5i’

1 Introduction au LED

1.1 *Introduction*

Le but poursuivi, en recommandant l’utilisation du LDS (langage de description et de sp´ecificationfonctionnelles), est d’avoir un langage permettant de sp´ecifier et de d´ecrire sans ambigu"it´e le comportement des syst`emes de t´el´ecommunication. Les sp´ecifications et les descriptions faites à l’aide du LDS doivent | tre formelles dans ce sens qu’il doit | tre possible de les analyser et de les interpr´eter sans ambigu"it´e.

Les termes sp´ecification et description sont utilis´es dans le sens ci-apr`es:

a) la sp´ecification d’un syst`eme est la description du comportement souhait´e de celui-ci, et

b) la description d’un syst`eme est la description du comportement r´eel de celui-ci.

Remarque — Etant donn´e qu’il n’est pas fait de distinction entre l’utilisation du LDS pour la sp´ecification et son utilisation pour la description, le terme sp´ecificationdans le texte qui suit est utilis´e pour d´esigner à la fois le comportement souhait´e et le comportement r´eel.

Une sp´ecification du syst`eme, au sens large, est la sp´ecificationà la fois du comportement et d’un ensemble de param`etres g´en´eraux du syst`eme. Toutefois, le LDS ne vise qu’`a d´ecrire les aspects relatifs au comportement d’un syst`eme; les param`etres g´en´eraux concernant des propri´et´es telles que la capacit´e et le poids doivent
| tre d´ecrits à l’aide de techniques diff´erentes.

1.1.1 *Objectifs*

Les objectifs g´en´eraux qui ont ´et´e pris en compte lors de la d´efinition du LDS sont de fournir un langage:

a) facile à apprendre, à utiliser et à interpr´eter;

b) permettant l’´elaboration de sp´ecifications d´epourvues d’ambigu"it´e pour faciliter la soumission des offres et le partage des commandes;

c) extensible pour permettre un d´eveloppement ult´erieur;

d) permettant l’application de plusieurs m´ethodologies de sp´ecification et de conception de syst`eme, sans supposer a priori l’une quelconque de ces m´ethodologies.

1.1.2 *Domaine d’application*

Le domaine d’application principal du LDS est la description du comportement des syst`emes en temps r´eel dans certains de leurs aspects. Ces applications comprennent:

a) le traitement des appels (par exemple: ´ecoulement, signalisation t´el´ephonique, comptage aux fins de taxation, etc.) dans les syst`emes de commutation;

b) la maintenance et la rel`eve des d´erangements (par exemple: alarme, rel`eve automatique des d´erangements, essais p´eriodiques, etc.) dans les syst`emes g´en´eraux de t´el´ecommunication;

c) la commande du syst`eme (par exemple: protection contre les surcharges, proc´edures de modification et d’extension, etc.);

d) les fonctions d’exploitation et de maintenance, la gestion des r´eseaux;

e) les protocoles de communication de donn´ees.

Il va de soi que le LDS peut aussi servir à la sp´ecification fonctionnelle du comportement d'un objet lorsque celui-ci peut | tre sp´ecifi´e au moyen d'un mod`ele discret, c'est-`a-dire un objet communiquant avec son environnement au moyen de messages discrets.

Le LDS est un langage particuli`erement riche qui peut | te utilis´e à la fois pour des sp´ecifications de haut niveau informelles (et/ou formellement incompl`etes), des sp´ecifications partiellement formelles et des sp´ecifications d´etaill´ees. L'utilisateur doit choisir les parties appropri´ees du LDS en fonction du niveau de communication souhait´e et de l'environnement dans lequel le langage sera utilis´e. Selon l'environnement dans lequel une sp´ecification est utilis´ee, certains ´el´ements qui rel`event du simple bon sens pour l'´emetteur et le destinataire de la sp´ecification, ne seront pas explicit´es.

Ainsi, le LDS peut | tre utilis´e pour:

a) ´etablir les sp´ecifications d'une installation,

b) ´etablir les sp´ecifications d'un syst`eme,

c) ´etablir des Recommandations du CCITT,

d) ´etablir des sp´ecifications de conception d'un syst`eme,

e) ´etablir des sp´ecifications d´etaill´ees,

f) concevoir un syst`eme (`a la fois globalement et dans le d´etail),

g) effectuer des essais d'un syst`eme,

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

.bp

1.1.3 *Sp´ecification d'un syst`eme*

Le LDS d´ecrit le comportement d'un syst`eme sous la forme de stimulus/r´eaction, ´etant admis que les stimuli aussi bien que les r´eactions sont des entit´es discr`etes et contiennent de l'information. En particulier, la sp´ecification d'un syst`eme est vue comme ´etant la s´equence de r´eactions associ´ee à une s´equence de stimuli.

Le mod`ele de sp´ecification d'un syst`eme est fond´e sur la notion de machine à ´etats finis ´etendue.

Le LDS fait appel à des concepts structurels qui facilitent la sp´ecification des grands syst`emes et des syst`emes complexes. Il est ainsi possible de subdiviser la sp´ecification d'un syst`eme en unit´es faciles à g´erer qui peuvent | tre trait´ees et comprises de mani`ere ind´ependante. La subdivision peut s'op´erer en plusieurs ´etapes qui permettent d'obtenir une structure hi´erarchique d'unit´es d´efinissant le syst`eme à diff´erents niveaux.

1.2 *Grammaires du LDS*

Le LDS offre le choix entre deux formes syntaxiques diff´erentes pour repr´esenter un syst`eme: une repr´esentation graphique (LDS/GR) et une repr´esentation textuelle (LDS/PR). Comme ces formes sont toutes les deux des repr´esentations concr`etes de la m | me s´emantique du LDS, elles sont ´equivalentes du point de vue s´emantique. En particulier, elles sont toutes les deux ´equivalentes 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 appel´e grammaire textuelle commune.

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

.rs

Figure 1.1, p.

A chaque grammaire concr`ete, est associ´ee sa propre syntaxe et les rapports qu'elle a avec la grammaire abstraite (c'est-`a-dire la mani`ere de la transformer en syntaxe abstraite). Avec cette m´ethode, la d´efinition de la s´emantique du LDS est unique; chacune des grammaires concr`etes h´eritera de la s´emantique par l'interm´ediaire de ses relations avec la grammaire abstraite. Cette m´ethode permet ´egalement d'assurer l'´equivalence entre le LDS/PR et le LDS/GR.

On dispose ´egalement d'une d´efinition formelle du LDS qui d´efinit la mani`ere de transformer la sp´ecification d'un syst`eme dans la syntaxe abstraite et comment interpr´eter une sp´ecification, donn´ee en termes de syntaxe abstraite.

1.3 *D´efinitions fondamentales*

La pr´esente Recommandation fait appel à des conventions et à des concepts g´en´eraux dont les d´efinitions sont donn´ees ci-apr`es:

1.3.1 *Type, d´efinition et instance*

Dans la pr´esente Recommandation, les concepts de type, d'instance de type et les relations qui existent entre elles, jouent un r | le fondamental. Le sch´ema et la terminologie utilis´es sont explicit´es ci-apr`es et illustr´es par la figure 1.2.

.bp

Figura 1.2, p.

Un type se d´efinit par des d´efinitions, lesquelles d´efinissent toutes les propri´et´es associ´ees à ce type. Un type peut | tre d´ecompos´e en un nombre quelconque d'instances. Toute instance d'un type particulier poss`ede toutes les propri´et´es d´efinies pour ce type.

Ce sch´ema s'applique à plusieurs concepts du LDS, c'est-`a-dire qu'il existe des d´efinitions de syst`eme et des instances de syst`eme, des d´efinitions de processus et des instances de processus.

Le type de donn´ees constitue une cat´egorie sp´eciale de type (voir les § 2.3 et 5).

Remarque — Pour ´eviter d'alourdir le texte, on peut s'abstenir d'utiliser le terme instance. Ainsi, pour exprimer qu'une instance du syst`eme est interpr´et´ee, on ´ecrira <<un syst`eme est interpr´et´e | | | | >.

1.3.2 *Environnement*

Les syst`emes qui sont sp´ecifi´es en LDS r´eagissent d'apr`es les stimuli qu'ils re,coivent du monde ext´erieur. Ce monde ext´erieur est appel´e environnement du syst`eme en cours de sp´ecification.

On suppose qu'il y a une ou plusieurs instances de processus dans l'environnement et, par cons´equent, les signaux circulant de l'environnement en direction du syst`eme ont des identit´es associ´ees à ces instances de processus. Ces processus ont des valeurs PId diff´erentes des autres valeurs PId du syst`eme (voir § 5.6.10).

Bien que le comportement du syst`eme soit non d´eterministe, il doit ob´eir aux contraintes impos´ees par la sp´ecification du syst`eme.

1.3.3 *Erreurs*

Une sp´ecification de syst`eme est une sp´ecification de syst`eme correcte en LDS seulement si elle r´epond aux r`egles syntaxiques et aux conditions statiques du LDS.

Lorsqu'une sp´ecification LDS correcte est interpr´et´ee et qu'une condition dynamique se trouve viol´ee, une erreur appara | t. Une interpr´etation d'une sp´ecification de syst`eme qui conduit à une erreur indique que le comportement du syst`eme ne peut pas | tre d´etermin´e à partir de la sp´ecification.

1.4 *Pr´esentation*

1.4.1 *Structuration du texte*

Les § 2, 3, 4 et 5 de la pr´esente Recommandation sont structur´es par th`emes, ils comportent des intitul´es pr´ec´ed´es ´eventuellement par une introduction: ces intitul´es sont les suivants:

a) *Grammaire abstraite* — d´ecrite par une syntaxe abstraite et des conditions statiques pour d´efinitions bien form´ees.

b) *Grammaire textuelle concr`ete* — qui concerne à la fois la grammaire textuelle courante utilis´ee pour le LDS/PR et le LDS/GR et la grammaire uniquement utilis´ee pour le LDS/PR. Cette grammaire est d´ecrite au moyen d'une syntaxe textuelle, de conditions statiques et de r`egles de d´efinitions bien form´ees, concernant la syntaxe textuelle, et de la relation de la syntaxe textuelle avec la syntaxe abstraite.

c) *Grammaire graphique concr`ete* — d´ecrite par la syntaxe graphique, les conditions statiques et les r`egles de d´efinitions bien form´ees concernant la syntaxe graphique, la relation de cette syntaxe avec la syntaxe abstraite et quelques r`egles de dessin qui viennent en suppl´ement (`a celles indiqu´ees au § 2.2.4).

.bp

d) *S´emantique* — donnant une signification à un type, conf`ere les propri´et´es de ce type, la fa,con avec laquelle une instance de ce type est interpr´et´ee et toutes conditions dynamiques qui doivent | tre remplies par l'instance de ce type pour avoir un comportement correct au sens du LDS.

e) *Mod`ele* — donne la correspondance avec les abr´eviations exprim´ees en termes de constructions en syntaxe concr`ete stricte pr´ec´edemment d´efinies.

f) *Exemples* .

1.4.2 *Intitul´es*

L'introduction qui pr´ec`ede ´eventuellement les intitul´es, est uniquement destin´ee à faciliter la compr´ehension du texte et à ce titre doit | tre consid´er´ee comme ´etant une partie officieuse de la Recommandation.

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

La suite de la pr´esente section d´ecrit les autres formalismes particuliers utilis´es dans chaque intitul´e et les titres utilis´es. Elle peut ´egalement | tre consid´er´ee comme un exemple de pr´esentation typographique du premier niveau des intitul´es d´efinis ci-dessus, ce texte appartenant à l'introduction.

Grammaire abstraite

La notation en syntaxe abstraite est d´efinie au § 1.5.1.

L'absence de l'intitul´e *grammaire abstraite* indique qu'il n'existe pas d'autres syntaxes abstraites pour le sujet trait´e et que la syntaxe concr`ete correspond à la syntaxe abstraite d´efinie par une autre section de texte num´erot´ee.

On peut se r´ef´erer à une des r`egles dans la syntaxe abstraite à partir de tout intitul´e en maintenant le nom de la r`egle en italique.

Les r`egles dans la notation formelle peuvent | tre suivies par des paragraphes qui d´efinissent les conditions qui doivent | tre satisfaites par une d´efinition LDS bien form´ee et qui peuvent | tre v´erifi´ees sans interpr´etation d'une instance. Les conditions statiques à ce niveau se r´ef`erent uniquement à la syntaxe abstraite. Les conditions statiques qui ne concernent seulement que la syntaxe concr`ete sont d´efinies post´erieurement à la syntaxe concr`ete. La syntaxe abstraite, associ´ee aux conditions statiques applicables à la syntaxe abstraite, d´efinit la grammaire abstraite du langage.

Grammaire textuelle concr`ete

La syntaxe textuelle concr`ete est sp´ecifi´ee au moyen de la Backus-Naur Form (BNF) ´etendue de la description de la syntaxe d´efinie au § 2.1 de la Recommandation Z.200 (voir ´egalement le § 1.5.2).

La syntaxe textuelle est suivie par des paragraphes d´efinissant les conditions statiques qui doivent | tre satisfaites dans un texte bien form´e et qui peuvent | tre v´erifi´ees sans interpr´etation d'une instance. Cela s'applique ´egalement aux conditions statiques, si elles existent, pour la grammaire abstraite.

Dans de nombreux cas, il y a une simple relation entre la syntaxe concr`ete et la syntaxe abstraite ´etant donn´e qu'une r`egle de la syntaxe concr`ete est simplement repr´esent´ee par une seule r`egle dans la syntaxe abstraite. Lorsque le m | me nom est utilis´e dans la syntaxe abstraite et dans la syntaxe concr`ete, afin d'indiquer qu'il repr´esente le m | me concept, le texte pr´ecisant que <<<x> dans la syntaxe concr`ete repr´esente *X* dans la syntaxe abstraite>> est implicite dans la description du langage et est souvent omis. Dans ce contexte, le cas est ignor´e mais les sous-cat´egories s´emantiques soulign´ees sont significatives.

La syntaxe textuelle concr`ete qui ne constitue pas une forme abr´eg´ee (syntaxe d´eriv´ee mod´elis´ee par d'autres constructions LDS) est une syntaxe textuelle concr`ete stricte. La relation entre la syntaxe textuelle concr`ete 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

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

L’expansion dans la syntaxe concr`ete provenant de constructions telles les d´efinitions diff´er´ees (§ 2.4.1), les macros (§ 4.2) et les mises en correspondance de litt´eraux (§ 5.4.1.15) etc., doit | tre examin´ee avant la mise en correspondance entre la syntaxe concr`ete et la syntaxe abstraite.

S´emantique

Des propri´et´es sont utilis´ees dans les r`egles de bonne formation qui font intervenir soit le type soit d’autres types qui se r´ef`erent à ce type.

Un exemple de propri´et´e est l’ensemble des identificateurs de signaux d’entr´ee valides d’un processus. Cette propri´et´e est utilis´ee dans la condition statique <<pour chaque noeud-d’´etat, tous les *identificateurs de signaux d’entr´ee* (dans l’ensemble des signaux d’entr´ee valides) apparaissent soit dans un *ensemble de signaux mis en r´eserve* ou dans un *noeud d’entr´ee* >>.

Toutes les instances ont une propri´et´e d’identit´e mais à moins que celle-ci soit form´ee d’une fa,con quelque peu inhabituelle, cette propri´et´e d’identit´e est d´etermin´ee comme ´etant d´efinie par la section g´en´erale traitant des identit´es au § 2. Par cons´equent, cela n’est pas habituellement mentionn´e comme ´etant une propri´et´e d’identit´e. Il n’est ´egalement pas n´ecessaire d’indiquer les sous-composantes d’une d´efinition contenues par la d´efinition ´etant donn´e que l’appartenance de telles sous-composantes est ´evidente à partir de la syntaxe abstraite. Par exemple, il est ´evident qu’une d´efinition de bloc <<a>> englob´e des d´efinitions de processus et ´eventuellement une d´efinition de sous-structure de bloc.

Les propri´et´es sont statiques, si elles peuvent | tre d´etermin´ees sans l’interpr´etation d’une sp´ecification de syst`eme en LDS et sont dynamiques si l’interpr´etation de ce syst`eme est n´ecessaire pour d´eterminer la propri´et´e.

L’interpr´etation est d´ecrite de mani`ere op´erationnelle. Lorsqu’il y a une liste dans la syntaxe abstraite, cette liste est interpr´et´ee dans l’ordre donn´e. C’est-`a-dire la Recommandation d´ecrit comment les instances sont cr´e´ees à partir de la d´efinition du syst`eme et comment celles-ci sont interpr´et´ees dans une <<machine abstraite LDS>>.

Les conditions dynamiques sont des conditions qui doivent | tre satisfaites durant l’interpr´etation et qui ne peuvent | tre v´erifi´ees sans interpr´etation. Les conditions dynamiques peuvent conduire à des erreurs (voir le § 1.3.3).

Mod`ele

Certaines constructions sont consid´er´ees comme ´etant une <<syntaxe concr`ete d´eriv´ee>> (ou une abr´eviation) pour d’autres constructions ´equivalentes en syntaxe concr`ete. Par exemple, l’omission d’une entr´ee pour un signal est une syntaxe concr`ete d´eriv´ee pour une entr´ee pour ce signal suivi par une transition nulle avec retour vers le m | me ´etat.

Dans certains cas, une telle <<syntaxe concr`ete d´eriv´ee>>, si elle est ´etendue, donnera lieu à une repr´esentation immens´ement grande (´eventuellement infinie). N´eanmoins, la s´emantique d’une telle sp´ecification peut | tre d´etermin´ee.

Exemples

L’intitul´e *exemples* contient des exemples.

1.5 *M´etalangages*

Pour la d´efinition des propri´et´es et des syntaxes du LDS, diff´erents m´etalangages ont ´et´e utilis´es en fonction des besoins particuliers.

Dans ce qui suit, on trouvera une introduction aux m´etalangages; des r´ef´erences renvoyant à des livres ou à des publications particuli`eres de l’UIT seront donn´ees au besoin.

1.5.1 *Le M´eta IV*

Le sous-ensemble suivant du *M´eta IV* est utilis´e pour d´ecrire la syntaxe abstraite du LDS.

Une d´efinition dans la syntaxe abstraite peut | tre consid´er´ee comme ´etant un objet composite nomm´e (une arborescence) d´efinissant un ensemble de sous-composantes.

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

D´efinition-de-variable :: *Nom-de-variable Identificateur-de-r´ef´erence-de-sorte*

qui d´efinit le domaine de l'objet composite (arborescence) appel´e *d´efinition-de-variable* . Cet objet comporte deux sous-composantes qui à leur tour peuvent | tre des arborescences.

La d´efinition en M´eta IV

Identificateur-de-r´ef´erence-de-sorte = *Identificateur*

indique qu'un *identificateur-de-r´ef´erence-de-sorte* est un *identificateur* et ne peut par cons´equent | tre syntaxiquement distingu´e des autres identificateurs.

.bp

Certains objets peuvent ´egalement | tre constitu´es par certains domaines ´el´ementaires (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* d´esigne un domaine composite contenant deux des valeurs enti`eres (*entier*) indiquant le nombre initial et le nombre maximal d'instances.

b) Mots cl´es

Les mots cl´es sont repr´esent´es par une s´equence en caract`eres 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 d´esign´e par le mot cl´e **ENVIRONNEMENT** .

c) Marques

Le terme *Token* d´esigne le domaine des marques. Ce domaine peut | tre consid´er´e comme ´etant compos´e d'un ensemble potentiellement infini d'objets atomiques distincts pour lesquels aucune repr´esentation n'est requise.

Exemple:

Nom :: *Token*

Un nom est un objet atomique tel que tout nom peut | tre distingu´e de tout autre nom.

d) Objets non sp´ecifi´es

Un objet non sp´ecifi´e d´esigne des domaines qui peuvent avoir une certaine repr´esentation, mais pour lesquels la repr´esentation n'int´eresse pas la pr´esente Recommandation.

Exemple:

Texte-informel :: ...

Texte-informel contient un objet qui n'est pas interpr´et´e.

Les op´erateurs ci-apr`es (constructeurs) dans la forme BNF (voir le § 1.5.2) sont ´egalement utilis´es dans la syntaxe abstraite: <<*>> pour d´esigner une liste pouvant | tre vide; <<+>> pour d´esigner une liste non vide; < | > pour repr´esenter une alternative, et <<[<< >>]>> pour indiquer une option.

Les parenth`eses sont utilis´ees pour regrouper les domaines qui pr´esentent un rapport logique.

Enfin, la syntaxe abstraite utilise un autre op´erateur de suffixe <<**-set** >> produisant un ensemble (collection non ordonn´ee d'objets distincts).

Exemple:

graphe-de-processus :: *noeud-de-d´epart-de-processus noeud-d'´etat* **-set**

Un *graphe-de-processus* est constitu´e par un *noeud-de-d´epart-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-`a-dire le signe inf´erieur à ou le signe sup´erieur `a, <et>) ou est l'une des deux repr´esentations <nom> et <cha | ne de caract`eres>. Notons que les deux terminaux sp´eciaux <nom> et <cha | ne de caract`eres> peuvent avoir ´egalement une s´emantique telle que celle qui est d´efinie ci-apr`es.

Les crochets angulaires et le(s) mot(s) sont soit un symbole non-terminal ou l'un des deux terminaux <cha | ne de caract`eres> ou <nom>. Les cat´egories syntaxiques sont les non-terminaux indiqu´es par un ou plusieurs mots compris entre des crochets angulaires. Pour chaque symbole non-terminal, une r`egle de production est donn´ee soit en grammaire textuelle concr`ete soit en grammaire graphique. Par exemple:

<expression de visibilit´e> ::=

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

Une r`egle 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 constitu´ees par un ou plusieurs symbole(s) non-terminaux et ´eventuellement un ou plusieurs symbole(s) terminaux. Par exemple <expression de visibilit´e>, <identificateur de variable > et <expression> dans l'exemple ci-dessus sont des symboles non-terminaux; VIEW, les parenth`eses et le point sont des symboles terminaux.

.bp

Parfois, le symbole inclut une partie soulign´ee. Cette partie soulign´ee met en relief un aspect s´emantique de ce symbole. Par exemple <identificateur de variable > est syntaxiquement identique à <identificateur>, mais sur le plan s´emantique, il sp´ecifie que l'identificateur doit | tre un identificateur de variable.

A la partie droite du symbole ::=, il existe plusieurs possibilit´es de production de symboles non-terminaux, s´epar´es par des barres verticales | . Par exemple:

<zone de bloc> ::=

<r´ef´erence de bloc graphique>

| <diagramme de bloc>

indique qu'une <zone de bloc> est soit une <r´ef´erence de bloc graphique> ou un <diagramme de bloc>.

Les ´el´ements syntaxiques peuvent | tre regroup´es au moyen d'accolades { | t } , analogues aux parenth`eses du M´eta IV (voir le § 1.5.1). Un groupe entre accolades peut contenir une ou plusieurs barres verticales, indiquant des ´el´ements syntaxiques possibles. Par exemple:

<zone d'interaction de bloc> ::=

{ zone de bloc> | <zone de d´efinition de canal }

La r´ep´etition de groupes entre accolades est indiqu´ee au moyen d'un ast´erisque (*) ou du signe plus (+). Un ast´erisque indique que le groupe est facultatif et peut ult´erieurement | tre r´ep´et´e un nombre quelconque de fois; un signe plus indique que le groupe doit | tre pr´esent et peut | tre r´ep´et´e 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 d´efinition de canal> et peut contenir plusieurs autres <zones de bloc> et <zones de d´efinition de canal>.

Si les ´el´ements syntaxiques sont regroup´es en utilisant des crochets ([| t |), cela indique que le groupe est facultatif. Par exemple:

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

PROCESS <identificateur de processus > [<param`etres formels>]

indique qu'un <en-t | te de processus> peut, mais pas n´ecessairement, contenir des <param`etres formels>.

1.5.3 *M´etalangage applicable à la grammaire graphique*

En ce qui concerne la grammaire graphique, le m´etalangage d´ecrit au § 1.5.2 est compl´et´e avec les m´etasymboles suivants:

a) **contains**

b) **is associated with**

c) **is followed by**

d) **is connected to**

e) **set**

Le m´etasymbole **set** est un op´erateur postfix´e agissant sur les ´el´ements syntaxiques plac´es à l'int´erieur d'accolades et qui le pr´ec`edent imm´ediatement, il d´esigne un ensemble (non ordonn´e) d'´el´ements. Chaque ´el´ement peut | tre un groupe quelconque d'´el´ements syntaxiques, auquel cas, il faut le d´evelopper avant d'appliquer le m´etasymbole **set** .

Exemple:

{ zone de texte de syst`eme } { 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’interation 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> associ´e. 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 associ´e avec son argument de gauche (comme s'il ´etait <<contenu>> dans cet argument, l'association d´epourvue d'ambigu"it´e 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 dessin) son argument de gauche.

Le m´etasymbole **is connected to** signifie que son argument de droite est reli´e (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>

| **Montage**

| <crochet droit>

| <accolade gauche>

| <ligne verticale>

| <accolade droite>

| <tilde>

| <pointe de fl`eche vers le haut>

<crochet gauche> ::=

[

<crochet droit> ::=

]

<accolade gauche> ::=

{

<ligne verticale> ::=

|

<accolade droite> ::=

}

<tilde> ::=

.~

Montage

<pointe de fl`eche vers le haut> ::=

.^

Montage

<point> ::=

.

<soulign´e> ::=

—

<cha | ne de caract`eres> ::=

<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> ::=

/* <texte> */

<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

| ENDNEWTYPE

```
|       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> repr´esente le caract`ere espace de l'Alphabet n^{o} 5 du CCITT.

Les caract`eres <national> sont repr´esent´es ci-dessus de la m | me fa,con que la version internationale de r´ef´erence de l'Alphabet n$^{o}$ 5 du CCITT (Recommandation T.50). La responsabilit´e de la definition des repr´esentations nationales de ces caract`eres rel`eve des organismes nationaux de normalisation.

.bp

Toutes les <lettre> sont toujours trait´ees comme des majuscules sauf celles qui sont à l’int´erieur d’une <cha | ne de caract`eres>. (Le traitement des <national> peut | tre d´efini par les organismes nationaux de normalisation.)

Une <unit´e lexicale> se termine par le premier caract`ere qui ne peut pas faire partie de l’<unit´e lexicale> conform´ement à la syntaxe sp´ecifi´ee ci-dessus. Lorsqu’un caract`ere <soulign´e> est suivi d’un ou plusieurs caract`eres de commande (les caract`eres de commande sont d´efinis comme dans la Recommandation T.50) ou espaces, tous ces caract`eres (y compris le <soulign´e>) sont ignor´es, par exemple A _B correspond au m | me <nom> que AB. Cet emploi de <soulign´e> permet de r´epartir des <unit´e lexicale> sur plus d’une ligne.

Lorsqu’un caract`ere <soulign´e>est suivi d’un <mot> dans un <nom>, il est autoris´e à sp´ecifier un ou plusieurs caract`eres de commande ou espaces, au lieu du caract`ere <soulign´e>, pour autant que l’un des <mot> englobant le caract`ere <soulign´e> ne forme pas un <mot cl´e>, par exemple A B d´esigne le m | me <nom> que A _B.

Toutefois, dans certains cas, l’absence de <soulign´e> dans les <nom> est ambigu"e. Les r`egles suivantes s’appliquent donc:

1) les <soulign´e> dans le <nom> dans un <´el´ement de trajet> doivent | tre sp´ecifi´es explicitement;

2) lorsqu’un ou plusieurs <nom> ou <identificateur> peuvent | tre suivis directement par une <sorte> (par exemple, <d´efinition de variable>, <d´efinition de vue>), les <soulign´e> dans ces <nom> ou <identificateur> doivent | tre sp´ecifi´es explicitement;

3) lorsqu’une <d´efinition de donn´ees> contient des <instantiations de g´en´erateur>, les <soulign´e> du <nom de sorte> suivant le mot cl´e NEWTYPE doivent | tre sp´ecifi´es explicitement.

Un caract`ere de commande a la m | me signification qu’un espace.

Les caract`eres de commande et les espaces peuvent appara | tre un nombre quelconque de fois entre deux <unit´e lexicale>. Un nombre quelconque de caract`eres de contr | le et d’espaces entre deux <unit´e lexicale> ont la m | me signification qu’un espace.

Le caract`ere / imm´ediatement suivi par le caract`ere * marque toujours le d´ebut d’une <note>. Le caract`ere * imm´ediatement suivi par le caract`ere / dans une <note> marque toujours la fin d’une <note>. Une <note> peut | tre ins´er´ee avant ou apr`es toute <unit´e lexicale>.

Des r`egles lexicales particuli`eres s’appliquent dans un <corps de macro> (voir le § 4.2.1).

2.2.2 *R`egles de visibilit´e et identificateurs*

Grammaire abstraite

Identificateur :: *Qualificatif-Nom*

Qualificatif = *El´ement-de-chemin* +

El´ement-de-chemin = *Qualificatif-de-syst`eme* |

Qualificatif-de-bloc |

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

Qualificatif-de-signal |

Qualificatif-de-processus |

Qualificatif-de-proc´edure **|**

Qualificatif-de-sorte **|**

Qualificatif-de-syst`eme* :: *Nom-de-syst`eme

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-proc´edure :: *Nom-de-proc´edure*

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

Qualificatif-de-sorte :: *Nom-de-sorte*

Nom :: *Token*

Grammaire textuelle concr`ete

<identificateur> ::=

[<qualificatif>] <nom>

<qualificatif> ::=

<´el´ement de chemin> { <´el´ement de chemin }

<´el´ement de chemin> ::=

<classe d'unit´e de port´ee> <nom>

.bp

<classe d'unit´e de port´ee> ::=

| SYSTEM

| BLOCK

| SUBSTRUCTURE

| SIGNAL

| PROCESS

| PROCEDURE

| TYPE

| SERVICE

Il n'y a pas de syntaxe abstraite correspondante pour la <classe d'unit´e de port´ee> indiqu´ee par service. Les <nom> et <identificateur> d'entit´es d´efinies dans une <d´efinition de service> sont transform´es en <nom> uniques ou en <identificateur> uniques d´efinis dans la <d´efinition de processus> contenant la <d´efinition de service>.

Le <qualificatif> refl`ete la structure hi´erarchique à partir du niveau du syst`eme vers le contexte de d´efinition, et de mani`ere telle que le niveau du syst`eme est la partie textuelle la plus à gauche. Toutefois, on peut omettre certains <´el´ement de chemin> les plus à gauche, lorsque le premier <´el´ement de chemin> restant le plus à gauche dans le <qualificatif> est unique à l'int´erieur de toute la <d´efinition de syst`eme>.

Il est permis d'omettre certains <´el´ement de chemin> les plus à gauche (sauf pour les <d´efinition diff´er´ee>, voir le § 2.4.1) ou bien tout le <qualificatif>. Lorsque tout le <qualificatif> est omis et que le <nom> d´esigneune entit´e de la classe d'entit´e contenant des variables, des synonymes, des litt´eraux et des op´erateurs (voir la s´emantique ci-apr`es), l'association du <nom> avec une d´efinition doit pouvoir | tre r´esolue par le contexte r´eel. Dans d'autres cas, l'<identificateur> est associ´e à une entit´e qui a son contexte de d´efinition dans l'unit´e de port´ee englobante la plus proche dans laquelle le <qualificatif> de l'<identificateur> est le m | me que la partie la plus à droite du <qualificatif> complet d´esignant cette unit´e de port´ee. Si l'<identificateur> ne contient pas de <qualificatif>, la n´ecessit´e de mise en correspondance des <qualificatif> est omise.

Un sous-signal doit | tre qualifi´e par ses signaux parents, à moins qu'aucun autre signal visible n'existe à cet endroit qui porte le m | me <nom>.

La r´esolution par contexte est possible dans les cas suivants:

a) l'unit´e de port´ee dans laquelle le <nom> est utilis´e n'est pas une <d´efinition partielle de type> et contient une d´efinition ayant ce <nom>. Le <nom> sera li´e à cette d´efinition;

b) l'unit´e de port´ee dans laquelle le <nom> est utilis´e ne contient pas de d´efinition ayant ce <nom> ou l'unit´e de port´ee est une <d´efinition partielle de type>, et dans toute la <d´efinition de syst`eme> il existe exactement une d´efinition visible d'une entit´e qui a le m | me <nom> et à laquelle le <nom> peut | tre li´e sans violer aucune des propri´et´es statiques (compatibilit´e de sorte, etc.) de la construction dans laquelle le <nom> appara | t. Le <nom> sera li´e à cette d´efinition.

Seuls les identificateurs visibles peuvent | tre utilis´es, à l'exception de l'<identificateur de variable> dans une <d´efinition de visibilit´e> et de l'<identificateur> utilis´e à la place d'un <nom> dans une d´efinition r´ef´erenc´ee (c'est-`a-dire une d´efinition extraite de la <d´efinition de syst`eme>).

S´emantique

Les unit´es de port´ee sont d´efinies par le sch´ema suivant:

Grammaire textuelle concr`ete* *Grammaire graphique concr`ete

<d´efinition de syst`eme> <diagramme de syst`eme>

<d´efinition de bloc> <diagramme de bloc>

<d´efinition de processus> <diagramme de processus>

<d´efinition de proc´edure> <diagramme de proc´edure>

<d´efinition de sous-structure de bloc> <diagramme de sous-structure de bloc>

<d´efinition de sous-structure de canal> <diagramme de sous-structure de canal>

<d´efinition de service> <diagramme de service>

<d´efinition partielle de type>

<affinage de signal>

.bp

Une liste de d´efinitions est associ´ee à une unit´e de port´ee. Chaque d´efinition d´efinit une entit´e appartenant à une certaine classe d’entit´e et ayant un nom associ´e. Pour une <d´efinition partielle de type>, la liste de d´efinitions associ´ee comprend les <signature d’op´erateur>, les <signature de litt´eral> et toutes <signature d’op´erateur> et <signature de litt´eral> provenant d’une sorte parente, d’un g´en´erateur d’instance ou impos´e par l’utilisation d’abr´eviations tel le mot cl´e ORDERING (voir le § 5.4.1.8). Il faut remarquer qu’une <d´efinition de visibilit´e> ne d´efinit pas une entit´e.

Bien que les <op´erateur infix´e>, les <op´erateur> avec un point d’exclamation et les <cha | ne de caract`eres> aient leur propre notation syntaxique, ils sont en r´ealit´e des <nom>, ils sont repr´esent´es dans la syntaxe abstraite repr´esent´ee par un nom suit, ils sont ´etudi´es comme s’ils ´etaient (syntaxiquement aussi) des <nom>. Toutefois, les <nom d’´etat >, les <nom de connecteur >, les <nom formel de g´en´erateur >, les <identificateur de valeur > dans les ´equations, les <nom formel de macro > et les <nom de macro > ont des r`egles de visibilit´e particuli`eres et ne peuvent par cons´equent | tre qualifi´es. Leurs r`egles de visibilit´e sont expliqu´ees dans les sections concern´ees.

Chaque entit´e est dite avoir son contexte de d´efinition dans l’unit´e de port´ee qui la d´efinit. Les entit´es sont r´ef´erenc´ees au moyen d’<identificateur>.

Le <qualificatif> dans un <identificateur> sp´ecifie uniquement le contexte de d´efinition du <nom>.

Les classes d’entit´es sont les suivantes:

a) syst`eme

b) blocs

c) canal, acheminement de signaux

d) signaux, temporisateurs

e) processus

f) proc´edures

g) variables (y compris les param`etres formels), synonymes, litt´eraux, op´erateurs

h) sortes

i) g´en´erateurs

j) entit´es import´ees

k) listes de signaux

l) services

m) sous-structures de bloc, sous-structures de canal

Un <identificateur> est dit | tre visible dans une unit´e de port´ee

a) si la partie nom de l’<identificateur> a son contexte de d´efinition dans cette unit´e de port´ee, ou

b) s’il est visible dans l’unit´e de port´ee qui d´efinit cette unit´e de port´ee, ou

c) si l’unit´e de port´ee contient une <d´efinition partielle de type> dans laquelle l’<identificateur> est d´efini, ou

d) si l’unit´e de port´ee contient une <d´efinition de signal> dans laquelle l’<identificateur> se trouve d´efini.

On ne peut avoir deux d´efinitions dans la m | me unit´e de port´ee et appartenant à la m | me classe d’entit´es portant le m | me <nom>. Il existe cependant une exception: les d´efinitions de <signature d’op´erateur> et de <signature de litt´eral> dans la m | me <d´efinition partielle de type> (voir le § 5.2.2): plusieurs op´erateurs et litt´eraux peuvent avoir le m | me <nom> avec diff´erentes <sorte d’argument> ou diff´erentes sortes de <r´esultat>.

On notera une autre exception: les entit´es import´ees. Pour cette classe, les paires de (<nom d’import>, <sorte>) dans <d´efinition d’import> dans l’unit´e de port´ee doivent | tre distinctes.

Dans la grammaire textuelle concr`ete, le nom ou l’identificateur optionnel dans une d´efinition apr`es le mot cl´e de terminaison (ENDSYSTEM, ENDBLOCK, etc.) doit | tre syntaxiquement le m | me que le nom ou l’identificateur qui suit le mot cl´e commen,cant correspondant (respectivement: SYSTEM, BLOCK, etc.).

.bp

2.2.3 *Texte informel*

Grammaire abstraite

Texte informel :: ...

Grammaire textuelle concr`ete

<texte informel> ::=

<cha | ne de caract`eres>

S´emantique

Lorsqu’un texte informel est utilis´e dans une sp´ecification LDS de syst`eme, il signifie que le texte n’est pas en LDS formel, c’est-`a-dire que le LDS ne donne pas de s´emantique. La s´emantique du texte informel peut | tre d´efinie par d’autres moyens.

2.2.4 *R`egles applicables aux dessins*

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

Les fronti`eres des symboles ne doivent ni se superposer, ni se couper. Font exception à cette r`egle les symboles de ligne, c’est-`a-dire le <symbole de canal>, le <symbole d’acheminement de signal>, le <symbole de cr´eation de ligne>, le <symbole de ligne de flot>, le <symbole d’association continu> et le <symbole d’association pointill´e>, qui peuvent se couper. Il n’existe pas d’association logique entre les symboles qui se coupent.

Le m´etasymbole **is followed by** implique un <symbole de ligne de flot>.

Les symboles de ligne peuvent | tre constitu´es par un ou plusieurs segments de droite en trait plein.

Les fl`eches sont n´ecessaires 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’´etat suivant>. Dans d’autres cas, ces fl`eches sont facultatives sur les <symbole de ligne de flot>. Les <symbole de ligne de flot> sont horizontaux ou verticaux.

On peut utiliser des images sym´etriques verticales des <symbole d’entr´ee>, des <symbole de sortie>, des <symbole de commentaire> et des <symbole d’extension de texte>.

L’argument de la partie droite du m´etasymbole **is associated with** doit
| tre plus proche de l’argument de gauche que tout autre symbole graphique. Les ´el´ements syntaxiques de l’argument de droite doivent pouvoir

| tre distingu´es les uns des autres.

Le texte situ´e à l’int´erieur d’un symbole graphique doit | tre lu de la gauche vers la droite, en partant du coin sup´erieur gauche. La limite droite du symbole est interpr´et´ee comme un caract`ere de nouvelle ligne, indiquant le cas ´ech´eant que la lecture doit continuer au point le plus à gauche de la ligne suivante.

2.2.5 *Subdivision des diagrammes*

La d´efinition qui suit concernant la division des diagrammes ne fait pas partie de la *grammaire graphique concr`ete* , n´eanmoins, on utilise le m | me m´etalangage.

<page> ::=

<symbole de cadre> **contains**

{ zone d’en-t | te> <zone de num´ero de page>

{ unit´e syntaxique } }

<zone d’en-t | te> ::=

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

<zone de num´ero de page> ::=

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

<num´ero de page> ::=

<nom de litt´eral >

<nombre de pages> ::=

<nom de litt´eral de naturel >

.bp

La <page> est un non-terminal de d´epart, par cons´equent, il n’est associ´e à aucune r`egle de production. Un diagramme peut | tre subdivis´e en un nombre de <page>, auquel cas le <symbole de cadre> d´elimitant le diagramme et l’<en-t | te> du diagramme est remplac´e par un <symbole de cadre> et un <en-t | te> pour chaque <page>.

L’utilisateur du LDS peut choisir les <symbole de cadre> impos´es par la limite du support sur lequel les diagrammes sont reproduits.

Afin d’avoir une s´eparation nette entre la <zone d’en-t | te> et la <zone de num´ero de page>, le <symbole de texte implicite> n’est pas mat´erialis´e, il est sous-entendu. La <zone d’en-t | te> est plac´ee au coin sup´erieur gauche du <symbole de cadre>, la <zone de num´ero de page> est situ´ee au coin sup´erieur droit du <symbole de cadre>. L’<en-t | te> et l’<unit´e syntaxique> d´ependent du type de diagramme.

2.2.6 *Commentaire*

Un commentaire est une notation qui repr´esente des commentaires associ´es avec des symboles ou du texte.

Dans la *grammaire textuelle concr`ete* , on utilise deux formes de commentaires. La premi`ere forme est la <note> d´efinie au § 2.2.1.

Des exemples sont donn´es aux figures 2.9.1 et 2.9.3.

La syntaxe concr`ete de la deuxi`eme forme est:

<fin> ::=

[<commentaire>];

<commentaire> ::=

COMMENT <cha | ne de caract`eres>

Un exemple est donn´e à la figure 2.9.2.

Dans la *grammaire graphique concr`ete* , la syntaxe suivante est utilis´ee:

<zone de commentaire> ::=

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

is connected to <symbole d'association pointill´e>

<symbole de commentaire> ::=

.rs

Figure, p. 29 (E)

<symbole d'association pointill´e> ::=

Une des extr´emit´es du <symbole d'association pointill´e> doit | tre reli´ee au milieu du segment vertical du <symbole de commentaire>.

Un <symbole de commentaire> peut | tre reli´e à tout symbole graphique au moyen d'un <symbole d'association pointill´e>. Le <symbole de commentaire> est consid´er´e comme ´etant un symbole ferm´e en compl´etant (par la pens´ee) le rectangle afin d'entourer le texte. Il contient le texte du commentaire se rapportant au symbole graphique.

Un exemple est donn´e 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 extr´emit´es du <symbole d’association continu> doit | tre reli´ee au milieu du segment vertical du <symbole d’extension de texte>.

Un <symbole d’extension de texte> peut | tre reli´e à tout symbole graphique au moyen d’un <symbole d’association continu>. Le <symbole d’extension de texte> est consid´er´e comme ´etant un symbole ferm´e en compl´etant (par la pens´ee) le rectangle.

Le texte contenu dans le symbole d’extension de texte> est la suite du texte dans le symbole graphique et est consid´er´e comme ´etant contenu dans ce symbole.

2.2.8 *Symbole de texte*

Le <symbole de texte> est utilis´e dans tout <diagramme>. Le contenu d´epend du diagramme.

<symbole de texte> ::=

.rs

Figure, p. 30 (E)

2.3 *Concepts de base concernant les donn´ees*

Le concept de donn´ees dans le LDS est expos´e dans le d´etail au § 5; plus pr´ecis´ement en ce qui concerne la terminologie du LDS pour ce qui est des donn´ees et la facilit´e à d´efinir des nouveaux types de donn´ees et les facilit´es concernant les donn´ees pr´ed´efinies.

Les donn´ees apparaissent dans les d´efinitions du type de donn´ees, les expressions, l’application d’op´erateurs, les variables, les valeurs et les litt´eraux.

2.3.1 *D´efinitions des types de donn´ees*

Les donn´ees dans le LDS sont principalement abord´ees sous l’aspect type de donn´ees. Un type de donn´ees d´efinit un ensemble de valeurs, un ensemble d’op´erateurs qui peut | tre appliqu´e à ces valeurs, et un ensemble de r`egles alg´ebriques (´equations) qui d´efinissent le comportement de ces op´erateurs lorsqu’ils sont appliqu´es aux valeurs. Les valeurs, les op´erateurs et les r`egles alg´ebriques d´efinissent tous ensemble les propri´et´es du type de donn´ees. Ces propri´et´es sont d´efinies par les d´efinitions de types de donn´ees.

Le LDS permet la d´efinition de tout type de donn´ees qui est n´ecessaire, y compris les m´ecanismes de composition (type composite) avec pour seule contrainte qu'une telle d´efinition puisse | tre sp´ecifi´ee de mani`ere formelle. En revanche, pour les langages de programmation, il y a des consid´erations de mise en oeuvre qui n´ecessitent que l'ensemble des types de donn´ees disponibles et, en particulier, les m´ecanismes de composition mis à disposition soient limit´es (tableau, structure, etc.).

2.3.2 *Variables*

Les variables sont des objets qui peuvent | tre associ´es à une valeur par une affectation. Quand on acc`ede à la variable, on renvoie la valeur associ´ee.

2.3.3 *Valeurs et litt´eraux*

Un ensemble de valeurs avec certaines caract´eristiques est appel´e sorte. Les op´erateurs sont d´efinisà partir et vers les valeurs des sortes. Par exemple, l'application de l'op´erateur somme (<<+>>) à partir et vers les valeurs de la sorte enti`ere est valable, tandis que la somme de la sorte bool´eenne ne l'est pas.

.bp

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

Pour la plupart des sortes, il y a des formes litt´erales qui d´ecrivent les valeurs de la sorte (par exemple, pour les entiers <<2>> est utilis´e de pr´ef´erence à <<1+1>>). Il peut y avoir plusieurs litt´eraux qui d´ecrivent la m | me valeur (par exemple, 12 et 012 peuvent | tre utilis´es pour d´ecrire la m | me valeur enti`ere). La m | me description litt´erale peut | tre utilis´ee pour plus d'une sorte; par exemple, 'A' est à la fois un caract`ere et une cha | ne de caract`eres de longueur 1. Certaines sortes peuvent ne pas avoir de litt´eraux; par exemple, une valeur composite n'a souvent pas de litt´eraux en propre mais a des valeurs qui sont d´efinies par des op´erations de composition sur les valeurs de ses composantes.

2.3.4 *Expressions*

Une expression d´ecrit une valeur. Si une expression ne contient pas de variable, par exemple, si elle est un litt´eral d'une sorte donn´ee, chaque occurrence de l'expression d´ecrira toujours la m | me valeur. Une expression qui contient des variables peut | tre interpr´et´ee comme diff´erentes valeurs durant l'interpr´etation d'un syst`eme LDS selon la valeur associ´ee aux variables.

2.4 *Structure du syst`eme*

2.4.1 *D´efinitions diff´er´ees*

Une <d´efinition diff´er´ee> est une d´efinition qui a ´et´e supprim´ee de son contexte de d´efinition pour accro | tre la visibilit´e. Elle est analogue à

une d´efinition de macro (voir le § 4.2), mais elle est <<appel´ee>> d'un seul endroit exactement (le contexte de d´efinition) en utilisant une r´ef´erence.

Grammaire concr`ete

<d´efinition diff´er´ee> ::=

<d´efinition> | <diagramme>

<d´efinition de syst`eme> ::=

{ d´efinition textuelle de syst`eme> | <diagramme de syst`eme }

{ d´efinition diff´er´ee }

<d´efinition> ::=

<d´efinition de bloc>

| <d´efinition de processus>

| <d´efinition de proc´edure>

| <d´efinition de sous-structure de bloc>

| <d´efinition de sous-structure de canal>

| <d´efinition de service>

| <d´efinition de macro>

<diagramme> ::=

<diagramme de bloc>

| <diagramme de processus>

| <diagramme de proc´edure>

| <diagramme de sous-structure de bloc>

| <diagramme de sous-structure de canal>

| <diagramme de service>

| <diagramme de macro>

Pour chaque <d´efinition diff´er´ee>, sauf pour des <d´efinition de macro> et des <diagramme de macro>, on doit avoir une r´ef´erence dans la <d´efinition de syst`eme>, le <diagramme de syst`eme>, ou une autre <d´efinition diff´er´ee>.

Pour chaque r´ef´erence on doit avoir une <d´efinition diff´er´ee> correspondante.

Dans chaque <d´efinition diff´er´ee>, on doit avoir un <identificateur> plac´e imm´ediatement apr`es le mot cl´e initial. Dans cet <identificateur>, le <qualificatif> doit | tre soit complet, soit omis. Si le <qualificatif> est omis, le <nom> doit | tre unique dans la d´efinition de syst`eme, à l'int´erieur de la classe d'entit´e pour la <d´efinition diff´er´ee>. On ne peut pas sp´ecifier un qualificatif dans l'<identificateur> apr`es le mot cl´e initial pour les d´efinitions qui ne sont pas des <d´efinition diff´er´ee> (c'est-`a-dire qu'un <nom> doit | tre sp´ecifi´e pour les d´efinitions normales).

S´emantique

Avant de pouvoir analyser une <d´efinition de syst`eme concret>, chaque r´ef´erence doit | tre remplac´ee par la <d´efinition diff´er´ee> correspondante. Dans cette substitution, l'<identificateur> de la <d´efinition diff´er´ee> est remplac´e par le <nom> dans la r´ef´erence.

.bp

2.4.2 *Syst`eme*

Grammaire abstraite

D´efinition-de-syst`eme* :: *Nom-de-syst`eme

D´efinition-de-bloc -**set**

D´efinition-de-canal -**set**

D´efinition-de-signal -**set**

D´efinition-de-type-de-donn´ees

D´efinition-de-type-de-synonyme -**set**

Nom-de-syst`eme = *Nom*

Une *d´efinition-de-syst`eme* a un nom qui peut | tre utilis´e dans les qualificatifs.

La *d´efinition-de-syst`eme* doit au moins contenir une *d´efinition-de-bloc* .

Les d´efinitions de tous les signaux, canaux, types de donn´ees, types de synonymes utilis´es dans l'interface avec l'environnement et entre les blocs du syst`eme sont contenues dans la *d´efinition-de-syst`eme* donn´ees pr´ed´efinies sont consid´er´ees comme ´etant d´efinies au niveau du syst`eme.

Grammaire textuelle concr`ete

<d´efinition textuelle de syst`eme> ::=

SYSTEM <nom de syst`eme > <fin>

{ d´efinition de bloc>

| <r´ef´erence textuelle de bloc>

| <d´efinition de canal>

| <d´efinition de signal>

| <d´efinition de liste de signaux>

| <d´efinition de s´election>

| <d´efinition de macro>

| <d´efinition de donn´ees } |

ENDSYSTEM [<nom de syst`eme >] <fin>

<r´ef´erence textuelle de bloc> ::=

BLOCK <nom de bloc > REFERENCED <fin>

<d´efinition de s´election> est d´efinie au § 4.3.3, <d´efinition de macro> au § 4.2, <d´efinition de donn´ees> au § 5.5.1, <d´efinition de bloc> au § 2.4.3, <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.

Un exemple de <d´efinition de syst`eme> est donn´e à la figure 2.9.5 du § 2.9.

Grammaire graphique concr`ete

<diagramme de syst`eme> ::=

<symbole de cadre> **contains**

{ en-t | te de syst`eme>

{ { zone texte de syst`eme } |

{ 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´erence de bloc graphique>

| <diagramme de bloc>

<r´ef´erence 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´equence 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*

D´efinition-de-processus -**set**

D´efinition-de-signal -**set**

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

D´efinition-d'acheminement-de-signal -**set**

D´efinition-de-type-de-donn´ees

D´efinition-de-type-de-synonyme -**set**

[*D´efinition-de-sous-structure-de-bloc*]

Nom-de-bloc = *Nom*

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

Il est possible de subdiviser les blocs en sp´ecifiant la *d´efinition-de-sous-structure-de-bloc* ; cet aspect du langage est ´etudi´e au § 3.2.2.

.bp

Grammaire textuelle concr`ete

<d´efinition de bloc> ::=

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

{ d´efinition de signal>

| <d´efinition de liste de signaux>

| <d´efinition de processus>

| <r´ef´erence textuelle de processus>

| <d´efinition d'acheminement de signal>

| <d´efinition de macro>

| <d´efinition de donn´ees>

| <d´efinition de s´election>

| <connexion de canal vers acheminement }

[<d´efinition de sous-structure de bloc | r´ef´erence textuelle de sous-structure de bloc>]

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

<r´ef´erence textuelle de processus> ::=

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

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

<d´efinition de sous-structure de bloc> au § 3.2.2, <r´ef´erence de sous-structure de bloc textuelle> au § 3.2.2, <d´efinition de macro> au § 4.2.2, <d´efinition de donn´ees> au § 5.5.1.

Un exemple de <d´efinition de bloc> est montr´e à la figure 2.9.7 au § 2.9.

Grammaire graphique concr`ete

<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 reli´e à un acheminement de signaux dans le <diagramme de bloc>. Il se trouve plac´e à l'ext´erieur du <symbole de cadre>, à proximit´e de l'extr´emit´e 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 syst`eme>

<zone d'interaction de processus> ::=

{ zone de processus>

| <zone de ligne de cr´eation>

| <zone de d´efinition d'acheminement de signal }

<zone de processus> ::=

<r´ef´erence graphique de processus | diagramme de processus>

<r´ef´erence 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 d´efini au § 2.4.4.

<zone de ligne de cr´eation> ::=

<symbole de ligne de cr´eation>

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

<symbole de ligne de cr´eation> ::=

La t | te de fl`eche plac´ee sur le <symbole de ligne de cr´eation> indique la <zone de processus> sur laquelle le processus de cr´eation 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 signauxs>] <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’intances contenus dans le nombre-d’instances sont tir´es 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 limit´e.

Le <nombre d’instances> utilis´e dans la d´erivation est le suivant:

a) s’il n’y a pas de <r´ef´erence textuelle de processus> pour le processus, le <nombre d’instances> dans la <d´efinition de processus> est utilis´e. 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 <d´efinition de processus> et le <nombre d’instances> dans une <r´ef´erence 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 <d´efinition de processus> soit le <nombre d’instances> dans une <r´ef´erence textuelle de processus> est omis, le <nombre d’instances> est celui qui est pr´esent;

d) si à la fois le <nombre d’instances> dans la <d´efinition de processus> et le <nombre d’instances> dans une <r´ef´erence textuelle de processus> sont sp´ecifi´es, les deux <nombre d’instances> doivent |tre ´egaux lexicalement et ce <nombre d’instances> est utilis´e.

Une relation analogue s’applique au <nombre d’instances> sp´ecifi´e dans le <diagramme de processus> et dans la <r´ef´erence textuelle de processus> d´efinis ci-dessous.

<d´efinition de signal> est d´efinie au § 2.5.4, <d´efinition de liste de signaux> au § 2.5.5, <d´efinition de visibilit´e> au § 2.6.1.2, <d´efinition de variable> au § 2.6.1.1, <d´efinition de proc´edure> au § 2.4.5, <d´efinition de temporisateur> au § 2.8, <d´efinition de macro> au § 4.2.2, <d´efinition d’import> au § 4.1.3, <d´efinition de s´election> au § 4.3.3, <expression simple> au § 4.3.2, <d´ecomposition de service> au § 4.10.1, <d´efinition de donn´ees> au § 5.5.1.

Le <nombre initial> d’instances doit |tre inf´erieur ou ´egal au <nombre maximal> et le <nombre maximal> doit |tre sup´erieur à z´ero.

L’utilisation du terme <ensemble des signaux d’entr´ee valides> est pr´ecis´ee au § 2.5.2, *mod`ele* .

Un exemple de <d´efinition de processus> est donn´e à la figure 2.9.9 du § 2.9.

Grammaire graphique concr`ete

<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 reli´e à un trajet de signal dans le <diagramme de processus>. Il est plac´e à l'ext´erieur du <symbole de cadre> à proximit´e de l'extr´emit´e du trajet du signal au <symbole de cadre>.

<zone de texte de processus> ::=

<symbole de texte> **contains** {

[<ensemble des signaux d'entr´ee valides>]

{ d´efinition de signal>

| <d´efinition de liste de signaux>

| <d´efinition de variable>

| <d´efinition de visibilit´e>

| <d´efinition d'import>

| <d´efinition de donn´ees>

| <d´efinition de macro>

| <d´efinition de temporisateur>

| <d´efinition de s´election } }

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

PROCESS { nom de processus
| identificateur de processus
}

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

[<param`etres formels>]

<zone graphique de processus> ::=

<zone de d´epart { zone d'´etat | zone de connecteur d'entr´ee }

<d´efinition de signal> est d´efinie au § 2.5.4, <d´efinition de liste de signaux> au § 2.5.5, <d´efinition de visibilit´e> au § 2.6.1.2, <d´efinition de variable> au § 2.6.1.1, <zone de proc´edure> au § 2.4.5, <d´efinition de temporisateur> au § 2.8, <d´efinition de macro> et <diagramme de macro> au § 4.2.2, <d´efinition d'import> au § 4.1.3, <d´efinition de s´election> au § 4.3.3, <d´efinition de donn´ees> au § 5.5.1, <zone de d´epart> au § 2.6.2, <zone d'´etat> au § 2.6.3, <zone de connecteur d'entr´ee> au § 2.6.6 et <zone d'interaction de service> au § 4.10.1.

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

S´emantique

Une d´efinition de processus introduit le type d'un processus qui est destin´e à repr´esenter un comportement dynamique.

Dans le *nombre-d'instances,* la premi`ere valeur repr´esente le *nombre-d'instances* du processus qui existe à la cr´eation du syst`eme, la deuxi`eme valeur repr´esente le nombre maximal d'instances simultan´ees du type de processus.

Une instance de processus est une machine à ´etats finis ´etendue communicante qui assure un certain nombre d'actions, appel´ees transitions, suite à la r´eception d'un signal donn´e, chaque fois qu'elle se trouve dans un certain ´etat. La r´ealisation de la transition aboutit à l'attente du processus dans un autre ´etat, qui n'est pas n´ecessairement diff´erent de l'´etat d'origine.

Le concept de machine à ´etats finis a ´et´e ´etendu en ce sens que l'´etat r´esultant d'une transition, à c | t´e du signal provoquant la transition, peut | tre affect´e par des d´ecisions prises à partir de variables connues du processus.

Plusieurs instances du m | me type de processus peuvent exister au m | me instant et s'ex´ecuter de mani`ere asynchrone et en parall`ele les unes aux autres, et avec d'autres instances de diff´erents types de processus dans le syst`eme.

Lorsqu'un syst`eme est cr´e´e, les processus initiaux sont cr´e´es dans un ordre al´eatoire. La communication des signaux entre les processus ne commence que lorsque tous les processus initiaux ont ´et´e cr´e´es. Les param`etres formels de ces processus initiaux sont initialis´es avec une valeur non d´efinie.

Les instances de processus existent à partir du moment où un syst`eme est cr´e´e ou peuvent | tre cr´e´ees par une des actions de demande de cr´eation qui lance les processus à interpr´eter; leur interpr´etation commence lorsque l'action de d´epart est interpr´et´ee; des actions d'arr | t peuvent arr | ter leur existence.

Les signaux re,cus par les instances de processus sont appel´es signaux d'entr´ee, et les signaux envoy´es aux instances de processus sont appel´es signaux de sortie.

Les signaux peuvent | tre trait´es par une instance de processus seulement lorsque celle-ci se trouve dans un certain ´etat. L'ensemble complet de signaux d'entr´ee 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'entr´ee valides>, des signaux implicites et des signaux de temporisation.

.bp

Un acc`es d'entr´ee unique est associ´e à chaque instance de processus. Lorsqu'un signal d'entr´ee parvient au processus, il est appliqu´e à l'acc`es d'entr´ee de l'instance de processus.

Le processus peut | tre soit mis en attente, en occupant un ´etat, soit en activit´e, en effectuant une transition. Pour chaque ´etat, il existe un ensemble de signaux de mise en r´eserve (voir ´egalement le § 2.6.5). Dans le cas d'attente dans un ´etat, le premier signal entrant dont l'identificateur ne figure pas dans l'ensemble des signaux de mise en r´eserve est extrait de la file d'attente, et trait´e par le processus.

L'acc`es d'entr´ee 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 ordonn´e dans la file d'attente, selon l'ordre d'arriv´ee. Si deux ou plusieurs signaux arrivent simultan´ement, ils sont ordonn´es arbitrairement.

Lorsqu'un processus est cr´e´e, on lui attribue un acc`es d'entr´ee vide, et il y a alors cr´eation de variables locales auxquelles on affecte des valeurs.

Les param`eres formels sont des variables qui sont cr´e´ees soit lorsque le syst`eme est cr´e´e (mais aucun param`etre r´eel ne lui est transmis et par cons´equent ces param`etres ne sont pas initialis´es), soit lorsque l'instance de processus est dynamiquement cr´e´ee.

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 r´esultat pour:

a) l'instance de processus (SELF);

b) l'instance du processus cr´eateur (PARENT);

c) l'instance de processus la plus r´ecente cr´e´ee par le processus (OFFSPRING);

d) l'instance de processus en provenance de laquelle le dernier signal entrant a ´et´e utilis´e (SENDER) (voir ´egalement le § 2.6.4).

Ces expressions sont expliqu´ees dans le d´etail au § 5.5.4.3.

SELF, PARENT, OFFSPRING et SENDER peuvent | tre utilis´es dans des expressions à l'int´erieur des instances de processus.

Pour toutes les instances de processus qui se trouvent pr´esentes au moment de l'initialisation du syst`eme, l'expression pr´ed´efinie PARENT pr´esente toujours la valeur NULL.

Pour toutes les instances de processus nouvellement cr´e´ees, les expressions pr´ed´efinies SENDER et OFFSPRING ont la valeur NULL.

2.4.5 *Proc´edure*

Les proc´edures sont d´efinies au moyen de d´efinitions de proc´edure. On fait appel à une proc´edure au moyen d'un appel de proc´edure faisant r´ef´erence à la d´efinition de proc´edure. Des param`etres sont associ´es à un appel de proc´edure: on les emploie à la fois pour fournir les valeurs et pour limiter la port´ee des variables pour l'ex´ecution de la proc´edure. C'est du m´ecanisme de transfert des param`etres que d´ependent les variables affect´ees par l'interpr´etation d'une proc´edure.

Grammaire abstraite

D´efinition-de-proc´edure :: *Nom-de-proc´edure*

Param`etre-formel-de-proc´edure |

D´efinition-de-proc´edure-**set**

D´efinition-de-type-de-donn´ees

D´efinition-de-type-de-synonyme-**set**

D´efinition-de-variable-**set**

Graphe-de-proc´edure

Nom-de-proc´edure = *Nom*

Param`etre-formel-de-proc´edure* = *Param`etre-d'entr´ee |

Param`etre-d'entr´ee-et-de-sortie

Param`etre-d'entr´ee :: *Nom-de-variable*

Identificateur-de-r´ef´erence-de-sorte

Param`etre-d'entr´ee-et-de-sortie :: *Nom-de-variable*

Identificateur-de-r´ef´erence-de-sorte

Graphe-de-proc´edure :: *Noeud-de-d´epart-de-proc´edure*

Noeud-d'´etat-**set**

Noeud-de-d´epart-de-proc´edure :: *Transition*

Grammaire textuelle concr`ete

<d´efinition de proc´edure> ::=

PROCEDURE { identificateur de proc´edure
| nom de proc´edure
} <fin>

[<param`etres formels de proc´edure> <fin>]

{ d´efinition de donn´ees>

| <d´efinition de variable>

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

| <d´efinition de proc´edure>

| <d´efinition de s´election>

| <d´efinition de macro }

<corps de proc´edure>

ENDPROCEDURE [<nom de proc´edure
| identificateur de proc´edure >] <fin>

<param`etres formels de proc´edure> ::=

FPAR <param`etre formel de variable>

{ <param`etre formel de variable }

<param`etre formel de variable>::=

[IN/OUT

| IN]

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

<corps de proc´edure> ::=

<corps de processus>

<d´efinition de variable> est d´efinie au § 2.6.1.1, <r´ef´erence textuelle de proc´edure> au § 2.4.4, <d´efinition de macro> au § 4.2, <d´efinition de s´election> au § 4.3.3, <d´efinition de donn´ees> au § 5.5.1, <sorte> au § 5.2.2.

Dans une <d´efinition de proc´edure>, la <d´efinition de variable> ne peut contenir les expressions REVAELED, EXPORTED, REVAELED EXPORTED, EXPORTED REVEALED <nom de variable > (voir le § 2.6.1). Un exemple de <d´efinition de proc´edure> est donn´e à la figure 2.9.11.

Grammaire graphique concr`ete

<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´erence 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´ef´erence graphique de proc´edure> ::=

<symbole de proc´edure> **contains** <nom de proc´edure >

<symbole de proc´edure> ::=

Figura 9, p.

<en-t | te de proc´edure> ::=

PROCEDURE { nom de proc´edure
| identificateur de proc´edure
}

[<param`etres formels de proc´edure>]

<zone de graphe de proc´edure> ::=

<zone de d´epart de proc´edure>

{ zone de d’´etat | zone de connecteur d’entr´ee> }

<zone de d´epart de proc´edure> ::=

<symbole de d´epart de proc´edure> **is followed by** <zone de transition>

<symbole de d´epart de proc´edure> ::=

.rs

Figura 10, p.

<d´efinition de variable> est d´efinie au § 2.6.1.1, <zone de transition> au § 2.6.7.1, <zone d’´etat> au § 2.6.3, <zone de connecteur d’entr´ee> au § 2.6.6, <d´efinition de macro> et <diagramme de macro> au § 4.2, <d´efinition de s´election> au § 4.3.3, <d´efinition de donn´ees> au § 5.5.1.

Un exemple de <diagramme de proc´edure> est montr´e à la figure 2.9.12 au § 2.9.

S´emantique

Une proc´edure est un moyen de donner un nom à un assemblage d’objets et de le repr´esenter par une r´ef´erence unique. Les r`egles relatives aux proc´edures imposent une discipline sur la mani`ere dont l’assemblage d’objets est choisi et elles limitent la port´ee du nom des variables d´efinies dans la proc´edure.

Une variable de proc´edure est une variable locale à la proc´edure qui ne peut appara | tre, | tre visible, | tre export´eeou | tre import´ee. Elle est cr´e´e lors de l’interpr´etation du noeud de d´epart de proc´edure et cesse d’exister lors de l’interpr´etation du noeud de retour du graphe de proc´edure.

Lorsqu’une d´efinition de proc´edure est interpr´et´ee, son graphe de proc´edure est ´egalement interpr´et´e.

Une d´efinition de proc´edure est interpr´et´ee seulement lorsqu’une instance de processus l’appelle, et l’interpr´etation est faite par l’instance de processus.

L'interpr´etation d'une d´efinition de proc´edure provoque la cr´eation d'une instance de proc´edure et l'interpr´etation commence de la mani`ere suivante:

a) une variable locale est cr´e´ee pour tout *param`etre-d'entr´ee* , ayant le *nom* et la *sorte* du *param`etre-d'entr´ee* param`etre effectif correspondant, qui peut ne pas | tre d´efini;

b) lorsqu'un param`etre effectif est vide, la valeur attribu´ee au param`etre formel correspondant n'est pas d´efinie;

c) un param`etre formel n'ayant pas d'attribut explicite, a un attribut IN implicite;

.bp

d) une variable locale est cr´e´ee pour chaque *d´efinition-de-variable* dans la *d´efinition-de-proc´edure* ayant le *nom* et la *sorte* de la *d´efinition-de-variable* ;

e) chaque *param`etre-d'entr´ee-sortie* d´ecrit un nom de synonyme pour la variable qui est donn´ee dans l'expression des param`etres effectifs. Ce nom de synonyme est utilis´e tout au long de l'interpr´etation du *graphe-de-proc´edure* lorsqu'on se r´ef`ere 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-d´epart-de-proc´edure* est interpr´et´ee.

2.5 *Communication*

2.5.1 *Canal*

Grammaire abstraite

D´efinition-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 | tre achemin´es sur le ou les chemins-de-canal d´efinis par le canal.

Une extr´emit´e du canal doit au moins | tre un bloc.

Si les deux points extr**|**mes sont des blocs, ceux-ci doivent | tre diff´erents.

La ou les extr´emit´e(s) des blocs doivent | tre d´efinies dans la m | me unit´e de port´ee que celles o`u est d´efini le canal.

Grammaire textuelle concr`ete

<d´efinition de canal> ::=

CHANNEL <nom de canal >

<trajet de canal>

[<trajet de canal>]

[<d´efinition de sous-structure de canal>

| <r´ef´erence 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 d´efinie au § 2.5.5, <d´efinition de sous-structure de canal> et <r´ef´erence textuelle de sous-structure de canal> au § 3.2.3.

Lorsque deux <trajet de canal> sont d´efinis, le sens de l'un doit | tre oppos´e par rapport au sens de l'autre.

.bp

Grammaire graphique concr`ete

<zone de d´efinition 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 reli´e au <diagramme de sous-structure de bloc> d´elimit´e par le <symbole de cadre>. L'<identificateur de bloc > identifie un bloc externe comme ´etant une extr´emit´e de canal pour le <diagramme de sous-structure de canal> d´elimit´e 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 d´efinie 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 fl`eche plac´ee 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 fl`eche à laquelle elle est associ´ee pour ´eviter toute ambigu"it´e.

S´emantique

Un canal est un moyen de transport pour les signaux. Un canal peut | tre consid´er´e comme ´etant un ou deux trajets de canal unidirectionnels ind´ependants entre deux blocs ou entre un bloc et son environnement.

Les signaux achemin´es par les canaux sont v´ehicul´es jusqu’au point extr | me de destination.

Au point extr | me de destination d’un canal, les signaux se pr´esentent dans le m | me ordre que celui des signaux au point d’origine. Si deux ou plusieurs signaux arrivent simultan´ement sur le canal, ils sont ordonn´es arbitrairement.

Un canal peut retarder l’acheminement des signaux sur le canal. Cela signifie qu’une file d’attente du type premier-entr´e-premier-sorti (FIFO) se trouve associ´ee à chaque direction dans un canal. Quand un signal se pr´esente sur le canal, il est plac´e dans la file d’attente. Apr`es un intervalle de temps non d´etermin´e et non constant, la premi`ere instance de signal dans la file d’attente est lib´er´ee et appliqu´ee à l’un des canaux ou l’un des trajets de signaux qui se trouvent connect´es au canal.

Plusieurs canaux peuvent exister entre deux points d’extr´emit´es. Le m | me type de signal peut | tre achemin´e sur des canaux diff´erents.

2.5.2 *Acheminement du signal*

Grammaire abstraite

D´efinition-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 concr`ete

<d´efinition 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 d´efinie au § 2.5.5.

Lorsque deux <trajet d'acheminement du signal> sont d´efinis, l'un doit
| tre en sens oppos´e à l'autre.

Grammaire graphique concr`ete

<zone de d´efinition 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 t | te de fl`eche à une extr´emit´e (une direction) ou une t | te de fl`eche à chaque extr´emit´e (deux directions) pour montrer la direction du flot des signaux.

Pour chaque fl`eche situ´ee sur le <symbole d'acheminement de signal>, il doit exister une <zone de liste de signaux>. Une <zone de liste de signaux> doit | tre suffisamment proche de la fl`eche à laquelle elle est associ´ee afin d'´eviter toute ambigu"it´e.

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

S´emantique

Un acheminement de signal est un moyen de transport pour les signaux. Un acheminement de signal peut | tre consid´er´e comme ´etant un ou deux trajets d'acheminement de signal unidirectionnels et ind´ependants entre deux processus ou entre un processus et son environnement.

Les signaux achemin´es par des acheminements de signaux sont d´elivr´es au point extr | me 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 m | me type. Si tel est le cas, l'interpr´etation du noeud-de-sortie a pour cons´equence implicite que le signal est appliqu´e directement à l'acc`es d'entr´ee du processus de destination.

Plusieurs trajets de signaux peuvent exister entre deux points extr | mes. Le m | me type de signal peut | tre achemin´e sur diff´erents acheminements de signaux.

Mod`ele

Un <ensemble de signaux d'entr´ee valides> contient les signaux que le processus est autoris´e à recevoir. Un <ensemble de signaux d'entr´ee valides> ne doit toutefois pas contenir de signaux de temporisateur. Si une <d´efinition de bloc> contient des <d´efinition d'acheminement de signal>, l'<ensemble de signaux d'entr´ee 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 <d´efinition de bloc> ne contient aucune <d´efinition d'acheminement de signal>, toutes les <d´efinition de processus> dans cette <d´efinition de bloc> doivent contenir un <ensemble de signaux d'entr´ee valides>. Dans ce cas, les <d´efinition d'acheminement de signal> et les <connexion de canal à acheminement> sont tir´ees de l'<ensemble de signaux d'entr´ee valides>, des<sortie> et des canaux qui se terminent à la fronti`ere des blocs. Les signaux correspondant à une direction donn´ee entre deux processus dans l'acheminement de signal implicite constituent l'intersection des signaux sp´ecifi´es dans l'<ensemble de signaux d'entr´ee valides> du processus de destination et les signaux mentionn´es dans une sortie du processus d'origine. Si l'un des points extr | mes est l'environnement, l'ensemble d'entr´ee/ensemble de sortie est constitu´e par les signaux achemin´es par le canal dans la direction donn´ee.

.bp

2.5.3 *Connexion*

Grammaire abstraite

Connexion-de-canal-`a-acheminement :: *Identificateur-de-canal*

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

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

D'autres constructions sont donn´ees au § 3.

Chaque *identificateur-de-canal* reli´e au bloc englobant doit | tre mentionn´e dans une seule *connexion-de-canal-`a-acheminement* L'*identificateur-de-canal* dans une *connexion-de-canal-`a-acheminement* doit d´enoter un canal connect´e au bloc entre crochets.

Chaque *identificateur-d'acheminement-de-signal* dans une *connexion-de-canal-`a-acheminement* doit | tre d´efini dans le m | me bloc oìu la *connexion-de-canal-`a-acheminement* se trouve d´efinie et doit avoir la fronti`ere de ce bloc situ´ee à l'un de ses points extr | mes. Chaque *identificateur-d'acheminement-de-signal* d´efini dans le bloc qui l'entoure et qui a son environnement comme ´etant l'un de ses points extr | mes, doit | tre mentionn´e dans une seule et unique *connexion-de-canal-`a-acheminement* .

Pour une direction donn´ee, l'union des ensembles d'*identificateur de signal* dans les acheminements de signaux dans une *connexion-de-canal-`a-acheminement* doit | tre ´egale aux signaux achemin´es par l'*identificateur-de-canal* dans la m | me *connexion-de-canal-`a-acheminement* et correspondant à la m | me direction.

Grammaire textuelle concr`ete

<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 concr`ete

Sur le plan graphique, l’´el´ement connexion est repr´esent´e par l’<identificateur de canal > associ´e au trajet de signal et contenu dans la <zone de d´efinition d’acheminement de signal> (voir le § 2.5.2 *grammaire graphique concr`ete*).

2.5.4 *Signal*

Grammaire abstraite

D´efinition-de-signal :: *Nom-de-signal*

Identificateur-de-r´ef´erence-de-sorte |

[*Affinage-de-signal*]

Nom-de-signal :: *Nom*

L’*identificateur-de-r´ef´erence-de-sorte* est d´efini au § 5.2.2.

Grammaire textuelle concr`ete

<d´efinition de signal>::=

SIGNAL { nom de signal > [<liste de sorte>] [<affinage de signal> }

{ <nom de signal > [<liste de sorte>] [<affinage de signal> } <fin>

<liste de sorte>::=

(<sorte> { <sorte })

L’<affinage de signal> est d´efini au § 3.3, la <sorte> au § 5.2.2.

S´emantique

Une instance de signal est un flot d’informations entre des processus; c’est une instanciation d’un type de signal d´efini par une d´efinition de signal. Une instance de signal peut | tre envoy´ee 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 d´epart et de destination, l’<identificateur de signal > sp´ecifi´e dans la sortie correspondante et les autres valeurs, dont les sortes sont d´efinies dans la d´efinition d’un signal.

.bp

2.5.5 *D´efinition de liste de signaux*

Un <identificateur de liste de signaux > peut
| tre utilis´e dans une <d´efinition de canal>, une <d´efinition d'acheminement de signal>, une <d´efinition de liste de signaux>, un <ensemble de signaux d'entr´ee valides> et <liste de mise en r´eserve> comme moyen abr´eg´e pour ´enum´erer les identificateurs de signaux et les signaux de temporisation.

Grammaire textuelle concr`ete

<d´efinition 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 ´etablie en rempla,cant tous les <identificateur de liste de signaux > dans la liste par les <identificateur de signal > qu'ils d´ecrivent, correspond à un *identificateur-de-signal-* **set** dans la grammaire abstraite. Dans chaque <liste de signaux> ainsi ´etablie, chaque <identificateur de signal > doit
| tre distinct.

Grammaire graphique concr`ete

<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)