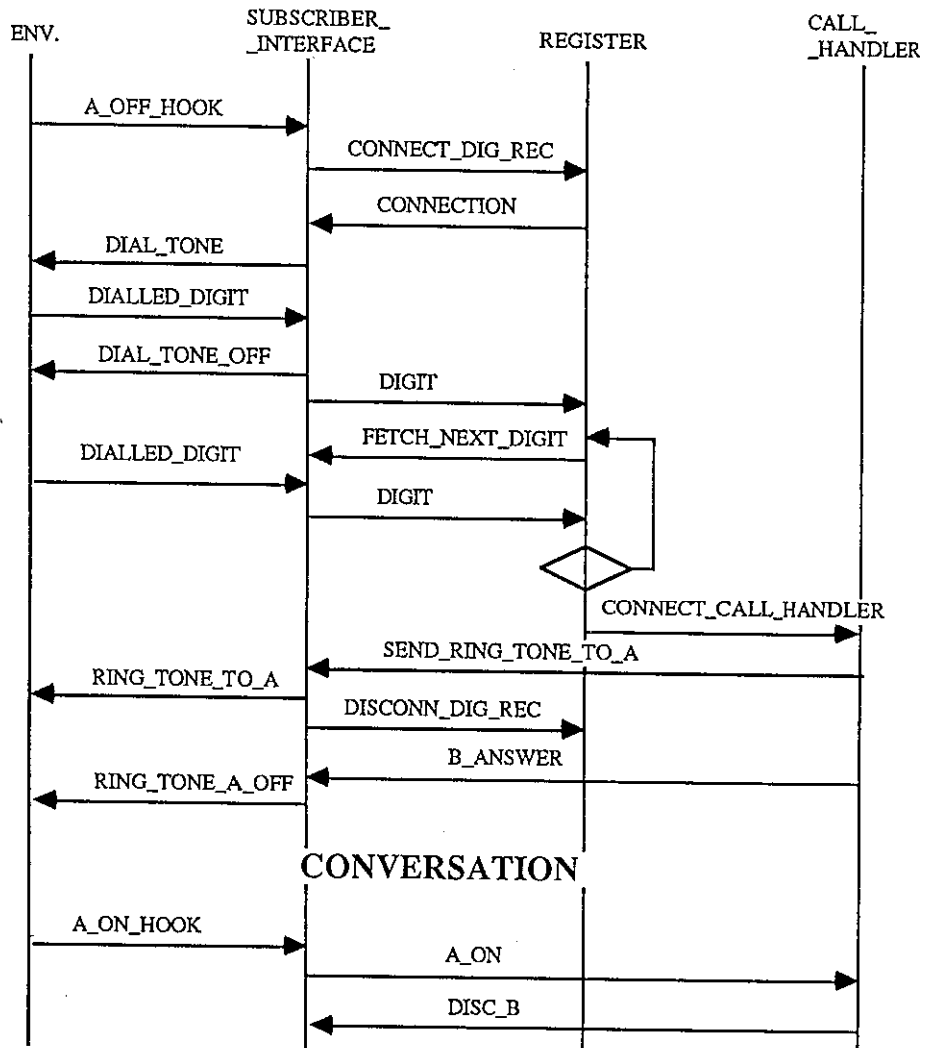


T1002580-33

FIGURE D-10.2.4  
Diagramme de processus

Pour aider à mieux comprendre le rôle de ces services et l'interaction entre les blocs, on a ajouté dans cet exemple plusieurs diagrammes de séquençement.

Les deux premiers diagrammes de séquençement illustrent le cas normal d'interaction entre les blocs au cours d'un appel. L'interaction en cas d'encombrement des enregistreurs est présentée dans le troisième diagramme. Pour simplifier les diagrammes, on a admis qu'il n'y avait pas de délai entre l'émission et la réception d'un signal (voir les figures D-10.2.5 et D-10.2.6).

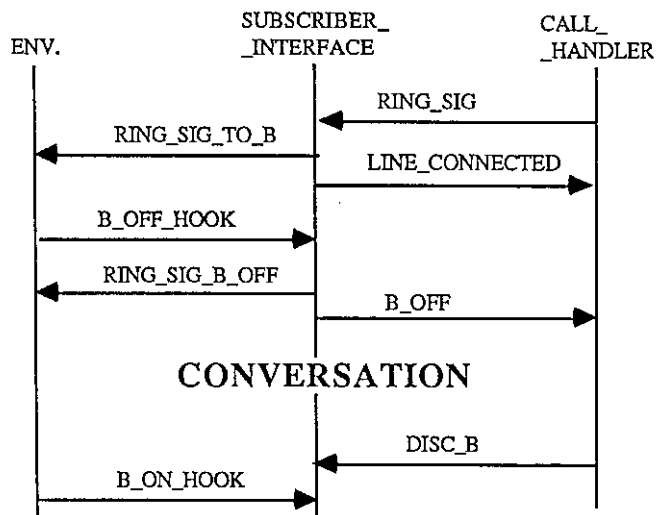


T1002590-88

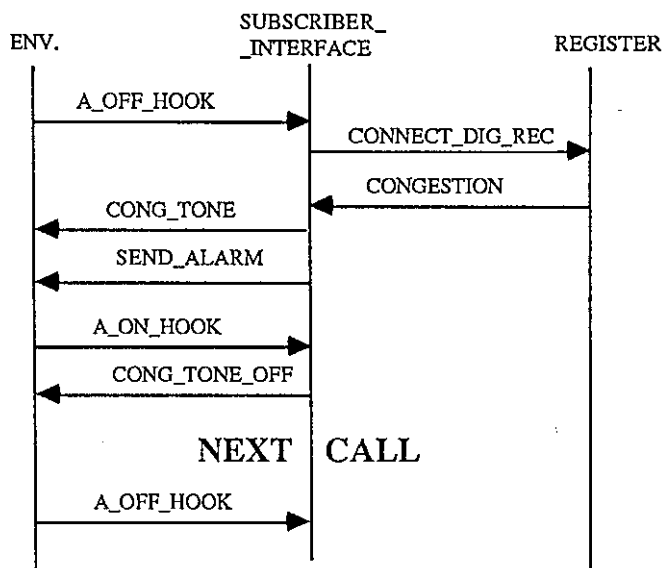
FIGURE D-10.2.5

Diagramme de séquençage. Interaction de bloc dans le cas normal.  
Signaux nécessaires pour l'abonné A





Remarque – Diagramme de séquençage. Interaction de bloc dans le cas normal. Signaux nécessaires pour l'abonné B.

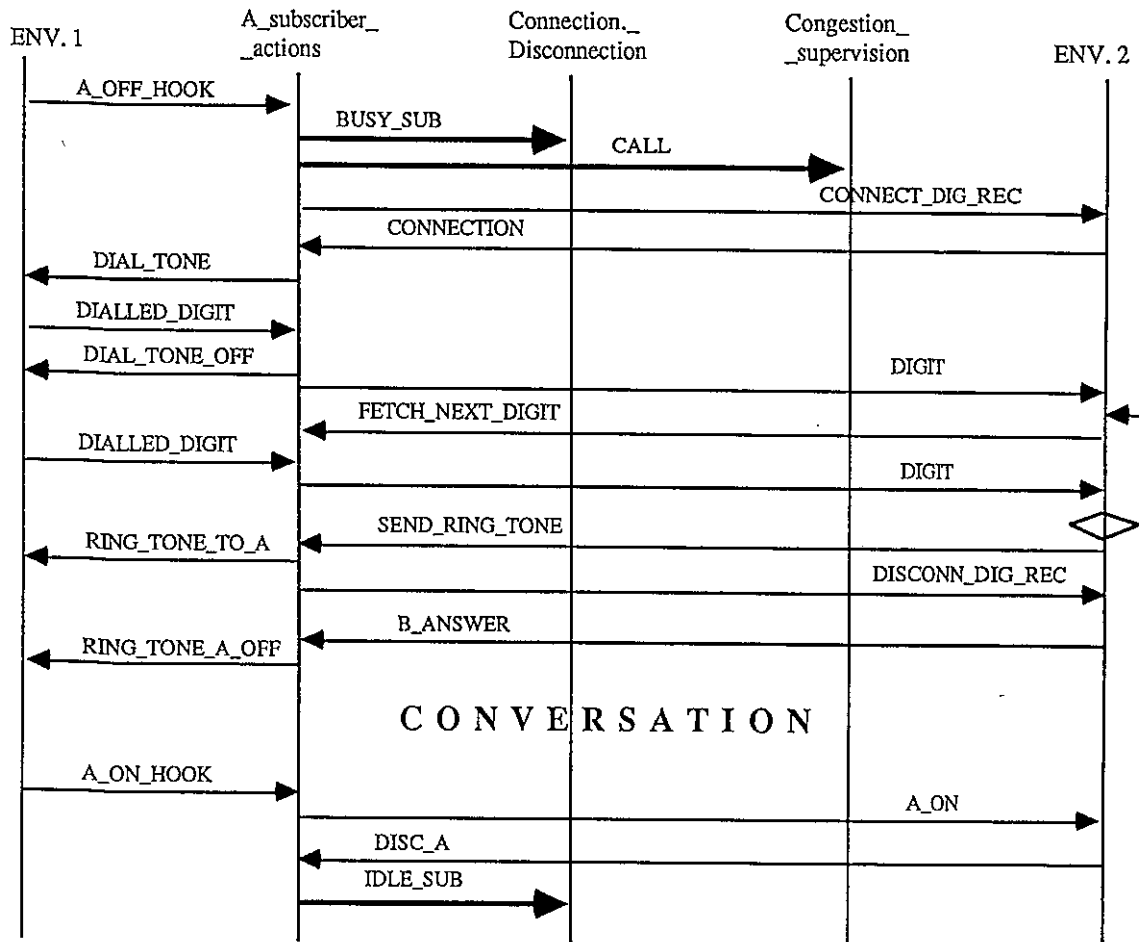


T3002600-88

FIGURE D-10.2.6

Diagramme de séquençage. Interaction de bloc en cas d'encombrement des enregistreurs

L'interaction entre les services du processus «SUBSCRIBER\_LINE» est décrite dans les diagrammes de séquençage qui suivent (voir les figures D-10.2.7 et D-10.2.8).

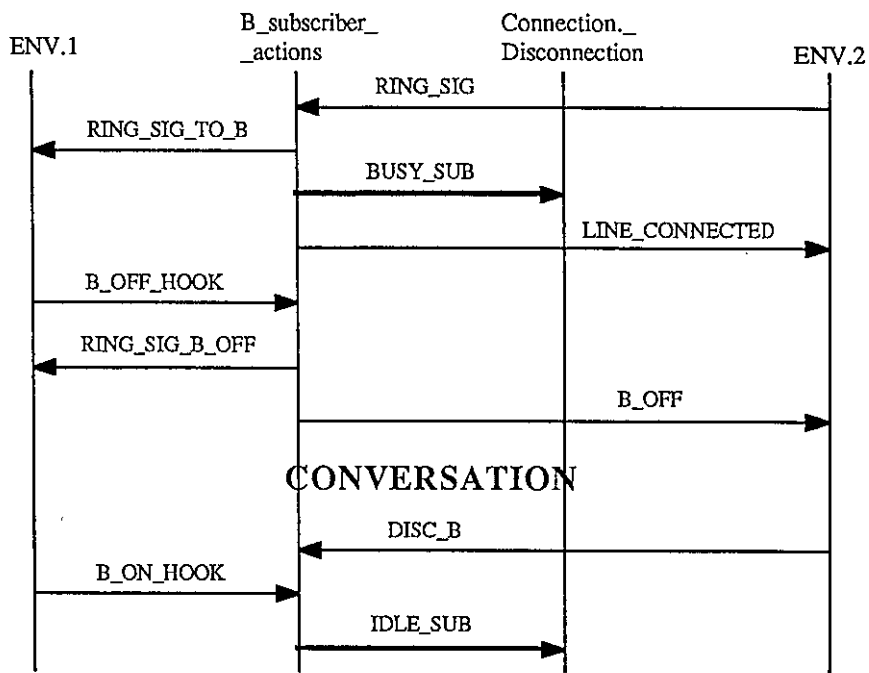


T1002610-88

Remarque – Les signaux prioritaires sont indiqués par des traits plus épais.

FIGURE D-10.2.7

Diagramme de séquençage. Interaction entre les services dans le cas normal. Signaux nécessaires pour l'abonné A



T1002620-88

Remarque – Les signaux prioritaires sont indiqués par des traits plus épais.

FIGURE D-10.2.8

Diagramme de séquençement. Interaction des services dans le cas normal.  
Signaux nécessaires pour l'abonné B

Le comportement de chaque service du processus «SUBSCRIBER\_LINE» est décrit dans les quatre diagrammes de service (voir les figures D-10.2.9 à D-10.2.15).

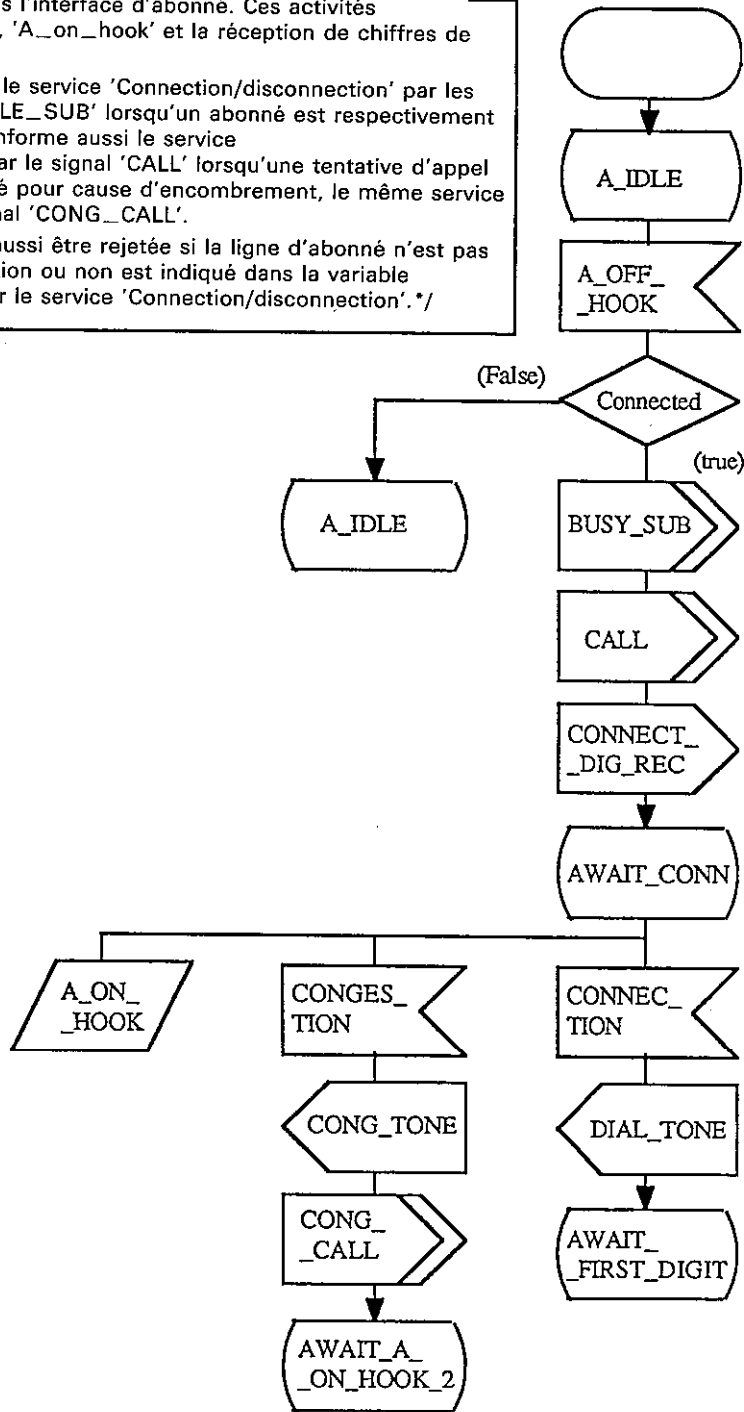
SERVICE A\_subscriber\_actions

1(3)

/\* Le service assure les activités, au cours d'un appel téléphonique, se rapportant à l'abonné A dans l'interface d'abonné. Ces activités comprennent 'A\_off\_hook', 'A\_on\_hook' et la réception de chiffres de l'abonné.

En outre, le service informe le service 'Connection/disconnection' par les signaux 'BUSY\_SUB' et 'IDLE\_SUB' lorsqu'un abonné est respectivement occupé ou non. Le service informe aussi le service 'Congestion\_Supervision' par le signal 'CALL' lorsqu'une tentative d'appel est faite. Si l'appel est rejeté pour cause d'encombrement, le même service est aussi informé par le signal 'CONG\_CALL'.

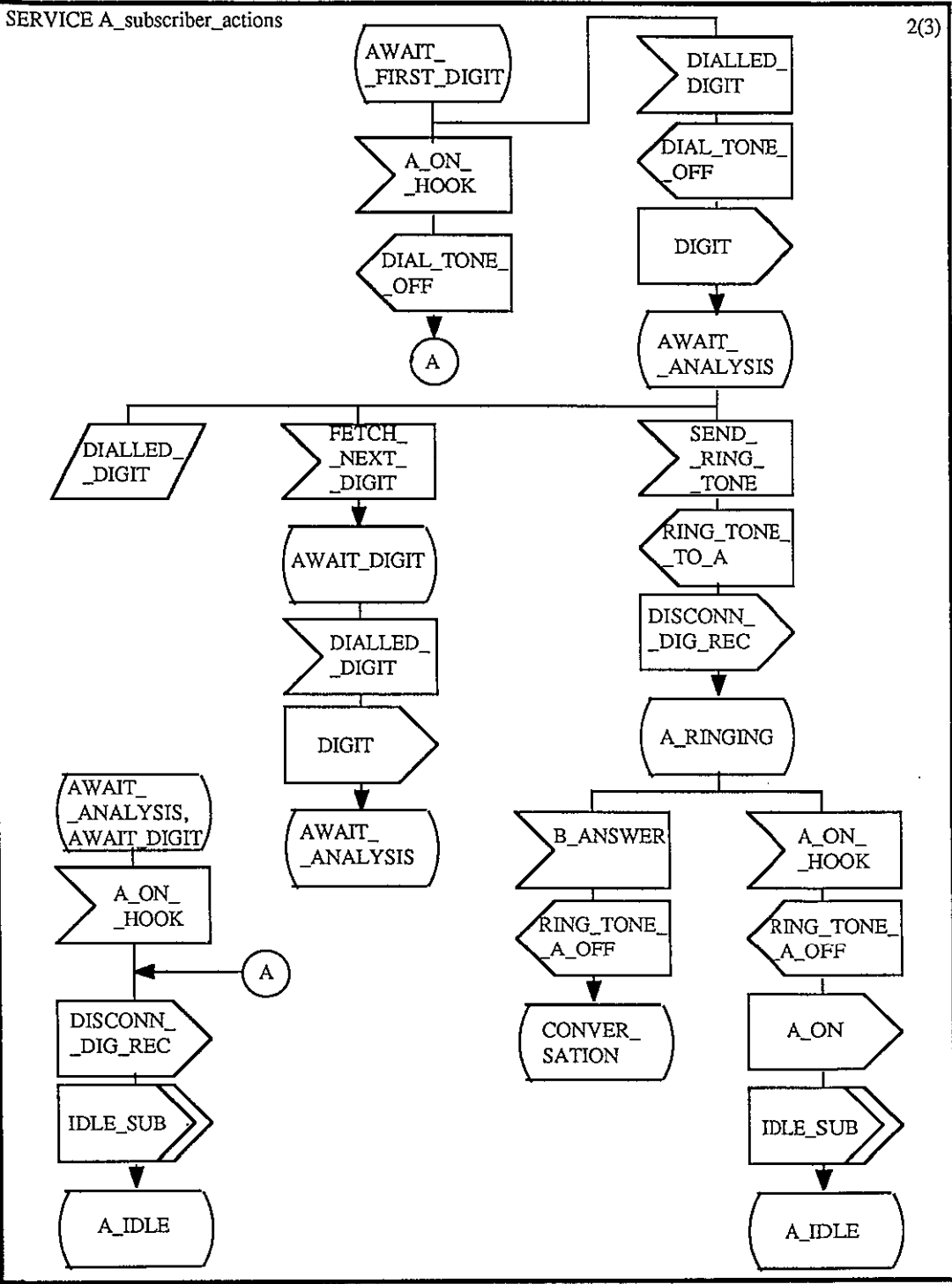
Une tentative d'appel peut aussi être rejetée si la ligne d'abonné n'est pas connectée. L'Etat de connexion ou non est indiqué dans la variable 'Connected' qui est fixée par le service 'Connection/disconnection'.\*/



T1002630-88

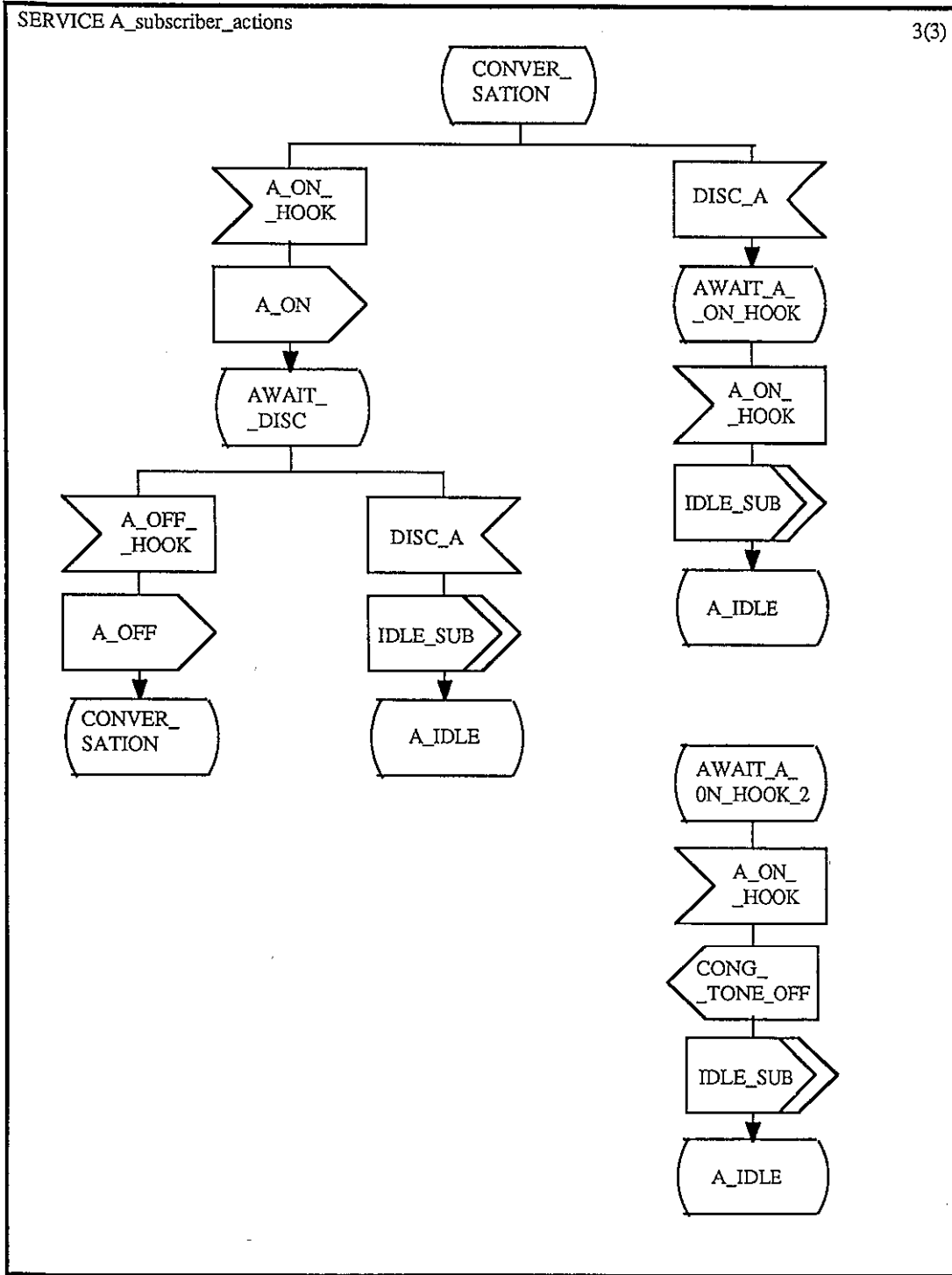
FIGURE D-10.2.9

Diagramme de service



T1002640-88

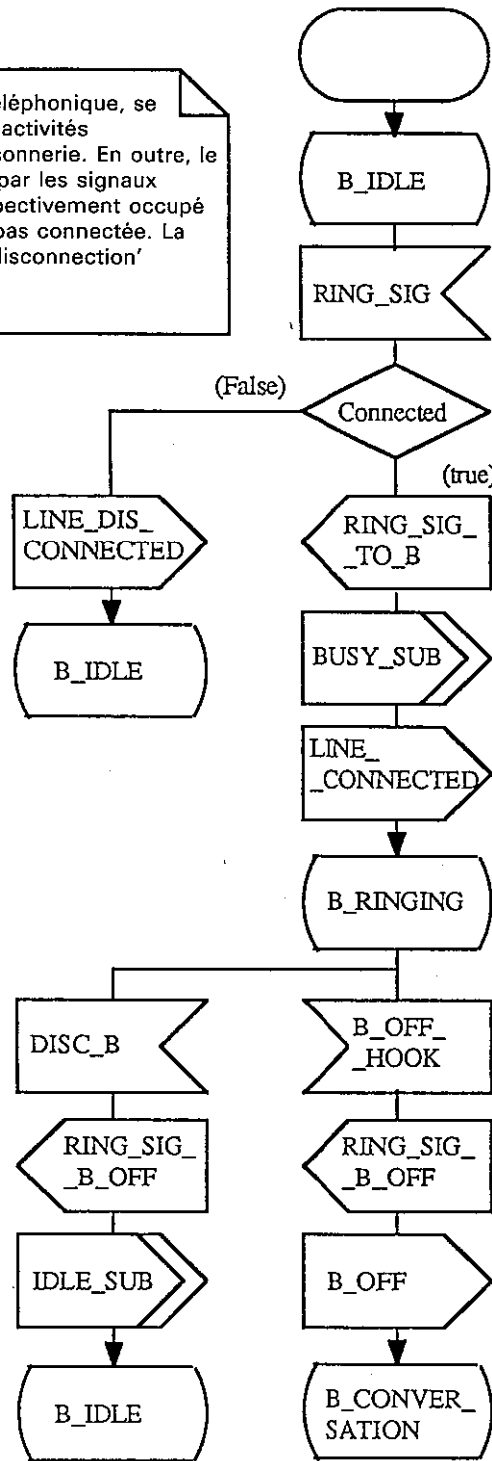
FIGURE D-10.2.10  
Diagramme de service



T1002650-33

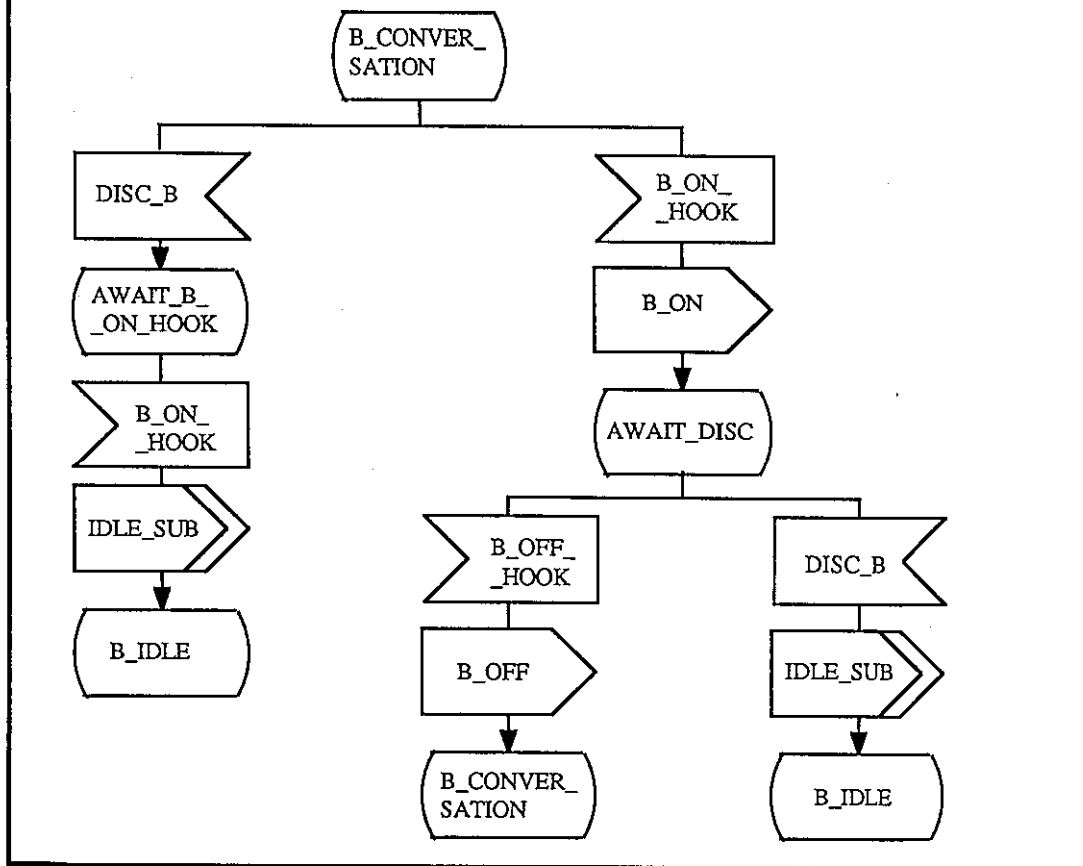
FIGURE D-10.2.11  
Diagramme de service

/\* Le service assure les activités, au cours d'un appel téléphonique, se rapportant à l'abonné B dans l'interface d'abonné. Ces activités comprennent 'B\_on\_hook' et l'émission du signal de sonnerie. En outre, le service informe le service 'Connection\_disconnection' par les signaux 'BUSY\_SUB' et 'IDLE\_SUB' lorsqu'un abonné est respectivement occupé ou non. Un appel, est rejeté si la ligne d'abonné n'est pas connectée. La variable 'Connected' fixée par le service 'Connection\_disconnection' indique si la ligne est connectée ou non.\*/



T1002660-88

FIGURE D-10.2.12  
Diagramme de service



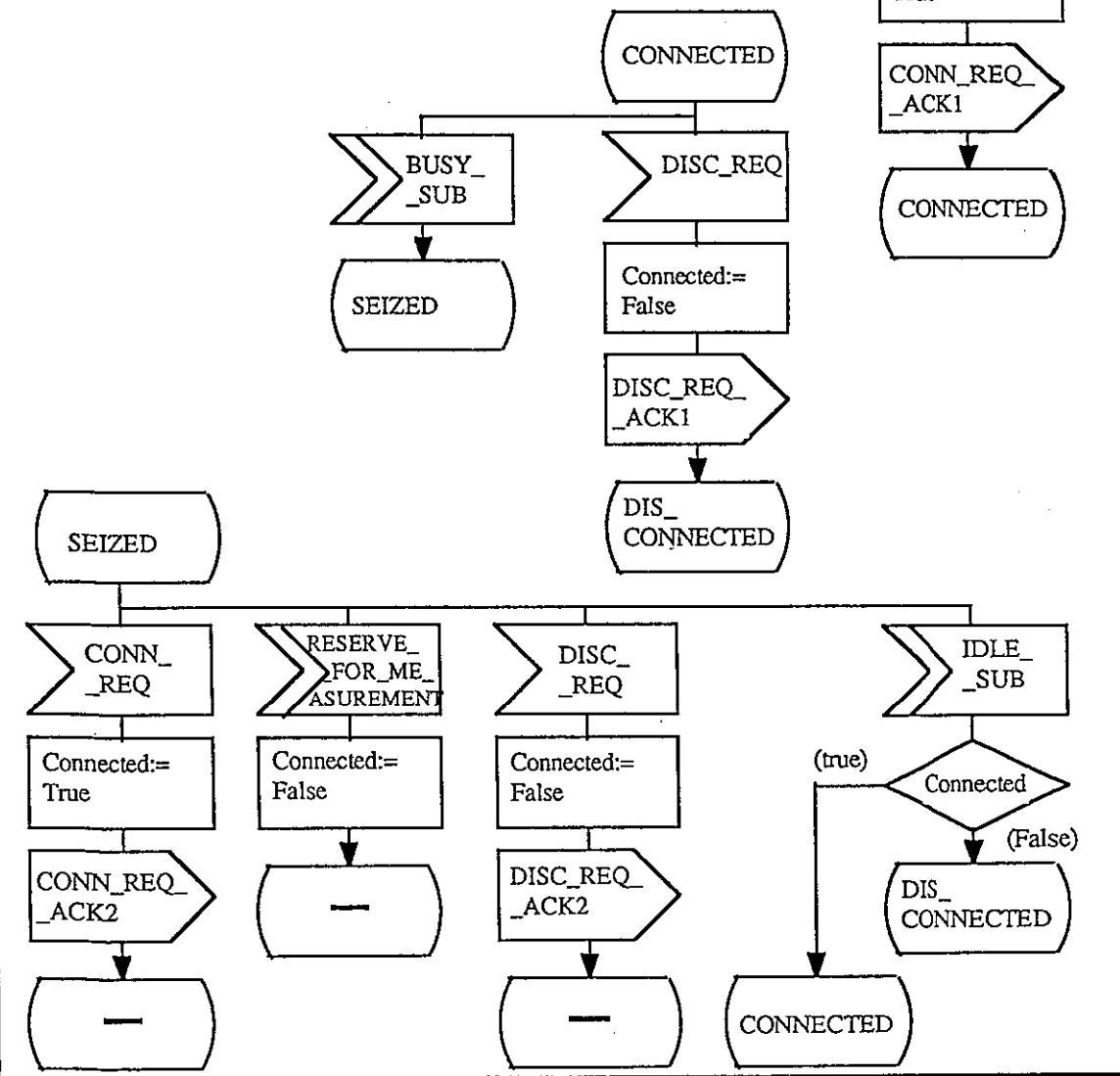
T1002670-83

FIGURE D-10.2.13

Diagramme de service



/\* Le service assure la connexion et la déconnexion d'une ligne d'abonné. L'information sur l'état de la ligne (connecté ou non) est donnée aux autres services par la variable 'connected'. L'action de connecter ou de déconnecter une ligne d'abonné est exécutée lorsque les signaux 'CONN\_REQ' ou 'DISC\_REQ' sont reçus du bloc de maintenance. La ligne est immédiatement connectée ou déconnectée mais, selon l'état de prise de la ligne (si elle est prise ou non), différents signaux sont envoyés au bloc de maintenance. L'information sur l'état de prise est reçue des services 'A\_Subscriber\_actions' et 'B\_Subscriber\_actions' par les signaux 'IDLE\_SUB' et 'BUSY\_SUB'. L'action de déconnecter la ligne d'abonné est aussi exécutée lorsque le signal 'RESERVE\_FOR\_MEASUREMENT' est reçu du service 'Congestion\_supervision'. \*/



T1002630-88

FIGURE D-10.2.14

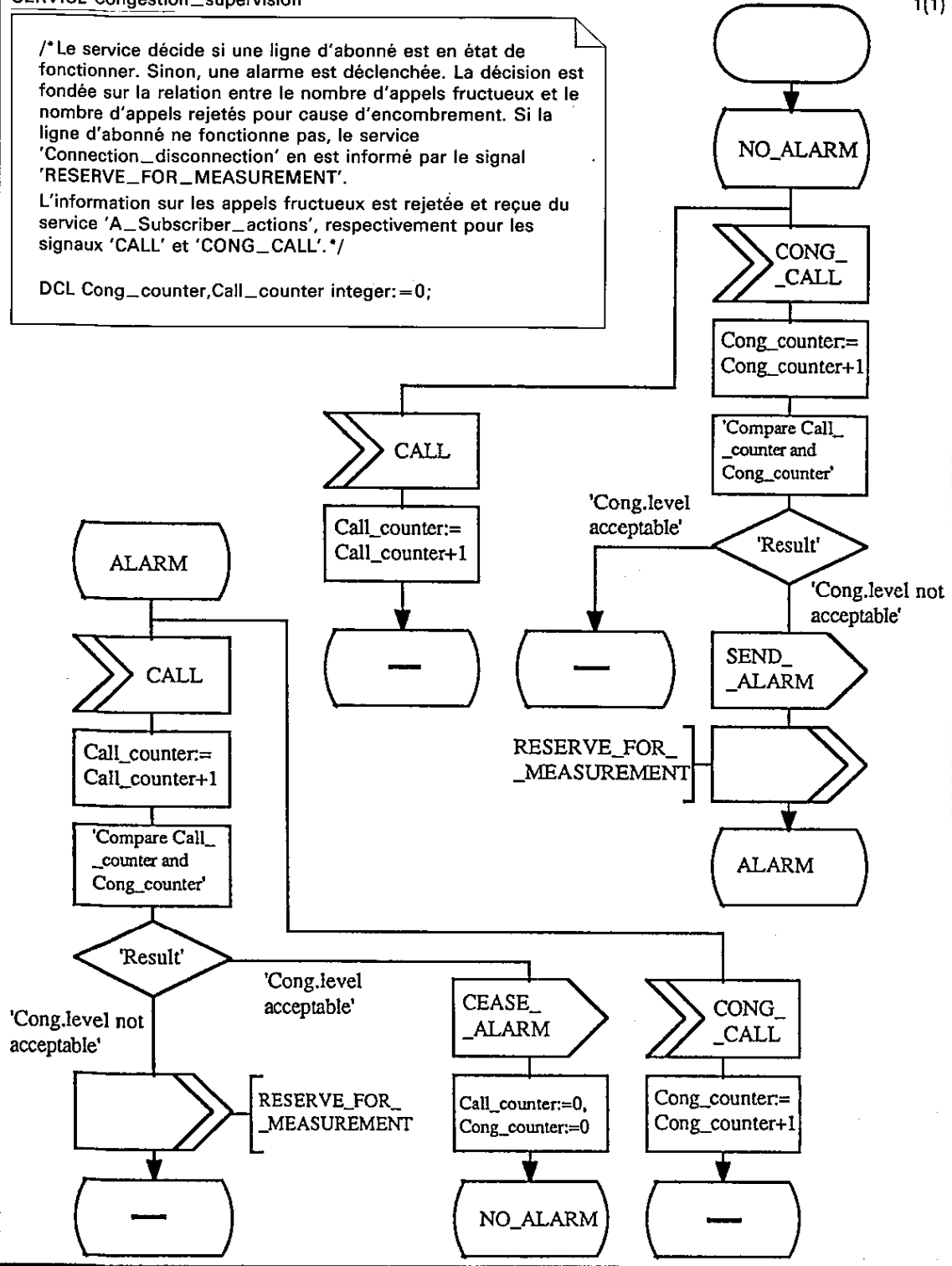
Diagramme de service

SERVICE Congestion\_supervision

1(1)

/\* Le service décide si une ligne d'abonné est en état de fonctionner. Sinon, une alarme est déclenchée. La décision est fondée sur la relation entre le nombre d'appels fructueux et le nombre d'appels rejetés pour cause d'encombrement. Si la ligne d'abonné ne fonctionne pas, le service 'Connection\_disconnection' en est informé par le signal 'RESERVE\_FOR\_MEASUREMENT'.  
L'information sur les appels fructueux est rejetée et reçue du service 'A\_Subscriber\_actions', respectivement pour les signaux 'CALL' et 'CONG\_CALL'.\*/

DCL Cong\_counter, Call\_counter integer:=0;



T1002690-11

FIGURE D-10.2.15

Diagramme de service

## D.11 *Outils pour le LDS*

### D.11.1 *Introduction*

Ce paragraphe présente une série d'outils de soutien pour le LDS. Ces outils peuvent aider à la production de documents, de diagrammes LDS (forme GR) ou de listes imprimées (forme PR), et/ou à la validation des représentations LDS.

Ces directives ne contiennent pas une liste exhaustive de tous les outils éventuels. Les outils nécessaires dépendent de la méthode choisie par l'utilisateur.

En principe, le LDS peut être utilisé sans outils. Toutefois, la complexité inhérente des systèmes modernes est telle que les représentations LDS se révèlent souvent compliquées. De ce fait, il est nécessaire de disposer d'outils automatiques pour préparer la spécification, la conception et la documentation de plusieurs systèmes. Par exemple, la complexité et le coût des tracés manuels, et éventuellement, la mise à jour des documents graphiques d'un central de commutation seraient réduits de façon significative, grâce à l'utilisation d'aides appropriées.

En raison des considérations ci-dessus, le LDS a été conçu de façon à intégrer l'utilisation efficace d'outils de soutien.

### D.11.2 *Catégories d'outils*

Les outils LDS peuvent être classés en fonction des activités effectuées dans le cadre de la production de documents LDS, par exemple:

- Outils pour l'entrée: selon les formes syntaxiques, nous disposons d'aides d'entrée pour les graphiques LDS, sous forme de phrases de texte ou d'illustrations.
- Outils pour la vérification syntaxique: ils comprennent notamment des analyseurs de syntaxe, pour chacune des deux syntaxes.
- Outils pour la production de documents: une fois les documents LDS enregistrés sur ordinateur, les outils peuvent y avoir accès et les reproduire, en utilisant éventuellement plusieurs périphériques. Ces derniers peuvent utiliser une forme syntaxique différente de celle utilisée pour introduire le document. En outre, les outils peuvent produire des documents à partir de ceux initialement introduits.
- Outils de modélisation et d'analyse des systèmes: à partir des documents LDS représentant un système, on peut tirer un modèle du système. Des vérifications peuvent être faites sur ce modèle. Les outils peuvent rechercher les blocages, faire des comparaisons entre les divers modèles du même système (soit, par exemple, entre une spécification et une description) ou exécuter une simulation du fonctionnement du système, etc.
- Outils pour la génération de code: des représentations LDS très détaillées peuvent être utilisées aux fins d'élaboration du logiciel. Les outils peuvent être conçus de façon à pouvoir exécuter de façon guidée et semi-automatique une séquence de code.

Il existe également une catégorie d'outils spécifiques mais utiles:

- Les outils de formation au LDS: ils peuvent être utilisés soit seuls, soit intégrés à d'autres outils. Cette intégration permet de les utiliser pour d'autres fonctions, si nécessaire.

Etant donné que le LDS est utilisé pour plusieurs phases du cycle de vie des systèmes, il est facile de voir l'utilisation qui peut être faite de tous les types d'outils dans un environnement intégré de projet.

### D.11.3 *Entrée des documents*

Aucune spécification particulière n'est requise pour l'introduction de la forme textuelle de phrases du LDS, étant donné que la syntaxe PR équivaut, du point de vue de l'entrée, à toute entrée de chaîne de caractères. En conséquence, on peut utiliser les mêmes outils (éditeurs de textes). Les deux autres syntaxes supposent toutefois une possibilité de traitement graphique.

C'est un fait qu'il est avantageux de disposer d'outils de soutien pour l'introduction du PR, mais il est indispensable d'avoir des outils de soutien pour l'introduction du GR/PE si nous prévoyons d'utiliser ces syntaxes comme moyens d'entrée.

Un éditeur graphique est toujours nécessaire pour des fonctions telles que la connexion de deux symboles, le déplacement d'une série de symboles vers une autre partie de la page ou vers d'autres pages, et pour assurer l'enchaînement des suppressions (la suppression d'un symbole entraîne la suppression de la connexion à ce symbole). Comme pour les outils PR, les outils d'entrée GR/PE devront être conçus sur le modèle de la sémantique/syntaxe LDS.

En conséquence, il devrait éliminer les connexions non valables et pousser les usagers à remplir toutes les parties non complètes, etc.

Lors de la conception des outils, on est confronté à plusieurs problèmes dus aux contraintes physiques des appareils graphiques tels que la «résolution». Il est presque impossible de disposer d'un nombre suffisant de caractères qui soient lisibles tout en permettant la visualisation d'un nombre raisonnable de symboles sur l'écran.

Il faudrait passer en revue les solutions telles que zoom sur fenêtre ou défilements, mais ces solutions ne sont pas totalement satisfaisantes. On peut estimer qu'il n'est pas nécessaire d'avoir un haut niveau de résolution lorsque les diagrammes sont reproduits par l'utilisateur, mais cela devient très souhaitable si les diagrammes sont produits directement par l'utilisateur. Pour la même raison (nécessité d'un tableau synoptique offrant un certain nombre de détails) un niveau élevé de résolution est souhaitable dans la visualisation des diagrammes.

Les outils de soutien des unités d'entrée du PR peuvent être utiles; ils peuvent présenter rapidement à l'utilisateur les mots clés PR attendus.

Ils peuvent procéder immédiatement au «formatage» du PR selon les mots-clés reçus, insérer automatiquement des séparateurs, et présenter à l'utilisateur des clés de fonctions orientées vers le PR, etc.

La mise en œuvre de ces outils peut être basée sur des éditeurs de texte déjà existants qui peuvent être ensuite étendus de façon à inclure les éléments mentionnés ci-dessus.

#### D.11.4 *Vérification des documents*

Une fois que les documents sont enregistrés en mémoire, l'étape suivante consiste à les vérifier. Ils doivent tout d'abord être vérifiés un à un, puis avec les diagrammes correspondants jusqu'à ce que le système total soit vérifié.

Si l'entrée a été faite au moyen d'un outil conçu pour le LDS, il se pourrait qu'une bonne partie de la vérification pour chaque document ait déjà été effectuée.

Les erreurs dues à des opérations «impossibles» (à savoir les entrées ou les mises en réserve précédées d'un quelconque élément, à l'exception d'un état) devront toutes être détectées et corrigées au cours de la phase d'entrée. La détection de certaines erreurs n'est toutefois possible qu'à la fin de la phase d'entrée, aussi bien sur un document unique que dans le cas d'incohérences pouvant exister entre des documents.

Plusieurs règles LDS peuvent être automatiquement vérifiées. Par exemple, la nécessité pour toutes les sorties d'avoir une entrée correspondante.

Dans le cas d'une représentation à plusieurs niveaux, la conformité entre les niveaux peut être vérifiée, dans une certaine mesure.

Le modèle formel LDS peut être utilisé comme base pour élaborer une collection de procédures de vérification.

#### D.11.5 *Reproduction des documents*

Les documents LDS mis en mémoire doivent pouvoir être retrouvés, visualisés et reproduits. Il est nécessaire de disposer d'outils pour toutes ces activités. Il peut s'avérer utile de pouvoir trouver seulement une partie, ou un sous-ensemble, du document. La recherche peut être orientée vers LDS, par exemple: «trouver tous les processus émetteurs» d'un signal donné, ou «dans quels états» est exécutée une action donnée, etc. Les outils de visualisation des informations sont particulièrement importants lorsque l'information doit être visualisée au moyen de la syntaxe graphique. Les mêmes observations que celles faites pour l'entrée des documents dans les syntaxes GR/PE s'appliquent. La reproduction des documents dépend du type de document à reproduire, de la façon dont ces documents sont mis en mémoire et des caractéristiques du périphérique de sortie. Elle peut également dépendre de la façon dont ces documents ont été introduits. Les usagers peuvent désirer une sortie imprimée dans une syntaxe différente de celle utilisée au moment de l'introduction du document.

La reproduction des documents est perturbée par les contraintes pesant sur les périphériques de sortie. Par exemple, un diagramme peut être trop large pour pouvoir être placé sur un espace donné de papier, et de ce fait, il doit être découpé en plusieurs parties. Il faut alors ajouter des connecteurs et des références. Il est quelquefois souhaitable de faire la distinction entre une «adjonction» faite par l'outil et les caractéristiques initiales d'entrée. D'autres contraintes physiques peuvent empêcher la sortie de toutes les informations disponibles, par exemple, une taille particulière de symbole peut s'avérer trop petite pour renfermer la totalité du texte correspondant. Plusieurs méthodes peuvent être choisies, éventuellement sur décision de l'utilisateur. On peut notamment allonger le symbole, découper le texte, le découper mais en ajoutant le texte complet en bas de page, placer le texte à côté du symbole . . . On peut également souhaiter disposer d'outils permettant un plus grand choix de formats de sorties: ces éléments comprennent notamment différentes tailles de symboles, différents formats de sortie, une présentation verticale ou horizontale, etc.

Un document devrait toujours pouvoir être reproduit exactement de la même façon qu'il a été introduit.

#### D.11.6 *Production de documents*

Sur la base des documents LDS introduits par les usagers et enregistrés en mémoire, plusieurs autres documents peuvent être produits automatiquement notamment:

- les listes de signaux, regroupés par processus, par bloc ou par système;
- les diagrammes synoptiques d'état représentant les graphes de processus comme un ensemble d'états reliés par des arcs représentant les transitions;
- les tables de références croisées, fournies par processus, par bloc ou par système;
- le diagramme d'arbre de blocs, montrant la structure des blocs et les niveaux;
- le fonctionnement du système, comme réponse aux séquences des actions de l'environnement;
- des index: ces documents, une fois produits, devront être reproduits et les mêmes considérations susmentionnées seront également valables.

Les documents LDS introduits sous une forme GR peuvent automatiquement être traduits dans la forme PR équivalente et vice versa.

Les considérations suivantes s'appliquent:

- la forme GR contient des informations visuelles qui ne peuvent être traduites dans la forme PR (ce genre d'information n'existe pas en PR). Par exemple, les coordonnées de symboles sont sans signification dans la forme PR;
- les connecteurs reliant des lignes de liaison sur différentes pages peuvent être éliminés.

La traduction inverse, de PR vers GR, est toutefois plus complexe et il est probable qu'elle ne sera pas tout à fait satisfaisante pour tous les lecteurs éventuels.

En raison de la représentation sur deux dimensions de la forme GR, certaines étiquettes qui ont été insérées afin de répondre à la structure séquentielle du PR, peuvent être supprimées, étant donné qu'une ligne de connexion est suffisante.

Habituellement, la traduction produit un modèle de diagramme GR. Ce modèle comprend tous les renseignements nécessaires pour pouvoir procéder au formatage et reproduire le diagramme sur une imprimante graphique.

Il convient de noter que deux outils différents traduisant des PR en GR peuvent obtenir deux représentations GR ayant des présentations différentes. Les représentations GR ainsi obtenues sont toutes les deux correctes à condition qu'elles gardent la sémantique exprimée dans la représentation d'origine.

#### D.11.7 *Modélisation et analyse du système*

Les documents LDS, indépendamment de leurs fonctions de spécification ou de description d'un système sont fondamentalement un modèle de ce système.

Ce modèle, qui est tout d'abord destiné au transfert de l'information d'une personne à une autre, peut également être interprété par des outils; ces derniers servent à vérifier si le modèle est conforme, complet (cet aspect peut éventuellement être laissé en suspens dans le cas des spécifications destinées à ne spécifier que certaines parties d'un système), s'il est correct et s'il répond aux règles du LDS (telles que décrites dans le paragraphe relatif à la vérification des documents).

En outre, les outils peuvent être conçus de façon à utiliser le modèle pour simuler le comportement fonctionnel des systèmes. Le simulateur peut entrer en interaction avec l'environnement et on peut alors tirer les conclusions quant à l'adéquation du modèle par rapport aux besoins des usagers.

Si l'on ajoute des renseignements supplémentaires pour indiquer le temps passé à exécuter chaque action, et pour évaluer les ressources disponibles, (files d'attente, instances, etc.), la simulation peut également étudier la capacité du système.

Les outils doivent être élaborés de façon à créer un modèle de l'environnement, en commençant par le modèle du système, afin d'établir des séquences significatives pour contrôler le système actuel. Une analyse de chemins peut permettre de détecter les blocages dans le modèle.

Le modèle de système peut également être utilisé comme documentation directe. S'il existe des liaisons appropriées entre le système réel et la mémoire de la documentation, on peut élaborer un outil chargé d'imprimer les événements en temps réel du système sur le modèle.

Pour cela, il faudrait établir une corrélation entre les événements physiques tels que vus par le système et les événements logiques traités dans la documentation LDS. Si la documentation est regroupée en plusieurs niveaux d'abstraction, l'utilisateur peut choisir le niveau à tracer. Cet aspect peut être utile dans la mesure où il permet aux usagers ayant des niveaux de formation et d'éducation différents, d'inspecter les activités du système.

Les outils destinés à interpréter le modèle LDS peuvent également être utilisés pour faire ressortir les différences de comportement des différents modèles du même système. Ils peuvent également être utilisés pour comparer les différentes descriptions du système (systèmes produits par différentes compagnies), ou pour comparer la spécification du système avec sa description. De cette manière, il est possible de vérifier si une description du système est conforme à la spécification originale.

#### D.11.8 *Génération de code*

Grâce à une syntaxe formellement définie et à une définition mathématique formelle du LDS, il est possible de concevoir des outils capables de faire correspondre la sémantique des représentations LDS et la sémantique des langages de programmation. Ces outils sont peut-être incapables de fournir des programmes complets d'application, mais ils peuvent s'avérer très utiles pour fournir au moins le cadre général pour un programme réel.

Le § D.9.1 de ces directives à l'intention des usagers, offre un exemple de la façon dont peut être obtenue la mise en correspondance des constructions LDS et CHILL.

#### D.11.9 *Formation*

Un cours complet de formation pour le LDS a été réalisé. Il comprend environ 200 pages de texte et une collection de diapositives (environ 200). Le cours couvre tous les aspects du langage et fournit des exemples et quelques propositions quant à l'utilisation du LDS.

Le cours LDS peut être obtenu auprès de l'Union internationale des télécommunications. Secrétariat général – Section des ventes, Place des Nations, CH-1211 Genève 20 (Suisse).



## 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	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, de télégraphie, de télécopie, circuits téléphoniques 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
<b>Série Z</b>	<b>Langages et aspects informatiques généraux des systèmes de télécommunication</b>