

## 1 Introduction au LDS

### 1.1 Introduction

Le but poursuivi, en recommandant l'utilisation du LDS (langage de description et de spécification fonctionnelles), est d'avoir un langage permettant de spécifier et de décrire sans ambiguïté le comportement des systèmes de télécommunication. Les spécifications et les descriptions faites à l'aide du LDS doivent être formelles dans ce sens qu'il doit être possible de les analyser et de les interpréter sans ambiguïté.

Les termes spécification et description sont utilisés dans le sens ci-après:

- a) la spécification d'un système est la description du comportement souhaité de celui-ci, et
- b) la description d'un système est la description du comportement réel de celui-ci.

*Remarque* — Etant donné qu'il n'est pas fait de distinction entre l'utilisation du LDS pour la spécification et son utilisation pour la description, le terme spécification dans le texte qui suit est utilisé pour désigner à la fois le comportement souhaité et le comportement réel.

Une spécification du système, au sens large, est la spécification à la fois du comportement et d'un ensemble de paramètres généraux du système. Toutefois, le LDS ne vise qu'à décrire les aspects relatifs au comportement d'un système; les paramètres généraux concernant des propriétés telles que la capacité et le poids doivent être décrits à l'aide de techniques différentes.

#### 1.1.1 Objectifs

Les objectifs généraux qui ont été pris en compte lors de la définition du LDS sont de fournir un langage:

- a) facile à apprendre, à utiliser et à interpréter;
- b) permettant l'élaboration de spécifications dépourvues d'ambiguïté pour faciliter la soumission des offres et le partage des commandes;
- c) extensible pour permettre un développement ultérieur;
- d) permettant l'application de plusieurs méthodologies de spécification et de conception de système, sans supposer a priori l'une quelconque de ces méthodologies.

#### 1.1.2 Domaine d'application

Le domaine d'application principal du LDS est la description du comportement des systèmes en temps réel dans certains de leurs aspects. Ces applications comprennent:

- a) le traitement des appels (par exemple: écoulement, signalisation téléphonique, comptage aux fins de taxation, etc.) dans les systèmes de commutation;
- b) la maintenance et la relève des dérangements (par exemple: alarme, relève automatique des dérangements, essais périodiques, etc.) dans les systèmes généraux de télécommunication;
- c) la commande du système (par exemple: protection contre les surcharges, procédures de modification et d'extension, etc.);
- d) les fonctions d'exploitation et de maintenance, la gestion des réseaux;

- e) les protocoles de communication de données.

Il va de soi que le LDS peut aussi servir à la spécification fonctionnelle du comportement d'un objet lorsque celui-ci peut être spécifié au moyen d'un modèle discret, c'est-à-dire un objet communiquant avec son environnement au moyen de messages discrets.

Le LDS est un langage particulièrement riche qui peut être utilisé à la fois pour des spécifications de haut niveau informelles (et/ou formellement incomplètes), des spécifications partiellement formelles et des spécifications détaillées. L'utilisateur doit choisir les parties appropriées du LDS en fonction du niveau de communication souhaité et de l'environnement dans lequel le langage sera utilisé. Selon l'environnement dans lequel une spécification est utilisée, certains éléments qui relèvent du simple bon sens pour l'émetteur et le destinataire de la spécification, ne seront pas explicites.

Ainsi, le LDS peut être utilisé pour:

- a) établir les spécifications d'une installation,
- b) établir les spécifications d'un système,
- c) établir des Recommandations du CCITT,
- d) établir des spécifications de conception d'un système,
- e) établir des spécifications détaillées,
- f) concevoir un système (à la fois globalement et dans le détail),
- g) effectuer des essais d'un système,

l'organisation à laquelle appartient l'utilisateur pouvant choisir le niveau d'application du LDS qui convient.

.bp

### 1.1.3 Spécification d'un système

Le LDS décrit le comportement d'un système sous la forme de stimulus/réaction, étant admis que les stimuli aussi bien que les réactions sont des entités discrètes et contiennent de l'information. En particulier, la spécification d'un système est vue comme étant la séquence de réactions associée à une séquence de stimuli.

Le modèle de spécification d'un système est fondé sur la notion de machine à états finis étendue.

Le LDS fait appel à des concepts structurels qui facilitent la spécification des grands systèmes et des systèmes complexes. Il est ainsi possible de subdiviser la spécification d'un système en unités faciles à gérer qui peuvent être traitées et comprises de manière indépendante. La subdivision peut s'opérer en plusieurs étapes qui permettent d'obtenir une structure hiérarchique d'unités définissant le système à différents niveaux.

## 1.2 Grammaires du LDS

Le LDS offre le choix entre deux formes syntaxiques différentes pour représenter un système: une représentation graphique (LDS/GR) et une représentation textuelle (LDS/PR). Comme ces formes sont toutes les deux des représentations concrètes de la même sémantique du LDS, elles sont équivalentes du point de vue sémantique. En particulier, elles sont toutes les deux équivalentes à une grammaire abstraite en ce qui concerne les concepts correspondants.

Un sous-ensemble du LDS/PR est commun avec le LDS/GR. Ce sous-ensemble est appelé grammaire textuelle commune.

La figure 1.1 montre les relations qui existent entre le LDS/PR, le LDS/GR, les grammaires concrètes et la grammaire abstraite.

.rs

#### **Figure 1.1, p.**

A chaque grammaire concrète, est associée sa propre syntaxe et les rapports qu'elle a avec la grammaire abstraite (c'est-à-dire la manière de la transformer en syntaxe abstraite). Avec cette méthode, la définition de la sémantique du LDS est unique; chacune des grammaires concrètes héritera de la sémantique par l'intermédiaire de ses relations avec la grammaire abstraite. Cette méthode permet également d'assurer l'équivalence entre le LDS/PR et le LDS/GR.

On dispose également d'une définition formelle du LDS qui définit la manière de transformer la spécification d'un système dans la syntaxe abstraite et comment interpréter une spécification, donnée en termes de syntaxe abstraite.

#### *1.3 Définitions fondamentales*

La présente Recommandation fait appel à des conventions et à des concepts généraux dont les définitions sont données ci-après:

##### *1.3.1 Type, définition et instance*

Dans la présente Recommandation, les concepts de type, d'instance de type et les relations qui existent entre elles, jouent un rôle fondamental. Le schéma et la terminologie utilisés sont explicités ci-après et illustrés par la figure 1.2.

.bp

## Figura 1.2, p.

Un type se définit par des définitions, lesquelles définissent toutes les propriétés associées à ce type. Un type peut être décomposé en un nombre quelconque d'instances. Toute instance d'un type particulier possède toutes les propriétés définies pour ce type.

Ce schéma s'applique à plusieurs concepts du LDS, c'est-à-dire qu'il existe des définitions de système et des instances de système, des définitions de processus et des instances de processus.

Le type de données constitue une catégorie spéciale de type (voir les § 2.3 et 5).

*Remarque* — Pour éviter d'alourdir le texte, on peut s'abstenir d'utiliser le terme instance. Ainsi, pour exprimer qu'une instance du système est interprétée, on écrira <<un système est interprété | | |>.

### 1.3.2 Environnement

Les systèmes qui sont spécifiés en LDS réagissent d'après les stimuli qu'ils reçoivent du monde extérieur. Ce monde extérieur est appelé environnement du système en cours de spécification.

On suppose qu'il y a une ou plusieurs instances de processus dans l'environnement et, par conséquent, les signaux circulant de l'environnement en direction du système ont des identités associées à ces instances de processus. Ces processus ont des valeurs PID différentes des autres valeurs PID du système (voir § 5.6.10).

Bien que le comportement du système soit non déterministe, il doit obéir aux contraintes imposées par la spécification du système.

### 1.3.3 Erreurs

Une spécification de système est une spécification de système correcte en LDS seulement si elle répond aux règles syntaxiques et aux conditions statiques du LDS.

Lorsqu'une spécification LDS correcte est interprétée et qu'une condition dynamique se trouve violée, une erreur apparaît. Une interprétation d'une spécification de système qui conduit à une erreur indique que le comportement du système ne peut pas être déterminé à partir de la spécification.

## 1.4 Présentation

### 1.4.1 Structuration du texte

Les § 2, 3, 4 et 5 de la présente Recommandation sont structurés par thèmes, ils comportent des intitulés précédés éventuellement par une introduction: ces intitulés sont les suivants:

a) *Grammaire abstraite* — d'écrite par une syntaxe abstraite et des conditions statiques pour définitions bien formées.

b) *Grammaire textuelle concrète* — qui concerne à la fois la grammaire textuelle courante utilisée pour le LDS/PR et le LDS/GR et la grammaire uniquement utilisée pour le LDS/PR. Cette grammaire est d'écrite au moyen d'une syntaxe textuelle, de conditions statiques et de règles de définitions bien formées, concernant la syntaxe textuelle, et de la relation de la syntaxe textuelle avec la syntaxe abstraite.

c) *Grammaire graphique concrète* — d'écrite par la syntaxe graphique, les conditions statiques et les règles de définitions bien formées concernant la syntaxe graphique, la relation de cette syntaxe avec la syntaxe abstraite et quelques règles de dessin qui viennent en supplément (à celles indiquées au § 2.2.4).

.bp

d) *Sémantique* — donnant une signification à un type, confère les propriétés de ce type, la façon avec laquelle une instance de ce type est interprétée et toutes conditions dynamiques qui doivent être remplies par l'instance de ce type pour avoir un comportement correct au sens du LDS.

e) *Modèle* — donne la correspondance avec les abréviations exprimées en termes de constructions en syntaxe concrète stricte précédemment définies.

f) *Exemples*.

#### 1.4.2 Intitulés

L'introduction qui précède éventuellement les intitulés, est uniquement destinée à faciliter la compréhension du texte et à ce titre doit être considérée comme étant une partie officielle de la Recommandation.

S'il n'existe pas de texte pour un intitulé, tout l'intitulé est omis.

La suite de la présente section décrit les autres formalismes particuliers utilisés dans chaque intitulé et les titres utilisés. Elle peut également être considérée comme un exemple de présentation typographique du premier niveau des intitulés définis ci-dessus, ce texte appartenant à l'introduction.

##### *Grammaire abstraite*

La notation en syntaxe abstraite est définie au § 1.5.1.

L'absence de l'intitulé *grammaire abstraite* indique qu'il n'existe pas d'autres syntaxes abstraites pour le sujet traité et que la syntaxe concrète correspond à la syntaxe abstraite définie par une autre section de texte numérotée.

On peut se référer à une des règles dans la syntaxe abstraite à partir de tout intitulé en maintenant le nom de la règle en italique.

Les règles dans la notation formelle peuvent être suivies par des paragraphes qui définissent les conditions qui doivent être satisfaites par une définition LDS bien formée et qui peuvent être vérifiées sans interprétation d'une instance. Les conditions statiques à ce niveau se réfèrent uniquement à la syntaxe abstraite. Les conditions statiques qui ne concernent seulement que la syntaxe concrète sont définies postérieurement à la syntaxe concrète. La syntaxe abstraite, associée aux conditions statiques applicables à la syntaxe abstraite, définit la grammaire abstraite du langage.

##### *Grammaire textuelle concrète*

La syntaxe textuelle concrète est spécifiée au moyen de la Backus-Naur Form (BNF) étendue de la description de la syntaxe définie au § 2.1 de la Recommandation Z.200 (voir également le § 1.5.2).

La syntaxe textuelle est suivie par des paragraphes définissant les conditions statiques qui doivent être satisfaites dans un texte bien formé et qui peuvent être vérifiées sans interprétation d'une instance. Cela s'applique également aux conditions statiques, si elles existent, pour la grammaire abstraite.

Dans de nombreux cas, il y a une simple relation entre la syntaxe concrète et la syntaxe abstraite étant donné qu'une règle de la syntaxe concrète est simplement représentée par une seule règle dans la syntaxe abstraite. Lorsque le m | me nom est utilisé dans la syntaxe abstraite et dans la syntaxe concrète, afin d'indiquer qu'il représente le m | me concept, le texte précisant que <<<x> dans la syntaxe concrète représente X dans la syntaxe abstraite>> est implicite dans la description du langage et est souvent omis. Dans ce contexte, le cas est ignoré mais les sous-catégories sémantiques soulignées sont significatives.

La syntaxe textuelle concrète qui ne constitue pas une forme abrégée (syntaxe dérivée modélisée par d'autres constructions LDS) est une syntaxe textuelle concrète stricte. La relation entre la syntaxe textuelle concrète et la syntaxe

abstraite est seulement définie pour la syntaxe textuelle concrète stricte.

La relation entre la syntaxe textuelle concrète et la syntaxe abstraite est omise si le sujet en cours de définition est une forme abrégée qui est modélisée par d'autres constructions LDS (voir le paragraphe *modèle* ci-après).

### *Grammaire graphique concrète*

La syntaxe graphique concrète est spécifiée dans la forme BNF élargie de la description de la syntaxe définie au § 1.5.3.

La syntaxe graphique est suivie par des paragraphes définissant les conditions statiques qui doivent être satisfaites dans une LDS/GR bien formée et qui peuvent être vérifiées sans interprétation d'une instance. Cela s'applique également aux conditions statiques, si elles existent, pour la grammaire abstraite.

La relation entre la syntaxe graphique concrète et la syntaxe abstraite est omise si le sujet en cours de définition est une forme abrégée qui est modélisée par d'autres constructions LDS (voir le paragraphe *modèle* ci-après).

Dans de nombreux cas, il existe une simple relation entre les diagrammes de la grammaire graphique concrète et les définitions de la syntaxe abstraite. Lorsque le nom d'un non-terminal commence dans la grammaire concrète par le mot <<diagramme>> et qu'il y a un nom dans la grammaire abstraite qui en diffère seulement parce qu'il commence par le mot

*d'efinition*, les deux règles représentent la même notion. Par exemple, <diagramme de système> dans la grammaire concrète correspond à la *D'efinition-de-système* dans la grammaire abstraite.

L'expansion dans la syntaxe concrète provenant de constructions telles les définitions différées (§ 2.4.1), les macros (§ 4.2) et les mises en correspondance de littéraux (§ 5.4.1.15) etc., doit être examinée avant la mise en correspondance entre la syntaxe concrète et la syntaxe abstraite.

### *Sémantique*

Des propriétés sont utilisées dans les règles de bonne formation qui font intervenir soit le type soit d'autres types qui se réfèrent à ce type.

Un exemple de propriété est l'ensemble des identificateurs de signaux d'entrée valides d'un processus. Cette propriété est utilisée dans la condition statique <<pour chaque noeud-d'état, tous les *identificateurs de signaux d'entrée* (dans l'ensemble des signaux d'entrée valides) apparaissent soit dans un *ensemble de signaux mis en réserve* ou dans un *noeud d'entrée* >>.

Toutes les instances ont une propriété d'identité mais à moins que celle-ci soit formée d'une façon quelque peu inhabituelle, cette propriété d'identité est déterminée comme étant définie par la section générale traitant des identités au § 2. Par conséquent, cela n'est pas habituellement mentionné comme étant une propriété d'identité. Il n'est également pas nécessaire d'indiquer les sous-composantes d'une définition contenues par la définition étant donné que l'appartenance de telles sous-composantes est évidente à partir de la syntaxe abstraite. Par exemple, il est évident qu'une définition de bloc <<a>> englobe des définitions de processus et éventuellement une définition de sous-structure de bloc.

Les propriétés sont statiques, si elles peuvent être déterminées sans l'interprétation d'une spécification de système en LDS et sont dynamiques si l'interprétation de ce système est nécessaire pour déterminer la propriété.

L'interprétation est écrite de manière opérationnelle. Lorsqu'il y a une liste dans la syntaxe abstraite, cette liste est interprétée dans l'ordre donné. C'est-à-dire la Recommandation décrit comment les instances sont créées à partir de la définition du système et comment celles-ci sont interprétées dans une <<machine abstraite LDS>>.

Les conditions dynamiques sont des conditions qui doivent être satisfaites durant l'interprétation et qui ne peuvent être vérifiées sans interprétation. Les conditions dynamiques peuvent conduire à des erreurs (voir le § 1.3.3).

### *Modèle*

Certaines constructions sont considérées comme étant une <<syntaxe concrète dérivée>> (ou une abréviation) pour d'autres constructions équivalentes en syntaxe concrète. Par exemple, l'omission d'une entrée pour un signal est une syntaxe concrète dérivée pour une entrée pour ce signal suivi par une transition nulle avec retour vers le même état.

Dans certains cas, une telle <<syntaxe concrète dérivée>>, si elle est étendue, donnera lieu à une représentation immensément grande (éventuellement infinie). Néanmoins, la sémantique d'une telle spécification peut être déterminée.

### *Exemples*

L'intitulé *exemples* contient des exemples.

#### *1.5 Métalangages*

Pour la définition des propriétés et des syntaxes du LDS, différents métalangages ont été utilisés en fonction des besoins particuliers.

Dans ce qui suit, on trouvera une introduction aux métalangages; des références renvoyant à des livres ou à des publications particulières de l'UIT seront données au besoin.

### 1.5.1 Le *Méta IV*

Le sous-ensemble suivant du *Méta IV* est utilisé pour écrire la syntaxe abstraite du LDS.

Une définition dans la syntaxe abstraite peut être considérée comme étant un objet composite nommé (une arborescence) définissant un ensemble de sous-composantes.

Par exemple, la syntaxe abstraite pour la définition d'une variable est

*Définition-de-variable* :: *Nom-de-variable* *Identificateur-de-référence-de-sort*

qui définit le domaine de l'objet composite (arborescence) appelé *définition-de-variable*. Cet objet comporte deux sous-composantes qui à leur tour peuvent être des arborescences.

La définition en *Méta IV*

*Identificateur-de-référence-de-sort* = *Identificateur*

indique qu'un *identificateur-de-référence-de-sort* est un *identificateur* et ne peut par conséquent être syntaxiquement distingué des autres identificateurs.

.bp

Certains objets peuvent également être constitués par certains domaines élémentaires (non composites). Dans le cas du LDS, ces objets sont:

#### a) Des objets entiers

Exemple:

*Nombre-d'instances* :: *entier entier*

Le terme *nombre-d'instances* désigne un domaine composite contenant deux des valeurs entières (*entier*) indiquant le nombre initial et le nombre maximal d'instances.

#### b) Mots clés

Les mots clés sont représentés par une séquence en caractères gras de majuscules et de chiffres.

Exemple:

*Processus-de-destination* = *Identificateur-de-processus* | **ENVIRONNEMENT**

Le *processus-de-destination* est soit un *identificateur-de-processus* soit l'environnement qui est désigné par le mot clé **ENVIRONNEMENT**.

#### c) Marques

Le terme *Token* désigne le domaine des marques. Ce domaine peut être considéré comme étant composé d'un ensemble potentiellement infini d'objets atomiques distincts pour lesquels aucune représentation n'est requise.

Exemple:

*Nom* :: *Token*

Un nom est un objet atomique tel que tout nom peut être distingué de tout autre nom.



d) Objets non spécifiés

Un objet non spécifié désigne des domaines qui peuvent avoir une certaine représentation, mais pour lesquels la représentation n'intéresse pas la présente Recommandation.

Exemple:

*Texte-informel* :: ...

*Texte-informel* contient un objet qui n'est pas interprété.

Les opérateurs ci-après (constructeurs) dans la forme BNF (voir le § 1.5.2) sont également utilisés dans la syntaxe abstraite: <<\*>> pour désigner une liste pouvant être vide; <<+>> pour désigner une liste non vide; < | > pour représenter une alternative, et <<[<<>>]>> pour indiquer une option.

Les parenthèses sont utilisées pour regrouper les domaines qui présentent un rapport logique.

Enfin, la syntaxe abstraite utilise un autre opérateur de suffixe <<-set >> produisant un ensemble (collection non ordonnée d'objets distincts).

Exemple:

*graphe-de-processus* :: *noeud-de-départ-de-processus* *noeud-d'état* **-set**

Un *graphe-de-processus* est constitué par un *noeud-de-départ-de-processus* et d'un ensemble de *noeud-de-processus*.

### 1.5.2 Backus-Naur Form

Dans la Backus-Naur Form (BNF), un symbole terminal est soit celui qui n'est pas mis entre crochets angulaires (c'est-à-dire le signe inférieur à ou le signe supérieur à <et>) ou est l'une des deux représentations <nom> et <chaîne de caractères>. Notons que les deux terminaux spéciaux <nom> et <chaîne de caractères> peuvent avoir également une sémantique telle que celle qui est définie ci-après.

Les crochets angulaires et le(s) mot(s) sont soit un symbole non-terminal ou l'un des deux terminaux <chaîne de caractères> ou <nom>. Les catégories syntaxiques sont les non-terminaux indiqués par un ou plusieurs mots compris entre des crochets angulaires. Pour chaque symbole non-terminal, une règle de production est donnée soit en grammaire textuelle concrète soit en grammaire graphique. Par exemple:

<expression de visibilité> ::=

VIEW (<identificateur de variable>, <expression>)

Une règle de production d'un symbole non-terminal consiste à placer le symbole non-terminal sur la partie gauche du symbole ::= et, sur la partie droite une ou plusieurs constructions constituées par un ou plusieurs symbole(s) non-terminaux et éventuellement un ou plusieurs symbole(s) terminaux. Par exemple <expression de visibilité>, <identificateur de variable> et <expression> dans l'exemple ci-dessus sont des symboles non-terminaux; VIEW, les parenthèses et le point sont des symboles terminaux.

.bp

Parfois, le symbole inclut une partie soulignée. Cette partie soulignée met en relief un aspect sémantique de ce symbole. Par exemple <identificateur de variable> est syntaxiquement identique à <identificateur>, mais sur le plan sémantique, il spécifie que l'identificateur doit être un identificateur de variable.

A la partie droite du symbole ::=, il existe plusieurs possibilités de production de symboles non-terminaux, séparées par des barres verticales |. Par exemple:

<zone de bloc> ::=

<référence de bloc graphique>

| <diagramme de bloc>

indique qu'une <zone de bloc> est soit une <référence de bloc graphique> ou un <diagramme de bloc>.

Les éléments syntaxiques peuvent être regroupés au moyen d'accolades { | t }, analogues aux parenthèses du Méta IV (voir le § 1.5.1). Un groupe entre accolades peut contenir une ou plusieurs barres verticales, indiquant des éléments syntaxiques possibles. Par exemple:

<zone d'interaction de bloc> ::=

{ zone de bloc | <zone de définition de canal >

La répétition de groupes entre accolades est indiquée au moyen d'un astérisque (\*) ou du signe plus (+). Un astérisque indique que le groupe est facultatif et peut ultérieurement être répété un nombre quelconque de fois; un signe plus indique que le groupe doit être présent et peut être répété par la suite un nombre quelconque de fois. L'exemple ci-dessus indique qu'une <zone d'interaction de bloc> contient au moins une <zone de bloc> ou une <zone de définition de canal> et peut contenir plusieurs autres <zones de bloc> et <zones de définition de canal>.

Si les éléments syntaxiques sont regroupés en utilisant des crochets ([ | t |]), cela indique que le groupe est facultatif. Par exemple:

<en-t | te de processus> ::=

PROCESS <identificateur de processus > [<paramètres formels>]

indique qu'un <en-t | te de processus> peut, mais pas nécessairement, contenir des <paramètres formels>.

### 1.5.3 Métalangage applicable à la grammaire graphique

En ce qui concerne la grammaire graphique, le métalangage décrit au § 1.5.2 est complété avec les métasymboles suivants:

- a) **contains**
- b) **is associated with**
- c) **is followed by**
- d) **is connected to**
- e) **set**

Le métasymbole **set** est un opérateur postfixé agissant sur les éléments syntaxiques placés à l'intérieur d'accolades et qui le précède immédiatement, il désigne un ensemble (non ordonné) d'éléments. Chaque élément peut être un groupe quelconque d'éléments syntaxiques, auquel cas, il faut le développer avant d'appliquer le métasymbole **set**.

Exemple:

{ zone de texte de système } { diagramme de macro } <zone d'interaction de blocs > **set**

est un ensemble de zéro, d'une ou de plusieurs <zone de texte de système>; de zéro, d'un ou plusieurs <diagramme de macro> et une <zone d'interaction de blocs>.

Tous les autres métasymboles sont des opérateurs infixes, ayant un symbole graphique non-terminal comme argument de gauche. L'argument de droite est soit un groupe d'éléments syntaxiques situés à l'intérieur d'accolades ou un seul élément syntaxique. Si le membre de droite d'une règle de production comporte un symbole graphique non-terminal comme premier élément et contient un ou plusieurs de ces opérateurs infixes, le symbole graphique non-terminal est alors l'argument de gauche de chacun de ces opérateurs infixes. Un symbole graphique non-terminal est un nom terminal ayant le mot <<symbole>> placé immédiatement avant le signe <.

Le m'etasymbole **contains** indique que son argument de droite doit  
| tre plac'e à l'int'erieur de son argument de gauche et, le cas 'ech'eant, à l'int'erieur du <symbole d'extension de texte>  
associe'. Par exemple:

<r'ef'erence de bloc graphique> ::=

<symbole de bloc> **contains** <nom de bloc >

<symbole de bloc> ::=

.rs

### Figure, p.

signifie

.rs

### Figure, p.

Le m'etasymbole **is associated with** indique que son argument de droite est logiquement associe' avec son argument de  
gauche (comme s'il 'etait <<contenu>> dans cet argument, l'association d'epourvue d'ambigu"ite est obtenue par des  
r'egles appropri'ees applicables aux dessins).

Le m'etasymbole **is followed by** signifie que son argument de droite suit (tant sur le plan logique que dans le des-  
sin) son argument de gauche.

Le m'etasymbole **is connected to** signifie que son argument de droite est relie (tant sur le plan logique que dans le  
dessin) à son argument de gauche.

.rs

## 2 Le LDS de base

### 2.1 Introduction

Un système LDS possède un ensemble de blocs. Les blocs sont connectés entre eux et à l'environnement par des canaux. A l'intérieur de chacun des blocs il y a un ou plusieurs processus. Ces processus communiquent entre eux par des signaux et sont supposés s'exécuter en parallèle.

Le § 2 a été subdivisé en huit principaux sujets:

a) *Règles générales*

Les concepts de base du LDS tels les règles lexicales et les identificateurs, les règles de visibilité, les textes informels, la subdivision des diagrammes, les règles applicables aux dessins, les commentaires, les extensions de textes, les symboles de texte.

b) *Concepts de base concernant les données*

Les concepts de base concernant les données en LDS telles les valeurs, les variables, les expressions.

c) *Structure des systèmes*

Contient les concepts du LDS relatifs aux principes généraux de structuration du langage. Il s'agit des concepts de système, de bloc, de processus, de procédure.

d) *Communication*

Décrit les mécanismes de communication utilisés dans le LDS tels le canal, l'acheminement du signal, le signal.

e) *Comportement*

Les constructions qui concernent le comportement d'un processus: règles de connectivité générales d'un processus ou graphe de procédure, définition de variable, départ, état, entrée, mise en réserve, étiquette, transition.

f) *Action*

Constructions actives tels les tâches, la création de processus, l'appel de procédure, la sortie, la décision.

g) *Temporisateurs*

Définition des temporisateurs et primitives des temporisateurs.

h) *Exemples*

Il s'agit d'exemples concernant les autres points.

### 2.2 Règles générales

#### 2.2.1 Règles lexicales

Les règles lexicales définissent des unités lexicales. Les unités lexicales sont les symboles terminaux de la syntaxe textuelle concrète.

<unité lexicale> ::=

<nom>

| <cha | ne de caract`eres>

| <sp´ecial>

| <sp´ecial composite>

| <note>

| <mot cl´e>

<nom> ::=

<mot> { | soulign´e> <mot> }

<mot> ::=

{ alphanum´erique> | <point> }

<alphanum´erique>

{ alphanum´erique> | <point> }

<alphanum´erique> ::=

| <lettre>

| <chiffre d´ecimal>

| <national>

<lettre> ::=

A | B | C | D | E | F | G | H | I | J | K | L | M

| N | O | P | Q | R | S | T | U | V | W | X | Y | Z

| a | b | c | d | e | f | g | h | i | j | k | l | m

| n | o | p | q | r | s | t | u | v | w | x | y | z

.bp

<chiffre d´ecimal> ::=

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

<national> ::=

##

| .`

**Montage**

| ○**Montage**

| **Montage**

	<crochet gauche>
	<b>Montage</b>
	<crochet droit>
	<accolade gauche>
	<ligne verticale>
	<accolade droite>
	<tilde>
	<pointe de flèche vers le haut>

<crochet gauche> ::=

```
[
  <crochet droit> ::=
]
<accolade gauche> ::=
{
  <ligne verticale> ::=
|
  <accolade droite> ::=
}
```

<tilde> ::=

. ~

**Montage**

<pointe de flèche vers le haut> ::=

. ^

**Montage**

<point> ::=

.

<souligné> ::=

—

<chaîne de caractères> ::=

```

<apostrophe> { alphanum´erique>
|
|   <autre caract`ere>
|
|   <sp´ecial>
|
|   <point>
|
|   <soulign´e>
|
|   <espace>
|
|   <apostrophe> <apostrophe } <apostrophe>

```

<texte> ::=

```

{ alphanum´erique>
|
|   <autre caract`ere>
|
|   <sp´ecial>
|
|   <point>
|
|   <soulign´e>
|
|   <espace>
|
|   <apostrophe }

```

<apostrophe> ::=

‘

<autre caract`ere> ::=

? | & | %

<sp´ecial> ::=

+ | - | ! | / | > | \* | ( | ) | " | , | ; | < | = | :

.bp

<sp´ecial composite> ::=

==

| ==>

| /=

| <=

| >=

| //

| :=

| =>



| ->  
| (.  
| .)

<note> ::=

/\* <text> \*/

<mot cl'e> ::=

| ACTIVE  
| ADDING  
| ALL  
| ALTERNATIVE  
| AND  
| AXIOMS  
| BLOCK  
| CALL  
| CHANNEL  
| COMMENT  
| CONNECT  
| CONSTANT  
| CONSTANTS  
| CREATE  
| DCL  
| DECISION  
| DEFAULT  
| ELSE  
| ENDALTERNATIVE  
| ENDBLOCK  
| ENDCHANNEL  
| ENDDECISION  
| ENDGENERATOR  
| ENDMACRO  
| ENDNEWTYP

- |     ENDPROCEDURE
- |     ENDPROCESS
- |     ENDREFINEMENT
- |     ENDSELECT
- |     ENDSERVICE
- |     ENDSTATE
- |     ENDSUBSTRUCTURE
- |     ENDSYNTYPE
- |     ENDSYSTEM
- |     ENV
- |     ERROR
- |     EXPORT
- |     EXPORTED
- |     EXTERNAL
- |     FI
- |     FOR
- |     FPAR
- |     FROM
- |     GENERATOR
- |     IF
- |     IMPORT
- |     IMPORTED
- |     IN
- |     INHERITS
- |     INPUT
- |     JOIN

	LITERAL
	LITERALS
	MACRO
	MACRODEFINITION
	MACROID
	MAP
	MOD
	NAMECLASS
	NEWTYPE
	NEXTSTATE
	NOT
	NOW
	OFFSPRING
	OPERATOR
	OPERATORS
	OR
	ORDERING
	OUT
	OUTPUT
	PARENT
	PRIORITY
	PROCEDURE
	PROCESS
	PROVIDED
	REFERENCED
	REFINEMENT
	REM
	RESET
	RETURN
	REVEALED
	REVERSE
	SAVE

	SELECT
	SELF
	SENDER
	SERVICE
	SET
	SIGNAL
	SIGNALLIST
	SIGNALROUTE
	SIGNALSET
	SPELLING
	START
	STATE
	STOP
	STRUCT
	SUBSTRUCTURE
	SYNONYM
	SYNTYPE
	SYSTEM
	TASK
	THEN
	TIMER
	TO
	TYPE
	VIA
	VIEW
	VIEWED
	WITH
	XOR

<espace> représente le caractère espace de l'Alphabet n° 5 du CCITT.

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

.bp

Toutes les <lettre> sont toujours traitées comme des majuscules sauf celles qui sont à l'intérieur d'une <chaîne de caractères>. (Le traitement des <national> peut être défini par les organismes nationaux de normalisation.)

Une <unité lexicale> se termine par le premier caractère qui ne peut pas faire partie de l'<unité lexicale> conformément à la syntaxe spécifiée ci-dessus. Lorsqu'un caractère <souligné> est suivi d'un ou plusieurs caractères de commande (les caractères de commande sont définis comme dans la Recommandation T.50) ou espaces, tous ces caractères (y compris le <souligné>) sont ignorés, par exemple A \_B correspond au même <nom> que AB. Cet emploi de <souligné> permet de répartir des <unités lexicales> sur plus d'une ligne.

Lorsqu'un caractère <souligné> est suivi d'un <mot> dans un <nom>, il est autorisé à spécifier un ou plusieurs caractères de commande ou espaces, au lieu du caractère <souligné>, pour autant que l'un des <mot> englobant le caractère <souligné> ne forme pas un <mot clé>, par exemple A B désigne le même <nom> que A \_B.

Toutefois, dans certains cas, l'absence de <souligné> dans les <nom> est ambiguë. Les règles suivantes s'appliquent donc:

- 1) les <souligné> dans le <nom> dans un <élément de trajet> doivent être spécifiés explicitement;
- 2) lorsqu'un ou plusieurs <nom> ou <identificateur> peuvent être suivis directement par une <sorte> (par exemple, <définition de variable>, <définition de vue>), les <souligné> dans ces <nom> ou <identificateur> doivent être spécifiés explicitement;
- 3) lorsqu'une <définition de données> contient des <instantiations de générateur>, les <souligné> du <nom de sorte> suivant le mot clé NEWTYPE doivent être spécifiés explicitement.

Un caractère de commande a la même signification qu'un espace.

Les caractères de commande et les espaces peuvent apparaître un nombre quelconque de fois entre deux <unités lexicales>. Un nombre quelconque de caractères de contrôle et d'espaces entre deux <unités lexicales> ont la même signification qu'un espace.

Le caractère / immédiatement suivi par le caractère \* marque toujours le début d'une <note>. Le caractère \* immédiatement suivi par le caractère / dans une <note> marque toujours la fin d'une <note>. Une <note> peut être insérée avant ou après toute <unité lexicale>.

Des règles lexicales particulières s'appliquent dans un <corps de macro> (voir le § 4.2.1).

### 2.2.2 Règles de visibilité et identificateurs

#### Grammaire abstraite

Identificateur :: Qualificatif-Nom

Qualificatif = Elément-de-chemin +

Elément-de-chemin = Qualificatif-de-système |

Qualificatif-de-bloc |

Qualificatif-de-sous-structure-de-bloc |

Qualificatif-de-signal |

Qualificatif-de-processus |

*Qualificatif-de-procédure* |

*Qualificatif-de-sortie* |

*Qualificatif-de-système* :: *Nom-de-système*

*Qualificatif-de-bloc* :: *Nom-de-bloc*

*Qualificatif-de-sous-structure-de-bloc* :: *Nom-de-sous-structure-de-bloc*

*Qualificatif-de-processus* :: *Nom-de-processus*

*Qualificatif-de-procédure* :: *Nom-de-procédure*

*Qualificatif-de-signal* :: *Nom-de-signal*

*Qualificatif-de-sortie* :: *Nom-de-sortie*

*Nom* :: *Token*

### *Grammaire textuelle concrète*

<identificateur> ::=

[<qualificatif>] <nom>

<qualificatif> ::=

<élément de chemin> { <élément de chemin >

<élément de chemin> ::=

<classe d'unité de portée> <nom>

.bp

<classe d'unité de portée> ::=

| SYSTEM

| BLOCK

| SUBSTRUCTURE

| SIGNAL

| PROCESS

| PROCEDURE

| TYPE

| SERVICE

Il n'y a pas de syntaxe abstraite correspondante pour la <classe d'unité de portée> indiquée par service. Les <nom> et <identificateur> d'entités définies dans une <définition de service> sont transformées en <nom> uniques ou en <identificateur> uniques définis dans la <définition de processus> contenant la <définition de service>.

Le <qualificatif> reflète la structure hiérarchique à partir du niveau du système vers le contexte de définition, et de manière telle que le niveau du système est la partie textuelle la plus à gauche. Toutefois, on peut omettre certains <éléments de chemin> les plus à gauche, lorsque le premier <élément de chemin> restant le plus à gauche dans le <qualificatif> est unique à l'intérieur de toute la <définition de système>.

Il est permis d'omettre certains <éléments de chemin> les plus à gauche (sauf pour les <définitions différées>, voir le § 2.4.1) ou bien tout le <qualificatif>. Lorsque tout le <qualificatif> est omis et que le <nom> désigne une entité de la classe d'entité contenant des variables, des synonymes, des littéraux et des opérateurs (voir la sémantique ci-après), l'association du <nom> avec une définition doit pouvoir être résolue par le contexte réel. Dans d'autres cas, l'<identificateur> est associée à une entité qui a son contexte de définition dans l'unité de portée englobante la plus proche dans laquelle le <qualificatif> de l'<identificateur> est le même que la partie la plus à droite du <qualificatif> complet désignant cette unité de portée. Si l'<identificateur> ne contient pas de <qualificatif>, la nécessité de mise en correspondance des <qualificatifs> est omise.

Un sous-signal doit être qualifié par ses signaux parents, à moins qu'aucun autre signal visible n'existe à cet endroit qui porte le même <nom>.

La résolution par contexte est possible dans les cas suivants:

a) l'unité de portée dans laquelle le <nom> est utilisé n'est pas une <définition partielle de type> et contient une définition ayant ce <nom>. Le <nom> sera lié à cette définition;

b) l'unité de portée dans laquelle le <nom> est utilisé ne contient pas de définition ayant ce <nom> ou l'unité de portée est une <définition partielle de type>, et dans toute la <définition de système> il existe exactement une définition visible d'une entité qui a le même <nom> et à laquelle le <nom> peut être lié sans violer aucune des propriétés statiques (compatibilité de sorte, etc.) de la construction dans laquelle le <nom> apparaît. Le <nom> sera lié à cette définition.

Seuls les identificateurs visibles peuvent être utilisés, à l'exception de l'<identificateur de variable> dans une <définition de visibilité> et de l'<identificateur> utilisé à la place d'un <nom> dans une définition référencée (c'est-à-dire une définition extraite de la <définition de système>).

### *Sémantique*

Les unités de portée sont définies par le schéma suivant:

*Grammaire textuelle concrète    Grammaire graphique concrète*

<définition de système>            <diagramme de système>

<définition de bloc>            <diagramme de bloc>

<définition de processus>            <diagramme de processus>

<définition de procédure>            <diagramme de procédure>

<définition de sous-structure de bloc>            <diagramme de sous-structure de bloc>

<définition de sous-structure de canal>            <diagramme de sous-structure de canal>

<définition de service>            <diagramme de service>

<définition partielle de type>

<affinage de signal>

.bp

Une liste de définitions est associée à une unité de portée. Chaque définition définit une entité appartenant à une certaine classe d'entité et ayant un nom associé. Pour une <définition partielle de type>, la liste de définitions associée comprend les <signature d'opérateur>, les <signature de littéral> et toutes <signature d'opérateur> et <signature de littéral> provenant d'une sorte parente, d'un générateur d'instance ou imposée par l'utilisation d'abréviations tel le mot d'ORDER-ING (voir le § 5.4.1.8). Il faut remarquer qu'une <définition de visibilité> ne définit pas une entité.

Bien que les <opérateur infixé>, les <opérateur> avec un point d'exclamation et les <chaîne de caractères> aient leur propre notation syntaxique, ils sont en réalité des <nom>, ils sont représentés dans la syntaxe abstraite représentée par un nom suit, ils sont étudiés comme s'ils étaient (syntaxiquement aussi) des <nom>. Toutefois, les <nom d'état>, les <nom de connecteur>, les <nom formel de générateur>, les <identificateur de valeur> dans les équations, les <nom formel de macro> et les <nom de macro> ont des règles de visibilité particulières et ne peuvent par conséquent être qualifiés. Leurs règles de visibilité sont expliquées dans les sections concernées.

Chaque entité est dite avoir son contexte de définition dans l'unité de portée qui la définit. Les entités sont référencées au moyen d'<identificateur>.

Le <qualificatif> dans un <identificateur> spécifie uniquement le contexte de définition du <nom>.

Les classes d'entités sont les suivantes:

- a) système
- b) blocs
- c) canal, acheminement de signaux
- d) signaux, temporisateurs
- e) processus
- f) procédures
- g) variables (y compris les paramètres formels), synonymes, littéraux, opérateurs
- h) sortes
- i) générateurs
- j) entités importées
- k) listes de signaux
- l) services
- m) sous-structures de bloc, sous-structures de canal

Un <identificateur> est dit être visible dans une unité de portée

- a) si la partie nom de l'<identificateur> a son contexte de définition dans cette unité de portée, ou
- b) s'il est visible dans l'unité de portée qui définit cette unité de portée, ou
- c) si l'unité de portée contient une <définition partielle de type> dans laquelle l'<identificateur> est défini,

ou



d) si l'unité de portée contient une <définition de signal> dans laquelle l'<identificateur> se trouve défini.

On ne peut avoir deux définitions dans la même unité de portée et appartenant à la même classe d'entités portant le même <nom>. Il existe cependant une exception: les définitions de <signature d'opérateur> et de <signature de littéral> dans la même <définition partielle de type> (voir le § 5.2.2): plusieurs opérateurs et littéraux peuvent avoir le même <nom> avec différentes <sorte d'argument> ou différentes sortes de <résultat>.

On notera une autre exception: les entités importées. Pour cette classe, les paires de (<nom d'import>, <sorte>) dans <définition d'import> dans l'unité de portée doivent être distinctes.

Dans la grammaire textuelle concrète, le nom ou l'identificateur optionnel dans une définition après le mot clé de terminaison (ENDSYSTEM, ENDBLOCK, etc.) doit être syntaxiquement le même que le nom ou l'identificateur qui suit le mot clé commençant correspondant (respectivement: SYSTEM, BLOCK, etc.).

.bp

### 2.2.3 Texte informel

#### Grammaire abstraite

*Texte informel* :: ...

*Grammaire textuelle concrète*

<texte informel> ::=

<chaîne de caractères>

*Sémantique*

Lorsqu'un texte informel est utilisé dans une spécification LDS de système, il signifie que le texte n'est pas en LDS formel, c'est-à-dire que le LDS ne donne pas de sémantique. La sémantique du texte informel peut être définie par d'autres moyens.

### 2.2.4 Règles applicables aux dessins

La taille des symboles graphiques est choisie par l'utilisateur.

Les frontières des symboles ne doivent ni se superposer, ni se couper. Font exception à cette règle les symboles de ligne, c'est-à-dire le <symbole de canal>, le <symbole d'acheminement de signal>, le <symbole de création de ligne>, le <symbole de ligne de flot>, le <symbole d'association continu> et le <symbole d'association pointillé>, qui peuvent se couper. Il n'existe pas d'association logique entre les symboles qui se coupent.

Le métasymbole **is followed by** implique un <symbole de ligne de flot>.

Les symboles de ligne peuvent être constitués par un ou plusieurs segments de droite en trait plein.

Les flèches sont nécessaires chaque fois qu'un <symbole de ligne de flot> entre dans un autre <symbole de ligne de flot>, dans un <symbole de connecteur de sortie> ou dans un <symbole d'état suivant>. Dans d'autres cas, ces flèches sont facultatives sur les <symbole de ligne de flot>. Les <symbole de ligne de flot> sont horizontaux ou verticaux.

On peut utiliser des images symétriques verticales des <symbole d'entrée>, des <symbole de sortie>, des <symbole de commentaire> et des <symbole d'extension de texte>.

L'argument de la partie droite du métasymbole **is associated with** doit être plus proche de l'argument de gauche que tout autre symbole graphique. Les éléments syntaxiques de l'argument de droite doivent pouvoir

|tre distingués les uns des autres.

Le texte situé à l'intérieur d'un symbole graphique doit |tre lu de la gauche vers la droite, en partant du coin supérieur gauche. La limite droite du symbole est interprétée comme un caractère de nouvelle ligne, indiquant le cas échéant que la lecture doit continuer au point le plus à gauche de la ligne suivante.

### 2.2.5 Subdivision des diagrammes

La définition qui suit concernant la division des diagrammes ne fait pas partie de la *grammaire graphique concrète*, néanmoins, on utilise le même métalangage.

<page> ::=

<symbole de cadre> **contains**

{ zone d'en-t |te> <zone de numéro de page>

{ unité syntaxique } }

<zone d'en-t |te> ::=

<symbole implicite de texte> **contains** <en-t |te>

<zone de numéro de page> ::=

<symbole implicite de texte> **contains** [<numéro de page> [(<nombre de pages>)] |

<numéro de page> ::=

<nom de littéral >

<nombre de pages> ::=

<nom de littéral de naturel >

.bp

La <page> est un non-terminal de départ, par conséquent, il n'est associé à aucune règle de production. Un diagramme peut |tre subdivisé en un nombre de <page>, auquel cas le <symbole de cadre> d'limitant le diagramme et l'<en-t |te> du diagramme est remplacé par un <symbole de cadre> et un <en-t |te> pour chaque <page>.

L'utilisateur du LDS peut choisir les <symbole de cadre> imposés par la limite du support sur lequel les diagrammes sont reproduits.

Afin d'avoir une séparation nette entre la <zone d'en-t |te> et la <zone de numéro de page>, le <symbole de texte implicite> n'est pas matérialisé, il est sous-entendu. La <zone d'en-t |te> est placée au coin supérieur gauche du <symbole de cadre>, la <zone de numéro de page> est située au coin supérieur droit du <symbole de cadre>. L'<en-t |te> et l'<unité syntaxique> dépendent du type de diagramme.

### 2.2.6 Commentaire

Un commentaire est une notation qui représente des commentaires associés avec des symboles ou du texte.

Dans la *grammaire textuelle concrète*, on utilise deux formes de commentaires. La première forme est la <note> définie au § 2.2.1.

Des exemples sont donnés aux figures 2.9.1 et 2.9.3.

La syntaxe concrète de la deuxième forme est:

<fin> ::=

[<commentaire>];

<commentaire> ::=

COMMENT <chaîne de caractères>

Un exemple est donné à la figure 2.9.2.

Dans la *grammaire graphique concrète*, la syntaxe suivante est utilisée:

<zone de commentaire> ::=

<symbole de commentaire> **contains** <texte>

**is connected to** <symbole d'association pointillé>

<symbole de commentaire> ::=

.rs

#### Figure, p. 29 (E)

<symbole d'association pointillé> ::=

-----

Une des extrémités du <symbole d'association pointillé> doit être reliée au milieu du segment vertical du <symbole de commentaire>.

Un <symbole de commentaire> peut être relié à tout symbole graphique au moyen d'un <symbole d'association pointillé>. Le <symbole de commentaire> est considéré comme étant un symbole fermé en complétant (par la pensée) le rectangle afin d'entourer le texte. Il contient le texte du commentaire se rapportant au symbole graphique.

Un exemple est donné à la figure 2.9.4 dans le § 2.9.

### 2.2.7 Extension de texte

<zone d'extension de texte> ::=

<symbole d'extension de texte> **contains** <texte>

**is connected to** <symbole d'association continu>

<symbole d'extension de texte> ::=

<symbole de commentaire>

<symbole d'association continu> ::=

Une des extrémités du <symbole d'association continu> doit être reliée au milieu du segment vertical du <symbole d'extension de texte>.

Un <symbole d'extension de texte> peut être relié à tout symbole graphique au moyen d'un <symbole d'association continu>. Le <symbole d'extension de texte> est considéré comme étant un symbole fermé en complétant (par la pensée) le rectangle.

Le texte contenu dans le symbole d'extension de texte> est la suite du texte dans le symbole graphique et est considéré comme étant contenu dans ce symbole.

### 2.2.8 Symbole de texte

Le <symbole de texte> est utilisé dans tout <diagramme>. Le contenu dépend du diagramme.

<symbole de texte> ::=

.rs

## Figure, p. 30 (E)

### 2.3 Concepts de base concernant les données

Le concept de données dans le LDS est exposé dans le détail au § 5; plus précisément en ce qui concerne la terminologie du LDS pour ce qui est des données et la facilité à définir des nouveaux types de données et les facilités concernant les données prédéfinies.

Les données apparaissent dans les définitions du type de données, les expressions, l'application d'opérateurs, les variables, les valeurs et les littéraux.

#### 2.3.1 Définitions des types de données

Les données dans le LDS sont principalement abordées sous l'aspect type de données. Un type de données définit un ensemble de valeurs, un ensemble d'opérateurs qui peut être appliqué à ces valeurs, et un ensemble de règles algébriques (équations) qui définissent le comportement de ces opérateurs lorsqu'ils sont appliqués aux valeurs. Les valeurs, les opérateurs et les règles algébriques définissent tous ensemble les propriétés du type de données. Ces propriétés sont définies par les définitions de types de données.

Le LDS permet la définition de tout type de données qui est nécessaire, y compris les mécanismes de composition (type composite) avec pour seule contrainte qu'une telle définition puisse être spécifiée de manière formelle. En revanche, pour les langages de programmation, il y a des considérations de mise en oeuvre qui nécessitent que l'ensemble des types de données disponibles et, en particulier, les mécanismes de composition mis à disposition soient limités (tableau, structure, etc.).

### 2.3.2 Variables

Les variables sont des objets qui peuvent être associés à une valeur par une affectation. Quand on accède à la variable, on renvoie la valeur associée.

### 2.3.3 Valeurs et littéraux

Un ensemble de valeurs avec certaines caractéristiques est appelé sorte. Les opérateurs sont définis à partir et vers les valeurs des sortes. Par exemple, l'application de l'opérateur somme ( $\ll + \gg$ ) à partir et vers les valeurs de la sorte entière est valable, tandis que la somme de la sorte booléenne ne l'est pas.

.bp

Toutes les sortes ont au moins une valeur. Chaque valeur appartient à une sorte unique, c'est-à-dire que les sortes n'ont jamais de valeurs en commun.

Pour la plupart des sortes, il y a des formes littérales qui décrivent les valeurs de la sorte (par exemple, pour les entiers  $\ll 2 \gg$  est utilisé de préférence à  $\ll 1+1 \gg$ ). Il peut y avoir plusieurs littéraux qui décrivent la même valeur (par exemple, 12 et 012 peuvent être utilisés pour écrire la même valeur entière). La même description littérale peut être utilisée pour plus d'une sorte; par exemple, 'A' est à la fois un caractère et une chaîne de caractères de longueur 1. Certaines sortes peuvent ne pas avoir de littéraux; par exemple, une valeur composite n'a souvent pas de littéraux en propre mais a des valeurs qui sont définies par des opérations de composition sur les valeurs de ses composantes.

### 2.3.4 Expressions

Une expression décrit une valeur. Si une expression ne contient pas de variable, par exemple, si elle est un littéral d'une sorte donnée, chaque occurrence de l'expression écrira toujours la même valeur. Une expression qui contient des variables peut être interprétée comme différentes valeurs durant l'interprétation d'un système LDS selon la valeur associée aux variables.

## 2.4 Structure du système

### 2.4.1 Définitions différées

Une <définition différée> est une définition qui a été supprimée de son contexte de définition pour accroître la visibilité. Elle est analogue à

une définition de macro (voir le § 4.2), mais elle est <appelée> d'un seul endroit exactement (le contexte de définition) en utilisant une référence.

#### Grammaire concrète

<définition différée> ::=

<définition> | <diagramme>

<définition de système> ::=

{ définition textuelle de système } | <diagramme de système>

{ définition différée }

<définition> ::=

<définition de bloc>

| <définition de processus>

| <définition de procédure>

| <définition de sous-structure de bloc>

| <définition de sous-structure de canal>

| <définition de service>

| <définition de macro>

<diagramme> ::=

<diagramme de bloc>

| <diagramme de processus>

| <diagramme de procédure>

| <diagramme de sous-structure de bloc>

| <diagramme de sous-structure de canal>

| <diagramme de service>

| <diagramme de macro>

Pour chaque <définition différée>, sauf pour des <définition de macro> et des <diagramme de macro>, on doit avoir une référence dans la <définition de système>, le <diagramme de système>, ou une autre <définition différée>.

Pour chaque référence on doit avoir une <définition différée> correspondante.

Dans chaque <définition différée>, on doit avoir un <identificateur> placé immédiatement après le mot d'é initial. Dans cet <identificateur>, le <qualificatif> doit être soit complet, soit omis. Si le <qualificatif> est omis, le <nom> doit être unique dans la définition de système, à l'intérieur de la classe d'entité pour la <définition différée>. On ne peut pas spécifier un qualificatif dans l'<identificateur> après le mot d'é initial pour les définitions qui ne sont pas des <définition différée> (c'est-à-dire qu'un <nom> doit être spécifié pour les définitions normales).

### *Sémantique*

Avant de pouvoir analyser une <définition de système concret>, chaque référence doit être remplacée par la <définition différée> correspondante. Dans cette substitution, l'<identificateur> de la <définition différée> est remplacé par le <nom> dans la référence.

.bp

## 2.4.2 *Système*

### *Grammaire abstraite*

*Définition-de-système* ::= *Nom-de-système*

*Définition-de-bloc* -**set**

*Définition-de-canal* -**set**

*Définition-de-signal -set*

*Définition-de-type-de-données*

*Définition-de-type-de-synonyme -set*

*Nom-de-système* = *Nom*

Une *définition-de-système* a un nom qui peut être utilisé dans les qualificatifs.

La *définition-de-système* doit au moins contenir une *définition-de-bloc*.

Les définitions de tous les signaux, canaux, types de données, types de synonymes utilisés dans l'interface avec l'environnement et entre les blocs du système sont contenues dans la *définition-de-système* données prédéfinies sont considérées comme étant définies au niveau du système.

*Grammaire textuelle concrète*

<définition textuelle de système> ::=

SYSTEM <nom de système > <fin>

{ définition de bloc>

| <référence textuelle de bloc>

| <définition de canal>

| <définition de signal>

| <définition de liste de signaux>

| <définition de sélection>

| <définition de macro>

| <définition de données } |

ENDSYSTEM [ <nom de système > ] <fin>

<référence textuelle de bloc> ::=

BLOCK <nom de bloc > REFERENCED <fin>

<définition de sélection> est définie au § 4.3.3, <définition de macro> au § 4.2, <définition de données> au § 5.5.1, <définition de bloc> au § 2.4.3, <définition de canal> au § 2.5.1, <définition de signal> au § 2.5.4, <définition de liste de signaux> au § 2.5.5.

Un exemple de <définition de système> est donné à la figure 2.9.5 du § 2.9.

*Grammaire graphique concrète*

<diagramme de système> ::=

<symbole de cadre> **contains**

{ en-tête de système>

{ { zone texte de système } |

```

{ diagramme de macro }

{ <zone interaction de bloc } set }

<symbole de cadre> ::=

```

**Figura 6, p.**

```

<en-t | te de syst`eme> ::=

    SYSTEM <nom de syst`eme >

    <zone de texte de syst`eme> ::=

    <symbole de texte> contains

    { d`efinition de signal>

    |    <d`efinition de liste de signaux>

    |    <d`efinition de donn`ees>

    |    <d`efinition de macro>

    |    <d`efinition de s`election } |

```



```

<zone interaction de bloc> ::=
    { zone de bloc>
    |      <zone de d'efinition de canal }
    <zone de bloc> ::=
    <r'ef'erece de bloc graphique>
    |      <diagramme de bloc>
    <r'ef'erece de bloc graphique> ::=
    <symbole de bloc> contains <nom de bloc >

```

```

<symbole de bloc> ::=

```

#### Figura 7, p.

<d'efinition de s'election> est d'efinie au § 4.3.3, <d'efinition de macro> au § 4.2, <diagramme de macro> au § 4.2, <d'efinition de donn'ees> au § 5.5.1, <diagramme de bloc> au § 2.4.3, <zone de d'efinition de canal> au § 2.5.1, <d'efinition de signal> au § 2.5.4, <d'efinition de liste de signaux> au § 2.5.5.

La *d'efinition-de-bloc -set* dans la *grammaire abstraite* correspond aux <zone de bloc>, la *d'efinition-de-canal- set* à la <zone de d'efinition de canal>.

Un exemple d'un <diagramme de syst'eme> est donn'e à la figure 2.9.6.

#### S'emantique

Une *d'efinition-de-syst'eme* est la repr'esentation en LDS d'une sp'ecification ou de la description d'un syst'eme.

Un syst'eme est s'epar'e de son environnement par une fronti'ere de syst'eme et contient un ensemble de blocs. La communication entre le syst'eme et l'environnement, ou entre les blocs int'erieurs au syst'eme ne peut se faire qu'au moyen de signaux. A l'int'erieur d'un syst'eme, ces signaux sont v'ehicul'es dans des canaux. Les canaux relient les blocs entre eux ou à la fronti'ere du syst'eme.

Avant d'interpr'eter une *d'efinition-de-syst'eme*, on choisit un sous-ensemble coh'erent (voir le § 3.2.1). Ce sous-ensemble est appel'e une instance de la *d'efinition-de-syst'eme*. Une instance de syst'eme est une instanciation d'un type de syst'eme d'efini par une *d'efinition-de-syst'eme*. L'interpr'etation d'une instance d'une *d'efinition-de-syst'eme* est r'ealis'ee par une machine abstraite LDS qui en cons'equene donne la s'emantique aux concepts du LDS. Pour interpr'eter une instance d'une *d'efinition-de-syst'eme*, il faut:

- a) initialiser le temps du syst'eme,
- b) interpr'eter les blocs et les canaux qui leur sont reli'es et qui sont contenus dans le sous-ensemble de subdivision coh'erent choisi.

#### 2.4.3 Bloc

##### Grammaire abstraite

*D'efinition-de-bloc* :: *Nom-de-bloc*

*Définition-de-processus* -set

*Définition-de-signal* -set

*Connexion-de-canal-vers-acheminement* -set

*Définition-d'acheminement-de-signal* -set

*Définition-de-type-de-données*

*Définition-de-type-de-synonyme* -set

[*Définition-de-sous-structure-de-bloc* ]

*Nom-de-bloc* = *Nom*

A moins qu'une *définition-de-bloc* ne contienne une *définition-de-sous-structure-de-bloc* , on doit avoir au moins une *définition-de-processus* et une *définition-d'acheminement-de-signal* à l'intérieur du bloc.

Il est possible de subdiviser les blocs en spécifiant la *définition-de-sous-structure-de-bloc* ; cet aspect du langage est étudié au § 3.2.2.

.bp

*Grammaire textuelle concrète*

<définition de bloc> ::=

BLOCK { nom de bloc  
| identificateur de bloc  
} fin>

{ définition de signal>

| <définition de liste de signaux>

| <définition de processus>

| <référence textuelle de processus>

| <définition d'acheminement de signal>

| <définition de macro>

| <définition de données>

| <définition de sélection>

| <connexion de canal vers acheminement >

[<définition de sous-structure de bloc | référence textuelle de sous-structure de bloc>]

ENDBLOCK [<nom de bloc  
| identificateur de bloc >] <fin>

<référence textuelle de processus> ::=

PROCESS <nom de processus >[<nombre d'instances>] REFERENCED <fin>

<définition de signal> est définie au § 2.5.4, <définition de liste de signaux> au § 2.5.5, <définition de processus> au § 2.4.4, <définition d'acheminement de signal> au § 2.5.2, <connexion de canal vers acheminement> au § 2.5.3,

<définition de sous-structure de bloc> au § 3.2.2, <référence de sous-structure de bloc textuelle> au § 3.2.2, <définition de macro> au § 4.2.2, <définition de données> au § 5.5.1.

Un exemple de <définition de bloc> est montré à la figure 2.9.7 au § 2.9.

*Grammaire graphique concrète*

<diagramme de bloc> ::=

<symbole de cadre>

**contains** { en-t | te de bloc }

{ { zone texte de bloc } { diagramme de macro }

[<zone d'interaction de processus>] [<zone de sous-structure de bloc> ] **set** }

**is associated with** { identificateur de canal

}

L'<identificateur de canal > identifie un canal relié à un acheminement de signaux dans le <diagramme de bloc>. Il se trouve placé à l'extérieur du <symbole de cadre>, à proximité de l'extrémité du trajet du signal arrivant au <symbole de cadre>. Si le <diagramme de bloc> ne contient pas une <zone d'interaction de processus>, il doit alors contenir une <zone de sous-structure de bloc>.

<en-t | te de bloc> ::=

**BLOCK** { nom de bloc

| identificateur de bloc

}

<zone texte de bloc> ::=

<zone texte de système>

<zone d'interaction de processus> ::=

{ zone de processus }

| <zone de ligne de création>

| <zone de définition d'acheminement de signal }

<zone de processus> ::=

<référence graphique de processus | diagramme de processus>

<référence graphique de processus> ::=

<symbole de processus> **contains** { nom de processus > [<nombre d'instances> ]

<symbole de processus> ::=

## Figura 8, p.

Le terme <nombre d'instances> est défini au § 2.4.4.

<zone de ligne de création> ::=

<symbole de ligne de création>

**is connected to** { zone de processus> <zone de processus }

<symbole de ligne de création> ::=

-----

La tête de flèche placée sur le <symbole de ligne de création> indique la <zone de processus> sur laquelle le processus de création a lieu.

Le <diagramme de processus> est défini au § 2.4.4, la <zone de définition d'acheminement de signal> au § 2.5.2, la <zone de sous-structure de bloc> au § 3.2.2, le <diagramme de macro> au § 4.2.2.

Un exemple de <diagramme de bloc> est donné à la figure 2.9.8 au § 2.9.

### *Sémantique*

Une définition de bloc contient une ou plusieurs définitions d'un système et éventuellement une sous-structure de bloc. La définition de bloc permet de regrouper les processus qui tous ensemble assurent une certaine fonction, soit directement soit par l'intermédiaire d'une sous-structure de bloc.

Une définition de bloc assure une interface statique de communication par laquelle les processus peuvent communiquer avec d'autres processus. De plus, elle donne la portée pour les définitions de processus.

Interpréter un bloc consiste à créer les processus initiaux dans le bloc.

#### 2.4.4 *Processus*

### *Grammaire abstraite*

*Définition-de-processus* :: *Nom-de-processus*

*Nombre-d'instances*

*Paramètre-formel-de-processus* |

*Définition-de-procédure-set*

*Définition-de-signal-set*

*Définition-de-type-de-signal*

*Définition-de-type-de-synonyme-set*

*Définition-de-variable-set*

*Définition-de-visibilité-set*

*Définition-de-temporisateur-set*

*Graphe-de-processus*

*Nombre-d'instances* :: *Entier Entier*

*Nom-de-processus* = *Nom*

*Graphe-de-processus* :: *Noeud-de-départ-de-processus*

*Noeud-d'état-set*

*Paramètre-formel-de-processus* :: *Nom-de-variable*

*Identificateur-de-référence-de-sorte*

### *Grammaire textuelle concrète*

<définition de processus> ::=

```

PROCESS { identificateur de processus
| nom de processus
}

```

```

[<nombre d'instances>] <fin>

```

```

[<param`etres formels> <fin>] [<ensemble des signaux d'entr`ee valides>]

```

```

{ d`efinition de signal>

```

```

|      <d`efinition de liste de signaux>

```

```

|      <d`efinition de proc`edure>

```

```

|      <r`ef`erence textuelle de proc`edure>

```

```

|      <d`efinition de macro>

```

```

|      <d`efinition de donn`ees>

```

```

|      <d`efinition de variable>

```

```

|      <d`efinition de visibilit`e>

```

```

|      <d`efinition de s`election>

```

```

|      <d`efinition d'import>

```

```

|      <d`efinition de temporisateur } |

```

```

{ corps de processus>

```

```

|      <d`ecomposition de service }

```

```

ENDPROCESS [<nom de processus
| identificateur de processus >] <fin>

```

```

<r`ef`erence textuelle de proc`edure> ::=

```

```

PROCEDURE <nom de proc`edure > REFERENCED <fin>

```

```

<ensemble de signaux d'entr`ee valides> ::=

```

```

SIGNALSET [<liste de signaux>] <fin>

```

```

<corps de processus> ::=

```

```

<d`epart { `etat } |

```

```

.bp

```

```

<param`etres formels> ::=

```

```

FPAR <nom de variable
{ <nom de variable
} | <sorte>

```

```

{ <nom de variable
{ <nom de variable
} <sorte }

```

<nombre d'instances> ::=

[(<nombre initial>], [<nombre maximal>])

<nombre initial> ::=

<expression simple de naturel >

<nombre maximal> ::=

<expression simple de naturel >

Le nombre initial d'instances et le nombre maximal d'instances contenus dans le nombre-d'instances sont tirés du <nombre d'instances>. Si le <nombre initial> est omis, le <nombre initial> est 1. Si le <nombre maximal> est omis, le <nombre maximal> n'est pas limité.

Le <nombre d'instances> utilisé dans la dérivation est le suivant:

a) s'il n'y a pas de <référence textuelle de processus> pour le processus, le <nombre d'instances> dans la <définition de processus> est utilisé. S'il ne contient pas un <nombre d'instances>, on utilise le <nombre d'instances> où le <nombre initial> et le <nombre maximal> sont omis;

b) si à la fois le <nombre d'instances> dans la <définition de processus> et le <nombre d'instances> dans une <référence textuelle de processus> sont omis, on utilise le <nombre d'instances> où sont omis à la fois le <nombre initial> et le <nombre maximal>;

c) si soit le <nombre d'instances> dans la <définition de processus> soit le <nombre d'instances> dans une <référence textuelle de processus> est omis, le <nombre d'instances> est celui qui est présent;

d) si à la fois le <nombre d'instances> dans la <définition de processus> et le <nombre d'instances> dans une <référence textuelle de processus> sont spécifiés, les deux <nombre d'instances> doivent être égaux lexicalement et ce <nombre d'instances> est utilisé.

Une relation analogue s'applique au <nombre d'instances> spécifiée dans le <diagramme de processus> et dans la <référence textuelle de processus> définis ci-dessous.

<définition de signal> est définie au § 2.5.4, <définition de liste de signaux> au § 2.5.5, <définition de visibilité> au § 2.6.1.2, <définition de variable> au § 2.6.1.1, <définition de procédure> au § 2.4.5, <définition de temporisateur> au § 2.8, <définition de macro> au § 4.2.2, <définition d'import> au § 4.1.3, <définition de sélection> au § 4.3.3, <expression simple> au § 4.3.2, <décomposition de service> au § 4.10.1, <définition de données> au § 5.5.1.

Le <nombre initial> d'instances doit être inférieur ou égal au <nombre maximal> et le <nombre maximal> doit être supérieur à zéro.

L'utilisation du terme <ensemble des signaux d'entrée valides> est précisée au § 2.5.2, *modèle*.

Un exemple de <définition de processus> est donné à la figure 2.9.9 du § 2.9.

*Grammaire graphique concrète*

<diagramme de processus> ::=

<symbole de cadre>

**contains** { en-t | te de processus>

{ zone de texte de processus }

{ zone de processus }

{ diagramme de macro }

{ zone graphique de processus | zone d'interaction de service } **set** }

**[is associated with** { identificateur d'acheminement de signal

} ]

L' <identificateur d'acheminement de signal > identifie le trajet externe d'un signal relié à un trajet de signal dans le <diagramme de processus>. Il est placé à l'extérieur du <symbole de cadre> à proximité de l'extrémité du trajet du signal au <symbole de cadre>.



<zone de texte de processus> ::=

<symbole de texte> **contains** {

[<ensemble des signaux d'entrée valides>]

{ définition de signal>

| <définition de liste de signaux>

| <définition de variable>

| <définition de visibilité>

| <définition d'import>

| <définition de données>

| <définition de macro>

| <définition de temporisateur>

| <définition de sélection } }

<en-tête de processus> ::=

PROCESS { nom de processus

| identificateur de processus

}

[<nombre d'instances> [<fin>] ]

[<paramètres formels>]

<zone graphique de processus> ::=

<zone de départ { zone d'état | zone de connecteur d'entrée }

<définition de signal> est définie au § 2.5.4, <définition de liste de signaux> au § 2.5.5, <définition de visibilité> au § 2.6.1.2, <définition de variable> au § 2.6.1.1, <zone de procédure> au § 2.4.5, <définition de temporisateur> au § 2.8, <définition de macro> et <diagramme de macro> au § 4.2.2, <définition d'import> au § 4.1.3, <définition de sélection> au § 4.3.3, <définition de données> au § 5.5.1, <zone de départ> au § 2.6.2, <zone d'état> au § 2.6.3, <zone de connecteur d'entrée> au § 2.6.6 et <zone d'interaction de service> au § 4.10.1.

Un exemple de <diagramme de processus> est donné à la figure 2.9.10 du § 2.9.

### *Sémantique*

Une définition de processus introduit le type d'un processus qui est destinée à représenter un comportement dynamique.

Dans le *nombre-d'instances*, la première valeur représente le *nombre-d'instances* du processus qui existe à la création du système, la deuxième valeur représente le nombre maximal d'instances simultanées du type de processus.

Une instance de processus est une machine à états finis étendue communicante qui assure un certain nombre d'actions, appelées transitions, suite à la réception d'un signal donné, chaque fois qu'elle se trouve dans un certain état. La réalisation de la transition aboutit à l'attente du processus dans un autre état, qui n'est pas nécessairement différent de l'état d'origine.

Le concept de machine à états finis a été étendu en ce sens que l'état résultant d'une transition, à côté du signal provoquant la transition, peut être affecté par des décisions prises à partir de variables connues du processus.

Plusieurs instances du même type de processus peuvent exister au même instant et s'exécuter de manière asynchrone et en parallèle les unes aux autres, et avec d'autres instances de différents types de processus dans le système.

Lorsqu'un système est créé, les processus initiaux sont créés dans un ordre aléatoire. La communication des signaux entre les processus ne commence que lorsque tous les processus initiaux ont été créés. Les paramètres formels de ces processus initiaux sont initialisés avec une valeur non définie.

Les instances de processus existent à partir du moment où un système est créé ou peuvent être créées par une des actions de demande de création qui lance les processus à interpréter; leur interprétation commence lorsque l'action de départ est interprétée; des actions d'arrêt peuvent arrêter leur existence.

Les signaux reçus par les instances de processus sont appelés signaux d'entrée, et les signaux envoyés aux instances de processus sont appelés signaux de sortie.

Les signaux peuvent être traités par une instance de processus seulement lorsque celle-ci se trouve dans un certain état. L'ensemble complet de signaux d'entrée valides est l'union de l'ensemble des signaux se trouvant dans tous les trajets des signaux conduisant au processus, de l'ensemble de signaux d'entrée valides, des signaux implicites et des signaux de temporisation.

.bp

Un accès d'entrée unique est associé à chaque instance de processus. Lorsqu'un signal d'entrée parvient au processus, il est appliqué à l'accès d'entrée de l'instance de processus.

Le processus peut être soit mis en attente, en occupant un état, soit en activité, en effectuant une transition. Pour chaque état, il existe un ensemble de signaux de mise en réserve (voir également le § 2.6.5). Dans le cas d'attente dans un état, le premier signal entrant dont l'identificateur ne figure pas dans l'ensemble des signaux de mise en réserve est extrait de la file d'attente, et traité par le processus.

L'accès d'entrée peut retenir un nombre quelconque de signaux, de sorte que plusieurs signaux entrants sont mis dans une file d'attente pour le processus. L'ensemble des signaux retenus est ordonné dans la file d'attente, selon l'ordre d'arrivée. Si deux ou plusieurs signaux arrivent simultanément, ils sont ordonnés arbitrairement.

Lorsqu'un processus est créé, on lui attribue un accès d'entrée vide, et il y a alors création de variables locales auxquelles on affecte des valeurs.

Les paramètres formels sont des variables qui sont créées soit lorsque le système est créé (mais aucun paramètre réel ne lui est transmis et par conséquent ces paramètres ne sont pas initialisés), soit lorsque l'instance de processus est dynamiquement créée.

Pour toutes les instances de processus, on peut utiliser quatre expressions produisant un PId (voir le § 5.6.10): SELF, PARENT, OFFSPRING et SENDER. Elles donnent un résultat pour:

- a) l'instance de processus (SELF);
  - b) l'instance du processus créateur (PARENT);
  - c) l'instance de processus la plus récente créée par le processus (OFFSPRING);
  - d) l'instance de processus en provenance de laquelle le dernier signal entrant a été utilisé (SENDER)
- (voir également le § 2.6.4).

Ces expressions sont expliquées dans le détail au § 5.5.4.3.

SELF, PARENT, OFFSPRING et SENDER peuvent être utilisées dans des expressions à l'intérieur des instances de processus.

Pour toutes les instances de processus qui se trouvent présentes au moment de l'initialisation du système, l'expression prédéfinie PARENT présente toujours la valeur NULL.

Pour toutes les instances de processus nouvellement créées, les expressions prédéfinies SENDER et OFFSPRING ont la valeur NULL.

#### 2.4.5 Procédure

Les procédures sont définies au moyen de définitions de procédure. On fait appel à une procédure au moyen d'un appel de procédure faisant référence à la définition de procédure. Des paramètres sont associés à un appel de procédure: on les emploie à la fois pour fournir les valeurs et pour limiter la portée des variables pour l'exécution de la procédure. C'est du mécanisme de transfert des paramètres que dépendent les variables affectées par l'interprétation d'une procédure.

*Grammaire abstraite*

*Définition-de-procédure* :: *Nom-de-procédure*

*Paramètre-formel-de-procédure* |

*Définition-de-procédure-set*

*Définition-de-type-de-données*

*Définition-de-type-de-synonyme-set*

*Définition-de-variable-set*

*Graphe-de-procédure*

*Nom-de-procédure* = *Nom*

*Paramètre-formel-de-procédure* = *Paramètre-d'entrée* |

*Paramètre-d'entrée-et-de-sortie*

*Paramètre-d'entrée* :: *Nom-de-variable*

*Identificateur-de-référence-de-sortie*

*Paramètre-d'entrée-et-de-sortie* :: *Nom-de-variable*

*Identificateur-de-référence-de-sortie*

*Graphe-de-procédure* :: *Noeud-de-départ-de-procédure*

*Noeud-d'état-set*

*Noeud-de-départ-de-procédure* :: *Transition*

*Grammaire textuelle concrète*

<définition de procédure> ::=

```
PROCEDURE { identificateur de procédure
| nom de procédure
} <fin>
```

[<paramètres formels de procédure> <fin>]

{ définition de données>

| <définition de variable>

| <référence textuelle de procédure>

| <définition de procédure>

| <définition de sélection>

| <définition de macro }

<corps de procédure>

```
ENDPROCEDURE [<nom de procédure
| identificateur de procédure >] <fin>
```

<paramètres formels de procédure> ::=

FPAR <paramètre formel de variable>

{ <paramètre formel de variable }

<paramètre formel de variable>::=

[IN/OUT

| IN]

<nom de variable

```
{ <nom de variable
} <sorte>
```

<corps de procédure> ::=

<corps de processus>

<définition de variable> est définie au § 2.6.1.1, <référence textuelle de procédure> au § 2.4.4, <définition de macro> au § 4.2, <définition de sélection> au § 4.3.3, <définition de données> au § 5.5.1, <sorte> au § 5.2.2.

Dans une <définition de procédure>, la <définition de variable> ne peut contenir les expressions REVAELED, EXPORTED, REVAELED EXPORTED, EXPORTED REVAELED <nom de variable > (voir le § 2.6.1). Un exemple

de <définition de proc'edure> est donné à la figure 2.9.11.

*Grammaire graphique concrète*

<diagramme de proc'edure> ::=

<symbole de cadre> **contains** { en-t | te de proc'edure>

{ zone de texte de proc'edure>

| <zone de proc'edure>

| <diagramme de macro }

<zone de graphe de proc'edure } **set** }

<zone de proc'edure> ::=

<r'ef'erece graphique de proc'edure>

| <diagramme de proc'edure>

<zone de texte de proc'edure> ::=

<symbole de texte> **contains**

{ d'efinition de variable>

| <d'efinition de donn'ees>

| <d'efinition de s'election>

| <d'efinition de macro }

<r  f  rence graphique de proc  dure> ::=

<symbole de proc  dure> **contains** <nom de proc  dure >

<symbole de proc  dure> ::=

### Figura 9, p.

<en-t | te de proc  dure> ::=

```
PROCEDURE { nom de proc  dure  
| identificateur de proc  dure  
}
```

[<param  tres formels de proc  dure>]

<zone de graphe de proc  dure> ::=

<zone de d  part de proc  dure>

{ zone de d  tat | zone de connecteur d  entr  e } }

<zone de d  part de proc  dure> ::=

<symbole de d  part de proc  dure> **is followed by** <zone de transition>

<symbole de d  part de proc  dure> ::=

.rs

### Figura 10, p.

<d  finition de variable> est d  finie au    2.6.1.1, <zone de transition> au    2.6.7.1, <zone d  tat> au    2.6.3, <zone de connecteur d  entr  e> au    2.6.6, <d  finition de macro> et <diagramme de macro> au    4.2, <d  finition de s  lection> au    4.3.3, <d  finition de donn  es> au    5.5.1.

Un exemple de <diagramme de proc  dure> est montr      la figure 2.9.12 au    2.9.

#### *S  mantique*

Une proc  dure est un moyen de donner un nom    un assemblage d  objets et de le repr  senter par une r  f  rence unique. Les r  gles relatives aux proc  dures imposent une discipline sur la mani  re dont l  assemblage d  objets est choisi et elles limitent la port  e du nom des variables d  finies dans la proc  dure.

Une variable de proc  dure est une variable locale    la proc  dure qui ne peut appara  tre,   tre visible,   tre export  e ou   tre import  e. Elle est cr   e lors de l  interpr  tation du noeud de d  part de proc  dure et cesse d  exister lors de l  interpr  tation du noeud de retour du graphe de proc  dure.

Lorsqu  une d  finition de proc  dure est interpr  t  e, son graphe de proc  dure est   galement interpr  t  .

Une d  finition de proc  dure est interpr  t  e seulement lorsqu  une instance de processus l  appelle, et l  interpr  tation est faite par l  instance de processus.

L'interprétation d'une définition de procédure provoque la création d'une instance de procédure et l'interprétation commence de la manière suivante:

- a) une variable locale est créée pour tout *paramètre-d'entrée* , ayant le *nom* et la *sorte* du *paramètre-d'entrée* paramètre effectif correspondant, qui peut ne pas être défini;
  - b) lorsqu'un paramètre effectif est vide, la valeur attribuée au paramètre formel correspondant n'est pas définie;
  - c) un paramètre formel n'ayant pas d'attribut explicite, a un attribut IN implicite;
- .bp
- d) une variable locale est créée pour chaque *définition-de-variable* dans la *définition-de-procédure* ayant le *nom* et la *sorte* de la *définition-de-variable* ;
  - e) chaque *paramètre-d'entrée-sortie* décrit un nom de synonyme pour la variable qui est donnée dans l'expression des paramètres effectifs. Ce nom de synonyme est utilisé tout au long de l'interprétation du *graphe-de-procédure* lorsqu'on se réfère à la valeur de la variable ou lorsque l'on affecte une nouvelle valeur à la variable;
  - f) la *transition* contenu dans le *noeud-de-départ-de-procédure* est interprétée.

## 2.5 Communication

### 2.5.1 Canal

*Grammaire abstraite*

*Définition-de-canal* :: *Nom-de-canal*

*Chemin-de-canal*

[*Chemin-de-canal* ]

*Chemin-de-canal* :: *Bloc-d'origine*

*Bloc-destinataire*

*Identificateur-de-signal-set*

*Bloc-d'origine* = *Identificateur-de-bloc* |

ENVIRONMENT

*Bloc-destinataire* = *Identificateur-de-bloc* |

ENVIRONMENT

*Identificateur-de-bloc* = *Identificateur*

*Identificateur-de-signal* = *Identificateur*

*Nom-de-canal* = *Nom*

L'*identificateur-de-signal-set* doit contenir la liste de tous les signaux qui peuvent être acheminés sur le ou les chemins-de-canal définis par le canal.

Une extrémité du canal doit au moins être un bloc.

Si les deux points extrêmes sont des blocs, ceux-ci doivent être différents.

La ou les extrémité(s) des blocs doivent être définies dans la même unité de portée que celles où est défini le canal.

#### *Grammaire textuelle concrète*

<définition de canal> ::=

CHANNEL <nom de canal >

<trajet de canal>

[<trajet de canal>]

[<définition de sous-structure de canal>

| <référence textuelle de sous-structure de canal>]

ENDCHANNEL [<nom de canal >] <fin>

<trajet de canal> ::=

{ FROM <identificateur de bloc > TO <identificateur de bloc >

| FROM <identificateur de bloc > TO ENV

| FROM ENV TO <identificateur de bloc

}

WITH <liste de signaux> <fin>

<liste de signaux> est définie au § 2.5.5, <définition de sous-structure de canal> et <référence textuelle de sous-structure de canal> au § 3.2.3.

Lorsque deux <trajet de canal> sont définis, le sens de l'un doit être opposé par rapport au sens de l'autre.

.bp

#### *Grammaire graphique concrète*

<zone de définition de canal>::=

<symbole de canal>

**is associated with** { nom de canal >

{ { identificateur de canal

| identificateur de bloc

}

<zone de liste de signaux>[<zone de liste de signaux> } **set** }

**is connected to** { zone de bloc { zone de bloc | symbole de cadre }

[<zone d'association de sous-structure de canal> } **set**

L'<identificateur de canal > identifie un canal externe relié au <diagramme de sous-structure de bloc> d'limité par le <symbole de cadre>. L'<identificateur de bloc > identifie un bloc externe comme étant une extrémité de canal pour le <diagramme de sous-structure de canal> d'limité par le <symbole de cadre>.

<symbole de canal>::=



<symbole de canal 1>

| <symbole de canal 2>

| <symbole de canal 3>

<symbole de canal 1> ::=

### **MONTAGE**

<symbole de canal 2> ::=

### **MONTAGE**

<symbole de canal 3> ::=

### **MONTAGE**

<zone de liste de signaux> est définie au § 2.5.5, <zone de bloc> et <symbole de cadre> au § 2.4.1 et <zone d'association de sous-structure de canal> au § 3.2.3.

Pour chaque flèche placée sur le <symbole de canal>, il doit y avoir une <zone de liste de signaux>. Une <zone de liste de signaux> doit

| tre suffisamment proche de la flèche à laquelle elle est associée pour éviter toute ambiguïté.

### *Sémantique*

Un canal est un moyen de transport pour les signaux. Un canal peut | tre considéré comme étant un ou deux trajets de canal unidirectionnels indépendants entre deux blocs ou entre un bloc et son environnement.

Les signaux acheminés par les canaux sont véhiculés jusqu'au point extr | me de destination.

Au point extr | me de destination d'un canal, les signaux se présentent dans le m | me ordre que celui des signaux au point d'origine. Si deux ou plusieurs signaux arrivent simultanément sur le canal, ils sont ordonnés arbitrairement.

Un canal peut retarder l'acheminement des signaux sur le canal. Cela signifie qu'une file d'attente du type premier-entré-premier-sorti (FIFO) se trouve associée à chaque direction dans un canal. Quand un signal se présente sur le canal, il est placé dans la file d'attente. Après un intervalle de temps non déterminé et non constant, la première instance de signal dans la file d'attente est libérée et appliquée à l'un des canaux ou l'un des trajets de signaux qui se trouvent connectés au canal.

Plusieurs canaux peuvent exister entre deux points d'extrémités. Le m | me type de signal peut | tre acheminé sur des canaux différents.

### *2.5.2 Acheminement du signal*

#### *Grammaire abstraite*

*Définition-d'acheminement-de-signal* :: *Nom-d'acheminement-de-signal*

*Trajet-d'acheminement-du-signal*

[*Trajet-d'acheminement-du-signal* ]

*Trajet-d'acheminement-du-signal* :: *Processus-d'origine*

*Processus-destinataire*

*Identificateur-de-signal- set*

*Processus-d'origine* = *Identificateur-de-processus* |

ENVIRONMENT

*Processus-destinataire* = *Identificateur-de-processus* |

ENVIRONMENT

*Nom-d'acheminement-de-signal* = *Nom*

Un des points extr | mes au moins du *trajet-d'acheminement-du-signal* doit | tre un processus.

Si les deux points extr | mes sont des processus, les *identificateur-de-processus* doivent | tre diff´erents.

Le ou les point(s) extr | me(s) du processus doivent | tre d´efinis dans la m | me unit´e de port´ee que celle de l'acheminement du signal.

### *Grammaire textuelle concrète*

<définition d'acheminement de signal>::=

SIGNALROUTE <nom d'acheminement de signal>

<trajet d'acheminement du signal>

[<trajet d'acheminement du signal>]

<trajet d'acheminement du signal>::=

{ FROM <identificateur de processus> TO <identificateur de processus>

| FROM <identificateur de processus> TO ENV

| FROM ENV TO <identificateur de processus> }

WITH <liste de signaux> <fin>

La <liste de signaux> est définie au § 2.5.5.

Lorsque deux <trajet d'acheminement du signal> sont définis, l'un doit être en sens opposé à l'autre.

### *Grammaire graphique concrète*

<zone de définition d'acheminement de signal> ::=

<symbole d'acheminement de signal>

**is associated with** { nom d'acheminement de signal >

{ <identificateur de canal >] <zone de liste de signaux> [<zone de liste de signaux> } **set** }

**is connected to**

{ zone de processus> { zone de processus> | <symbole de cadre > } **set**

<symbole d'acheminement de signal> ::=

<symbole d'acheminement de signal 1> | <symbole d'acheminement de signal 2>

<symbole d'acheminement de signal 1> ::=

**MONTAGE**

<symbole d'acheminement de signal 2> ::=

**MONTAGE**

Un symbole d'acheminement de signal comprend une tête de flèche à une extrémité (une direction) ou une tête de flèche à chaque extrémité (deux directions) pour montrer la direction du flot des signaux.

Pour chaque flèche située sur le <symbole d'acheminement de signal>, il doit exister une <zone de liste de signaux>. Une <zone de liste de signaux> doit être suffisamment proche de la flèche à laquelle elle est associée afin d'éviter toute ambiguïté.

Lorsque le <symbole d'acheminement de signal> est relié au <symbole de cadre>, l'<identificateur de canal > identifie un canal auquel l'acheminement de signal est relié.

### *Sémantique*

Un acheminement de signal est un moyen de transport pour les signaux. Un acheminement de signal peut être considéré comme étant un ou deux trajets d'acheminement de signal unidirectionnels et indépendants entre deux processus ou entre un processus et son environnement.

Les signaux acheminés par des acheminements de signaux sont délivrés au point extrême de destination.

Un acheminement de signaux n'apporte aucun retard dans le transport des signaux.

Aucun acheminement de signal ne peut relier des instances de processus du même type. Si tel est le cas, l'interprétation du noeud-de-sortie a pour conséquence implicite que le signal est appliqué directement à l'accès d'entrée du processus de destination.

Plusieurs trajets de signaux peuvent exister entre deux points extrêmes. Le même type de signal peut être acheminé sur différents acheminements de signaux.

### Modèle

Un <ensemble de signaux d'entrée valides> contient les signaux que le processus est autorisé à recevoir. Un <ensemble de signaux d'entrée valides> ne doit toutefois pas contenir de signaux de temporisateur. Si une <définition de bloc> contient des <définition d'acheminement de signal>, l'ensemble de signaux d'entrée valides, s'il y en a un, n'a pas besoin de contenir des signaux dans des acheminements de signaux conduisant au processus.

Si une <définition de bloc> ne contient aucune <définition d'acheminement de signal>, toutes les <définition de processus> dans cette <définition de bloc> doivent contenir un <ensemble de signaux d'entrée valides>. Dans ce cas, les <définition d'acheminement de signal> et les <connexion de canal à acheminement> sont tirées de l'ensemble de signaux d'entrée valides, des <sorties> et des canaux qui se terminent à la frontière des blocs. Les signaux correspondant à une direction donnée entre deux processus dans l'acheminement de signal implicite constituent l'intersection des signaux spécifiés dans l'ensemble de signaux d'entrée valides du processus de destination et les signaux mentionnés dans une sortie du processus d'origine. Si l'un des points extrêmes est l'environnement, l'ensemble d'entrée/ensemble de sortie est constitué par les signaux acheminés par le canal dans la direction donnée.

.bp

### 2.5.3 Connexion

#### Grammaire abstraite

*Connexion-de-canal-à-acheminement* :: *Identificateur-de-canal*

*Identificateur-d'acheminement-de-signal* set

*Identificateur-d'acheminement-de-signal* = *Identificateur*

D'autres constructions sont données au § 3.

Chaque *identificateur-de-canal* relié au bloc englobant doit

être mentionné dans une seule *connexion-de-canal-à-acheminement*. L'*identificateur-de-canal* dans une *connexion-de-canal-à-acheminement* doit dénoter un canal connecté au bloc entre crochets.

Chaque *identificateur-d'acheminement-de-signal* dans une *connexion-de-canal-à-acheminement* doit être défini dans le même bloc où la *connexion-de-canal-à-acheminement* se trouve définie et doit avoir la frontière de ce bloc située à l'un de ses points extrêmes. Chaque *identificateur-d'acheminement-de-signal* défini dans le bloc qui l'entoure et qui a son environnement comme étant l'un de ses points extrêmes, doit être mentionné dans une seule et unique *connexion-de-canal-à-acheminement*.

Pour une direction donnée, l'union des ensembles d'*identificateur de signal* dans les acheminements de signaux dans une *connexion-de-canal-à-acheminement* doit être égale aux signaux acheminés par l'*identificateur-de-canal* dans la même *connexion-de-canal-à-acheminement* et correspondant à la même direction.

### *Grammaire textuelle concrète*

<connexion de canal à acheminement> ::=

CONNECT <identificateur de canal >

AND <identificateur d'acheminement de signal > { <identificateur d'acheminement de signal  
> } <fin>

On ne peut mentionner deux fois un <identificateur d'acheminement de signal dans une <connexion de canal à acheminement>.

### *Grammaire graphique concrète*

Sur le plan graphique, l'élément connexion est représenté par l'<identificateur de canal > associé au trajet de signal et contenu dans la <zone de définition d'acheminement de signal> (voir le § 2.5.2 *grammaire graphique concrète*).

## 2.5.4 Signal

### *Grammaire abstraite*

*Définition-de-signal* :: *Nom-de-signal*

*Identificateur-de-référence-de-sortie* |

[*Affinage-de-signal* ]

*Nom-de-signal* :: *Nom*

L'*identificateur-de-référence-de-sortie* est défini au § 5.2.2.

### *Grammaire textuelle concrète*

<définition de signal>::=

SIGNAL { nom de signal > [<liste de sortie>] [<affinage de signal> ]

{ <nom de signal > [<liste de sortie>] [<affinage de signal> ] <fin>

<liste de sortie>::=

(<sortie> { <sortie > } )

L'<affinage de signal> est défini au § 3.3, la <sortie> au § 5.2.2.

### *Sémantique*

Une instance de signal est un flot d'informations entre des processus; c'est une instanciation d'un type de signal défini par une définition de signal. Une instance de signal peut être envoyée par l'environnement ou par un processus, elle se dirige toujours vers un processus ou vers l'environnement.

On associe à chaque instance de signal deux valeurs de PId (voir le § 5.6.10) indiquant les processus de départ et de destination, l'<identificateur de signal > spécifié dans la sortie correspondante et les autres valeurs, dont les sorties sont définies dans la définition d'un signal.

.bp

### 2.5.5 Définition de liste de signaux

Un <identificateur de liste de signaux > peut

| tre utilisé dans une <définition de canal>, une <définition d'acheminement de signal>, une <définition de liste de signaux>, un <ensemble de signaux d'entrée valides> et <liste de mise en réserve> comme moyen abrégé pour énumérer les identificateurs de signaux et les signaux de temporisation.

#### Grammaire textuelle concrète

<définition de liste de signaux>::=

SIGNALLIST <nom de liste de signaux >=<liste de signaux><fin>

<liste de signaux>::=

<objet signal> { <objet signal >

<objet signal>::=

<identificateur de signal > | <identificateur de signal prioritaire > (<identificateur de liste de signaux >) |  
<identificateur de temporisation >

La <liste de signaux> qui est établie en remplaçant tous les <identificateur de liste de signaux > dans la liste par les <identificateur de signal > qu'ils écrivent, correspond à un *identificateur-de-signal-set* dans la grammaire abstraite. Dans chaque <liste de signaux> ainsi établie, chaque <identificateur de signal > doit  
| tre distinct.

#### Grammaire graphique concrète

<zone de liste de signaux>::=

<symbole de liste de signaux> **contains** <liste de signaux>

<symbole de liste de signaux>::=

.rs

Figura 1, p. 50 (E)

