Union internationale des télécommunications

UIT-T

SECTEUR DE LA NORMALISATION
DES TÉLÉCOMMUNICATIONS
DE L'UIT

Z.104

(10/2004)

SÉRIE Z: LANGAGES ET ASPECTS GÉNÉRAUX LOGICIELS DES SYSTÈMES DE TÉLÉCOMMUNICATION

Techniques de description formelle – Langage de description et de spécification (SDL)

Codage des données SDL

Recommandation UIT-T Z.104

Union internationale des télécommunications

RECOMMANDATIONS UIT-T DE LA SÉRIE Z

LANGAGES ET ASPECTS GÉNÉRAUX LOGICIELS DES SYSTÈMES DE TÉLÉCOMMUNICATION

TECHNIQUES DE DESCRIPTION FORMELLE	
Langage de description et de spécification (SDL)	**Z.100–Z.109**
Application des techniques de description formelle	Z.110–Z.119
Diagrammes des séquences de messages	Z.120–Z.129
Langage étendu de définition d'objets	Z.130–Z.139
Notation de test et de commande de test	Z.140–Z.149
Notation de prescriptions d'utilisateur	Z.150–Z.159
LANGAGES DE PROGRAMMATION	
CHILL: le langage de haut niveau de l'UIT-T	Z.200–Z.209
LANGAGE HOMME-MACHINE	
Principes généraux	Z.300–Z.309
Syntaxe de base et procédures de dialogue	Z.310–Z.319
LHM étendu pour terminaux à écrans de visualisation	Z.320–Z.329
Spécification de l'interface homme-machine	Z.330–Z.349
Interfaces homme-machine orientées données	Z.350–Z.359
Interfaces homme-machine pour la gestion des réseaux de télécommunication	Z.360–Z.379
QUALITÉ	
Qualité des logiciels de télécommunication	Z.400–Z.409
Aspects qualité des Recommandations relatives aux protocoles	Z.450–Z.459
MÉTHODES	
Méthodes de validation et d'essai	Z.500–Z.519
INTERGICIELS	
Environnement de traitement réparti	Z.600–Z.609

Pour plus de détails, voir la Liste des Recommandations de l'UIT-T.

Recommandation UIT-T Z.104

Codage des données SDL

Résumé

Le codage de données SDL permet la communication de valeurs de données entre modules de langage SDL de façon indépendante de l'implémentation.

Les valeurs de données peuvent être codées dans le format textuel défini par la présente Recommandation. En variante, quand les données sont définies en notation ASN.1, des ensembles de règles de codage ASN.1 normalisées peuvent être invoqués.

Les résultats de codage peuvent être utilisés à l'intérieur du modèle SDL ou peuvent servir à communiquer entre modèles SDL, ou entre éléments SDL et autres éléments non SDL tels que les environnements d'essai.

Source

La Recommandation UIT-T Z.104 a été approuvée le 7 octobre 2004 par la Commission d'études 17 (2001-2004) de l'UIT-T selon la procédure définie dans la Recommandation UIT-T A.8.

Mots clés

ASN.1, codage, données, Rec. UIT-T Z.100, Rec. UIT-T Z.105, SDL.

AVANT-PROPOS

L'UIT (Union internationale des télécommunications) est une institution spécialisée des Nations Unies dans le domaine des télécommunications. L'UIT-T (Secteur de la normalisation des télécommunications) est un organe permanent de l'UIT. Il est chargé de l'étude des questions techniques, d'exploitation et de tarification, et émet à ce sujet des Recommandations en vue de la normalisation des télécommunications à l'échelle mondiale.

L'Assemblée mondiale de normalisation des télécommunications (AMNT), qui se réunit tous les quatre ans, détermine les thèmes d'étude à traiter par les Commissions d'études de l'UIT-T, lesquelles élaborent en retour des Recommandations sur ces thèmes.

L'approbation des Recommandations par les Membres de l'UIT-T s'effectue selon la procédure définie dans la Résolution 1 de l'AMNT.

Dans certains secteurs des technologies de l'information qui correspondent à la sphère de compétence de l'UIT-T, les normes nécessaires se préparent en collaboration avec l'ISO et la CEI.

NOTE

Dans la présente Recommandation, l'expression "Administration" est utilisée pour désigner de façon abrégée aussi bien une administration de télécommunications qu'une exploitation reconnue.

Le respect de cette Recommandation se fait à titre volontaire. Cependant, il se peut que la Recommandation contienne certaines dispositions obligatoires (pour assurer, par exemple, l'interopérabilité et l'applicabilité) et considère que la Recommandation est respectée lorsque toutes ces dispositions sont observées. Le futur d'obligation et les autres moyens d'expression de l'obligation comme le verbe "devoir" ainsi que leurs formes négatives servent à énoncer des prescriptions. L'utilisation de ces formes ne signifie pas qu'il est obligatoire de respecter la Recommandation.

DROITS DE PROPRIÉTÉ INTELLECTUELLE

L'UIT attire l'attention sur la possibilité que l'application ou la mise en œuvre de la présente Recommandation puisse donner lieu à l'utilisation d'un droit de propriété intellectuelle. L'UIT ne prend pas position en ce qui concerne l'existence, la validité ou l'applicabilité des droits de propriété intellectuelle, qu'ils soient revendiqués par un Membre de l'UIT ou par une tierce partie étrangère à la procédure d'élaboration des Recommandations.

A la date d'approbation de la présente Recommandation, l'UIT n'avait pas été avisée de l'existence d'une propriété intellectuelle protégée par des brevets à acquérir pour mettre en œuvre la présente Recommandation. Toutefois, comme il ne s'agit peut-être pas de renseignements les plus récents, il est vivement recommandé aux responsables de la mise en œuvre de consulter la base de données des brevets du TSB.

TABLE DES MATIÈRES

Introduction

Le codage permet de spécifier comment les valeurs de données sont codées quand des informations sont communiquées entre modules de langage SDL.

En général, seules des valeurs peuvent être transmises entre modules de langage SDL distincts et entre le modèle SDL et son environnement. C'est pourquoi le codage défini par la présente Recommandation s'applique seulement aux données de la sorte des valeurs: le codage des données de la sorte des objets n'est pas pris en charge. Cette règle est compatible avec l'utilisation de données dans la communication de signaux SDL, où des répliques des items de données de la sorte des objets sont créées (sauf si une instance de processus contient à la fois l'émetteur et le récepteur).

Il est escompté que la spécification de codage pour données SDL sera normalement utilisée soit dans un canal qui représente une interface normative, ou pour des données qui doivent être encapsulées dans un autre item de données afin d'être traitées de façon transparente.

Quand le codage est spécifié dans un canal, celui-ci sera transparent au reste du modèle SDL si les agents récepteurs n'utilisent pas la syntaxe additionnelle qui est indiquée dans la Recommandation pour les entrées. Si aucune des extrémités du canal ne conduit à l'environnement SDL, la spécification de codage consiste uniquement à ajouter les exigences de codage au reste du modèle SDL et le codage est supprimé pendant l'entrée de signal normale. Si l'origine ou la destination du signal est l'environnement SDL, le codage doit imposer une exigence au codage des messages à destination ou en provenance de l'environnement.

L'encapsulation de données dans un autre item de données se produit normalement dans les protocoles stratifiés où la communication entre les deux couches peut correspondre à un canal entre deux agents SDL. Le codage offre la possibilité de simplifier les modèles SDL. La sortie du signal peut être codée dans une couche donnée, mais quand ce signal est reçu à partir du canal, le codage peut être ignoré de façon à recevoir un signal de l'un quelconque des types codés. Les données peuvent être mémorisées dans la couche du récepteur (ou être encapsulées pour d'autres couches) et être transmises à un homologue. Quand les données sont envoyées vers la couche originale, le décodage peut avoir lieu de façon à rétablir le signal original.

Le codage et le décodage sont également fournis comme des expressions de données, de façon que l'encapsulation (et son inverse) n'ait pas à être effectuée dans le cadre de l'entrée et de la sortie.

D'autres utilisations du codage sont envisagées, particulièrement pour le codage textuel. Celui-ci est susceptible d'être utile afin d'effectuer des essais et de valider des modèles SDL, ainsi qu'afin de fournir un codage indépendant de la notation ASN.1, bien que l'ensemble des règles de codage textuel soit conçu pour le transfert d'informations sous forme de caractères textuels vers d'autres automates plutôt que pour leur lecture par des êtres humains.

Etant donné que le codage est essentiellement orienté vers la communication, il est associé aux données acheminées par des ensembles de signaux ainsi qu'à des interfaces, à des portes et à des canaux. Il y a une prise en charge spécifique de l'utilisation d'un nom en syntaxe abstraite ASN.1 (ASN.1 ABSTRACT-SYNTAX) dans une interface afin d'impliquer un ensemble de signaux fondé sur un type CHOICE de niveau supérieur dans la syntaxe abstraite ABSTRACT-SYNTAX.

La présente Recommandation ne fournit pas la spécification du codage des variables.

Recommandation UIT-T Z.104

Codage des données SDL

1 Domaine d'application

La présente Recommandation développe le langage SDL afin de fournir des mécanismes permettant de spécifier le codage de données, de façon que les données transmises entre différentes parties d'un système SDL (ou entre systèmes SDL) possèdent un codage indépendant de l'implémentation.

Un mécanisme est fourni afin de fonder le codage sur différentes règles. Un seul codage est défini lorsque le résultat est une chaîne textuelle. En variante, pour les données qui sont fondées sur des définitions en notation ASN.1 ou qui peuvent y être mappées, les ensembles de règles de codage définis dans la série des Recommandations UIT-T X.69x peuvent être utilisés.

1.1 Utilisation des processus de codage et de décodage

Dans un module de langage SDL implémenté séparément, la façon dont les valeurs de données sont représentées n'a pas besoin d'être connue et l'implémentation garantit que les données se comporteront de la façon prévue.

Quand des informations sont communiquées entre des modules de langage SDL qui sont implémentés séparément, les valeurs de données doivent être transmises au moyen d'un codage qui puisse être traité par tous les modules de langage SDL qui utilisent ces informations. Par exemple, si une valeur booléenne est transmise, elle pourrait l'être comme un simple bit et elle aurait besoin d'être définie si un bit zéro représente une valeur Vraie ou Fausse. Afin que la valeur booléenne soit communiquée correctement, l'émetteur et le récepteur doivent tous les deux appliquer le même ensemble de règles de codage. Avant que la valeur soit envoyée, ou une fois que la valeur a été reçue, il peut y avoir de bonnes raisons pour lesquelles les informations sont représentées d'une façon différente. Par exemple, dans une partie d'un système assurant une bonne prise en charge des entiers de 16 bits, où la capacité de stockage n'est pas importante mais où l'accès à des bits individuels est inefficace, un opérateur booléen pourrait être représenté au mieux par 16 éléments binaires où 16 zéros représenteraient une valeur fausse et où tout autre ensemble de bits représenterait une valeur vraie. Dans une autre partie du même système, un opérateur booléen peut être mieux implémenté sous la forme d'un simple octet.

La communication en langage SDL a lieu via des canaux, qui sont explicites ou implicites. La communication par canaux peut avoir lieu entre parties du système SDL, ou entre parties du système et l'environnement.

La communication qui a lieu dans un canal correspond à un protocole ou peut être considérée comme un protocole. En général, un canal utilisé pour la communication transporte plusieurs signaux différents dans chaque sens. Normalement, les signaux allant dans un sens donné sont différents des signaux allant dans l'autre sens. Parfois, un canal n'achemine des signaux que dans un seul sens, mais c'est inhabituel.

Un canal entre deux agents transporte les signaux afin d'établir un protocole entre ces deux agents. Si l'on suppose qu'un type de signal quelconque peut être reçu à un instant quelconque, chaque type de signal codé allant dans un sens donné doit toujours être distinct de tout autre type de signal allant dans le même sens; sinon l'agent récepteur ne sera pas en mesure de distinguer un signal d'un autre. Deux instances d'un signal ayant les mêmes données à la sortie du même agent dans le même canal ont le même codage. Si le type de signal ou les données ou l'instance originelle d'agent est différent, les informations codées sont différentes. L'agent émetteur est inclus parce que chaque instance de signal achemine l'identificateur pid de son instance originelle d'agent sous la forme d'un paramètre

implicite; sinon, deux instances d'un signal ayant les mêmes données en sortie du même canal ont le même codage.

2 Références normatives

La présente Recommandation se réfère à certaines dispositions des Recommandations UIT-T et textes suivants qui, de ce fait, en sont partie intégrante. Les versions indiquées étaient en vigueur au moment de la publication de la présente Recommandation. Toute Recommandation ou tout texte étant sujet à révision, les utilisateurs de la présente Recommandation sont invités à se reporter, si possible, aux versions les plus récentes des références normatives suivantes. La liste des Recommandations de l'UIT-T en vigueur est régulièrement publiée. La référence à un document figurant dans la présente Recommandation ne donne pas à ce document, en tant que tel, le statut d'une Recommandation.

– Recommandation UIT-T X.680 (2002) | ISO/CEI 8824-1:2002, *Technologies de l'information – Notation de syntaxe abstraite numéro un: spécification de la notation de base*.

– Recommandation UIT-T X.690 (2002) | ISO/CEI 8825-1:2002, *Technologies de l'information – Règles de codage ASN.1: spécification des règles de codage de base, des règles de codage canoniques et des règles de codage distinctives*.

– Recommandation UIT-T X.691 (2002) | ISO/CEI 8825-2:2002, *Technologies de l'information – Règles de codage ASN.1: spécification des règles de codage compact*.

– Recommandation UIT-T X.693 (2001) | ISO/CEI 8825-4:2002, *Technologies de l'information – Règles de codage ASN.1: règles de codage XML*.

– Recommandation UIT-T Z.100 (2002), *SDL: langage de description et de spécification*.

– Recommandation UIT-T Z.105 (2003), *Langage SDL combiné avec des modules ASN.1 (SDL/ASN.1)*.

– ISO/CEI 10646:2003, *Technologies de l'information – Jeu universel de caractères codés sur plusieurs octets (JUC)*.

3 Définitions

La présente Recommandation définit les termes suivants:

3.1 **décoder** (processus): processus de construction d'une valeur de donnée SDL à partir d'une chaîne textuelle ou d'une séquence binaire censée être un codage de la valeur de donnée SDL par l'application des mêmes règles de codage que celles utilisées pour le décodage.

3.2 **décodage**: résultat d'un processus consistant à décoder.

3.3 **coder** (processus): processus visant à produire un codage.

3.4 **codage**: chaîne textuelle ou séquence binaire résultant de l'application d'un ensemble de règles de codage à une valeur de donnée SDL.

3.5 **ensemble de règles de codage**: un des ensembles de règles de codage (ASN.1) définis dans la série des Recommandations UIT-T X.69x, ou l'ensemble de règles de codage textuel défini dans le § 10.7.1 et dans l'Annexe A de la présente Recommandation, ou ensemble de règles de codage dépendant de l'implémentation réalisation ou de l'application, défini par un code de procédure invoqué conformément à la présente Recommandation (voir § 10.7).

4 Abréviations

La présente Recommandation utilise les abréviations suivantes:

ASN.1 notation de syntaxe abstraite numéro un (*abstract syntax notation one*)

SDL langage de description et de spécification (*specification and description language*)

5 Conventions

Les conventions de notation et de présentation de la Rec. UIT-T Z.100 sont utilisées dans la présente Recommandation.

La présente Recommandation est organisée de façon que le numérotage des articles 6 à 13 suive celui de la Rec. UIT-T Z.100.

Lorsqu'une règle de syntaxe abstraite ou concrète est définie dans la présente Recommandation avec le même nom qu'une règle figurant dans la Rec. UIT-T Z.100, la règle indiquée ici remplace celle qui figure dans la Rec. UIT-T Z.100. Toutes les éventuelles conditions de type *Grammaire*, *Sémantique* et *Modèle,* définies d'après une règle nommée et figurant dans la Rec. UIT-T Z.100, s'appliquent à la règle redéfinie, sauf spécification d'une définition contraire dans la présente Recommandation.

Les caractères sont identifiés par les noms (en lettres capitales) qu'ils reçoivent dans l'ISO/CEI 10646.

6 Règles générales

La présente Recommandation ajoute les mots clés additionnels **encode** (coder) et **decode** (décoder) à la règle lexicale <keyword> (mot clé).

6.1 Règle lexicales

<keyword> ::=

abstract	**active**	**adding**
| **aggregation**	| **alternative**	| **and**
| **any**	| **as**	| **association**
| **atleast**	| **block**	| **break**
| **call**	| **channel**	| **choice**
| **comment**	| **composition**	| **connect**
| **connection**	| **constants**	| **continue**
| **create**	| **dcl**	| **decision**
| **decode**	| **default**	| **else**
| **encode**	| **endalternative**	| **endblock**
| **endchannel**	| **endconnection**	| **enddecision**
| **endexceptionhandler**	| **endinterface**	| **endmacro**
| **endmethod**	| **endobject**	| **endoperator**
| **endpackage**	| **endprocedure**	| **endprocess**
| **endselect**	| **endstate**	| **endsubstructure**
| **endsyntype**	| **endsystem**	| **endtype**
| **endvalue**	| **env**	| **exception**
| **exceptionhandler**	| **export**	| **exported**
| **external**	| **fi**	| **finalized**
| **from**	| **gate**	| **handle**
| **if**	| **import**	| **in**
| **inherits**	| **input**	| **interface**
| **join**	| **literals**	| **loop**
| **macro**	| **macrodefinition**	| **macroid**
| **method**	| **methods**	| **mod**
| **nameclass**	| **nextstate**	| **nodelay**
| **none**	| **not**	| **now**
| **object**	| **offspring**	| **onexception**

operator	**operators**	**optional**
or	**ordered**	**out**
output	**package**	**parent**
priority	**private**	**procedure**
protected	**process**	**provided**
public	**raise**	**redefined**
referenced	**rem**	**remote**
reset	**return**	**save**
select	**self**	**sender**
set	**signal**	**signallist**
signalset	**size**	**spelling**
start	**state**	**stop**
struct	**substructure**	**synonym**
syntype	**system**	**task**
then	**this**	**timer**
to	**try**	**type**
use	**value**	**via**
virtual	**with**	**xor**

7 Organisation des spécifications SDL

7.1 Cadre

Le cadre des spécifications SDL est défini dans la Rec. UIT-T Z.100.

7.2 Module logiciel

Modèle

Quand un module ASN.1 est utilisé comme un module logiciel et qu'une sélection de définition <definition selection> contenue dans l'article d'utilisation de module logiciel <package use clause> se rapportant au module ASN.1 possède l'**interface** avec le genre d'entité sélectionnée <selected entity kind> et que le nom <name> est celui d'un type de donnée CHOICE, il y a une interface impliquée. Cette interface impliquée a le même nom que le type CHOICE et définit des signaux équivalents aux options offertes par le type CHOICE.

8 Concepts structuraux

Les concepts structuraux sont définis dans la Rec. UIT-T Z.100 avec l'adjonction de l'identification facultative d'un ensemble de règles de codage pour portes et canaux. Sinon les concepts structuraux sont définis dans la Rec. UIT-T Z.100.

8.1 Types, instances et portes

La présente Recommandation ajoute l'identification facultative d'un ensemble de règles de codage à la définition des portes.

8.1.1 Définitions de type structural

Les définitions de type structural se trouvent dans la Rec. UIT-T Z.100.

8.1.2 Expressions de type

Les expressions de type sont définies dans la Rec. UIT-T Z.100.

8.1.3 Définitions fondées sur des types

Les définitions fondées sur des types sont reproduites dans la Rec. UIT-T Z.100.

8.1.4 Types abstraits

Les types abstraits sont définis dans la Rec. UIT-T Z.100.

8.1.5 Portes

Une porte peut avoir un ensemble de règles de codage. Une sortie de signal issue d'un agent par l'intermédiaire de cette porte est codée comme spécifié par l'ensemble des règles de codage. Les informations reçues par l'intermédiaire de cette porte sont décodées conformément à l'ensemble de règles de codage. Si aucun ensemble spécifique de règles de codage n'est indiqué, le codage n'est pas défini par la spécification SDL.

Grammaire abstraite

Gate-definition :: *Gate-name*
[*Encoding-rules*]
*In-signal-identifier-**set***
*Out-signal-identifier-**set***

S'il y a un ensemble de règles de codage *Encoding-rules*, l'ensemble identificateur de signal d'entrée *In-signal-identifier-**set*** et l'ensemble identificateur de signal de sortie *Out-signal-identifier-**set*** ne doivent pas contenir d'éventuels signaux implicites pour procédures distantes ou variables distantes.

Si un canal externe contenant un ensemble de règles de codage *Encoding-rules* est connecté à la porte d'un agent ou d'un état composite, les règles de codage *Encoding-rules* de la définition *Gate-definition* de cette porte doivent être les mêmes que les règles de codage *Encoding-rules* de ce canal. Il doit y avoir au plus un seul canal de ce type qui soit connecté à la porte et les signaux acheminés dans un sens par la porte doivent être les mêmes que les signaux acheminés dans le sens correspondant du canal.

Si un canal interne d'un agent ou type d'agent est connecté à la porte de l'agent ou type d'agent qui possède une définition de porte *Gate-definition* comportant un ensemble de règles de codage *Encoding-rules*, les règles de codage *Encoding-rules* de ce canal doivent être les mêmes. Les signaux acheminés dans un sens par la porte doivent être les mêmes que les signaux acheminés dans le sens correspondant du canal. Il peut y avoir plusieurs voies internes de ce type.

Grammaire concrète

<gate definition> ::=
{ <gate symbol> | <inherited gate symbol> }
is associated with { <gate> [<encoding rules>]
[<signal list area>] [<signal list area>] } ***set***
[***is connected to*** <endpoint constraint>]

NOTE 1 – Quand une porte contenue dans un sous-type est une extension d'une porte héritée d'un supertype, le symbole de porte héritée <inherited gate symbol> est utilisé dans la syntaxe concrète.

NOTE 2 – Si un codage différent est requis dans chaque sens, la communication doit être spécifiée par deux portes qui transportent chacune des signaux dans un seul sens.

<interface gate definition> ::=
<gate symbol 1>
is associated with { <interface identifier> [<encoding rules>] }

Une spécification de règles de codage <encoding rules> qui est associée à un symbole de porte héritée <inherited gate symbol> doit spécifier le même ensemble de règles de codage *Encoding-rules* que celles de la définition de la porte correspondante, contenue dans le supertype, si cette porte possède un ensemble de règles *Encoding-rules*. S'il n'y a aucun ensemble de règles de codage <encoding rules> qui soit associé à un symbole de porte héritée <inherited gate symbol> et s'il y a un ensemble de règles *Encoding-rules* pour la porte contenue dans le supertype, la porte héritée possède cet ensemble de règles *Encoding-rules*. S'il n'y a aucun ensemble de règles *Encoding-rules* pour la porte contenue dans le supertype, la présence et la valeur de l'ensemble *Encoding-rules* de la porte héritée sont déterminées par la présence et la valeur des règles de codage <encoding rules>.

Sémantique

L'ensemble de règles de codage *Encoding-rules* d'une définition de porte *Gate-definition* d'un type d'agent ou d'un type d'état composite correspond à l'ensemble des règles *Encoding-rules* du canal situé dans le domaine de visibilité englobant qui est indiqué par les (l'ensemble des) spécifications d'instance. Le codage utilisé dans le comportement du type peut être déterminé indépendamment du canal réel qui est connecté à la porte de l'instance.

Modèle

L'ensemble des règles de codage *Encoding-rules* d'une définition de porte *Gate-definition* du type d'agent impliqué d'un diagramme d'agent <agent diagram> (ou le type d'état composite impliqué d'une zone <composite state area>) est le même que l'ensemble des règles de codage *Encoding-rules* du canal externe (explicites ou impliquées) à partir desquelles la définition de porte *Gate-definition* est déduite. Il n'y a pas de règles de codage *Encoding-rules* dans cette définition de porte *Gate-definition* s'il n'y en a aucune dans le canal externe.

L'ensemble des règles *Encoding-rules* d'une définition de porte *Gate-definition* impliquée par une définition de type d'agent *Agent-type-definition* du système (que ce type soit défini par un type de système <system type> ou impliqué à partir d'un diagramme de système <system diagram>) est le même que l'ensemble des règles de codage *Encoding-rules* du canal interne (explicites ou impliquées) à partir desquelles la définition de porte *Gate-definition* est déduite. Il n'y a aucune règle de codage *Encoding-rules* dans cette définition de porte *Gate-definition* s'il n'y en a aucune dans le canal interne.

Si un diagramme explicite de porte <gate on diagram> est indiqué pour le diagramme de type de système <system type diagram> ou pour le diagramme de système <system diagram> et si les règles de codage <encoding rules> correspondantes sont présentes, l'ensemble des règles *Encoding-rules* de la définition de porte *Gate-definition* correspondant à la définition du type d'agent *agent-type-definition* est déterminé par les règles de codage <encoding rules> de ce diagramme de porte <gate on diagram>. Si les règles de codage <encoding rules> sont absentes, l'ensemble des règles *Encoding-rules* est déterminé à partir du canal interne de la même façon que pour une définition de porte *Gate-definition* impliquée.

8.2 Paramètres de contexte

Les paramètres de contexte (y compris les paramètres de contexte de porte) sont définis dans la Rec. UIT-T Z.100.

NOTE – Quand une porte contenue dans un type paramétré est définie par un paramètre de contexte formel, l'ensemble des règles de codage de la porte contenue dans un type spécialisé, qui est utilisé afin de définir des instances, sera déterminé par la définition de porte réelle, identifiée par le paramètre de contexte réel.

8.3 Spécialisation

La spécialisation est définie dans la Rec. UIT-T Z.100.

8.4 Références de type

Les références de type sont définies dans la Rec. UIT-T Z.100.

NOTE – Une zone de propriété de porte <gate property area> contenue dans une référence de type est une définition de porte <gate definition> ou une définition de porte d'interface <interface gate definition> et peut donc contenir une spécification de règles de codage <encoding rules> qui doit toujours être compatible avec la définition indiquée avec le type.

8.5 Associations

Les associations sont définies dans la Rec. UIT-T Z.100.

9 Agents

Les agents sont définis dans la Rec. UIT-T Z.100.

10 Communication

Le codage des données développe la définition des canaux et des connexions.

La grammaire des règles de codage est définie.

10.1 Canal

Un canal connectant deux agents détermine le codage (éventuel) à utiliser pour une communication entre les agents. Un canal qui est connecté à l'environnement d'un agent possède le codage défini (éventuellement) pour la porte qui le connecte avec l'environnement.

Grammaire abstraite

Channel-definition :: *Channel-name*
[*Encoding-rules*]
[**NODELAY**]
*Channel-path-**set***

La porte d'origine *Originating-gate* ou de destination *Destination-gate* doit avoir les mêmes règles de codage *Encoding-rules* que la définition de canal *Channel-definition*. Si la définition de canal *Channel-definition* ne possède aucune règle de codage *Encoding-rules*, ni la porte d'origine *Originating-gate* ni la porte de destination *Destination-gate* ne doit avoir de règles de codage *Encoding-rules*.

Grammaire concrète

<channel definition area> ::=
<channel symbol>
is associated with
{ [<channel name> [<encoding rules>]]
{ [<signal list area>] [<signal list area>] } ***set*** }
is connected to {
{ <agent area> | <state partition area> | <gate on diagram> }
{ <agent area> | <state partition area> | <gate on diagram> } } ***set***

NOTE 1 – Si un codage différent est requis dans chaque sens, la communication doit être spécifiée par deux voies qui transportent chacune des signaux dans un seul sens.

Sémantique

Les règles de codage *Encoding-rules* d'une définition de canal *Channel-definition* connectée à un ensemble de spécifications d'instance sont utilisées dans le comportement des instances.

Modèle

Si les règles de codage <encoding rules> sont omises et que le symbole de canal <channel symbol> soit connecté à un diagramme de porte <gate on diagram> avec un ensemble de règles de codage <encoding rules>, la définition de canal *Channel-definition* a les mêmes règles de codage *Encoding-rules* que la définition de porte *Gate-definition* du diagramme de porte <gate on diagram>. Si la définition de porte *Gate-definition* ne possède pas de règles de codage *Encoding-rules,* la définition de canal *Channel-definition* ne possède pas de règles de codage *Encoding-rules*.

Les canaux implicites n'ont pas de règles de codage *Encoding-rules* si les portes auxquelles ils sont connectés n'ont pas de règles de codage *Encoding-rules*. Sinon les règles de codage *Encoding-rules* d'un canal implicite sont les mêmes que les règles de codage *Encoding-rules* de chacune des portes.

NOTE 2 – Si le canal est connecté au cadre du diagramme englobant et qu'il n'y ait aucun diagramme de porte <gate on diagram>, cet état représente une connexion.

10.2 Connexion

Modèle

Les règles de codage *Encoding-rules* d'une porte implicite de connexion sont les mêmes que pour le canal externe connecté. Un canal sans règles de codage <encoding rules> (ou un canal implicite) qui est connecté à une porte possédant un ensemble de règles de codage *Encoding-rules* a les mêmes règles de codage *Encoding-rules*.

10.3 Signal

Les signaux sont définis dans la Rec. UIT-T Z.100.

10.4 Définition de la liste des signaux

La liste des signaux est définie dans la Rec. UIT-T Z.100.

10.5 Procédures distantes

Les procédures distantes sont définies dans la Rec. UIT-T Z.100.

10.6 Variables distantes

Les variables distantes sont définies dans la Rec. UIT-T Z.100.

10.7 Règles de codage du trajet de communication (coder et décoder)

L'ensemble des règles de codage spécifie les règles utilisées afin de coder et de décoder les données acheminées par un canal particulier ou par une porte particulière.

Grammaire abstraite

Encoding-rules	::	*Rules-identifier*
Encoding-expression	::	*Signal-identifier* [*Expression*]* *Encoding-path*
Encoding-path	::	{ *Channel-identifier* | *Gate-identifier* }
Decoding-expression	::	*Expression* *Encoding-path*
Rule-identifier	::	*Literal-identifier*

L'identificateur de règle *Rule-identifier* doit être un des identificateurs de littéral appartenant au type de donnée `Encoding` (voir ci-dessous dans *Sémantique*). Si la règle réelle qui a été identifiée correspond à un codage défini dans la série des Recommandations UIT-T X.69x, l'ensemble des signaux transportés par un trajet de codage *Encoding-path* doit correspondre aux éléments d'un type ASN.1 CHOICE qui est transporté par le canal. Le trajet de codage *Encoding-path* et le type ASN.1 CHOICE correspondent dans un même sens si chaque nom de signal correspond à un nom CHOICE et si, pour chaque signal, le (seul) paramètre du signal est le même que le type de donnée de la définition CHOICE correspondante.

Le type de donnée `Encoding` doit être le type prédéfini de donnée `Encoding` ou un type de donnée portant le nom `Encoding` c'est-à-dire une spécialisation (directe ou indirecte) du type prédéfini de donnée `Encoding`. La spécialisation doit seulement ajouter des noms de littéral au type prédéfini de donnée `Encoding` et ne doit pas changer d'autres propriétés.

Si l'identificateur de règle *Rule-identifier* correspond (par le nom *Name*) à un littéral `Encoding` défini dans le **`module logiciel`** `Predefined`, des procédures incorporées et impliquées par les ensembles normalisés de règles de codage sont invoquées (et d'autres procédures – même si elles sont visibles avec des noms correspondants comme ci-dessous – sont ignorées).

Si l'identificateur de règle *Rule-identifier* correspond à un littéral supplémentaire qui est ajouté à une spécialisation du type prédéfini de donnée `Encoding`, il doit y avoir une procédure visible avec la signature appropriée pour chaque invocation (implicite ou explicite) de l'expression de codage *Encoding-expression* ou de décodage *Decoding-expression*.

Le nom de la procédure visant à coder est le nom `encode` concaténé avec le nom d'un littéral additionnel `Encoding` (par exemple, `encodemyprotocol` où le littéral additionnel `Encoding` est `myprotocol`). Cette procédure doit avoir un seul paramètre du type CHOICE implicite concernant le trajet applicable à l'invocation (voir ci-dessous dans *Sémantique*). La procédure doit renvoyer une chaîne de caractères, de bits ou d'octets (charstring, Bitstring ou Octetstring).

Le nom de la procédure visant à décoder est le nom `decode` concaténé avec le nom d'un littéral additionnel `Encoding` (par exemple, `decodemyprotocol` où le littéral additionnel `Encoding` est `myprotocol`). Cette procédure doit avoir un seul paramètre du même type (chaîne de caractères, de bits ou d'octets) que la procédure de codage correspondante et doit renvoyer le type CHOICE concernant le trajet applicable à l'invocation (voir ci-dessous dans *Sémantique*).

Chaque procédure visant à coder ou à décoder doit être fonctionnelle (c'est-à-dire qu'elle ne doit pas contenir d'états et ne doit pas changer la valeur d'une quelconque variable SDL externe à la procédure quand elle est interprétée).

La longueur de la liste des *Expressions* facultatives doit être la même que le nombre d'identificateurs de référence de sorte *Sort-reference-identifier* contenus dans la définition de signal *Signal-definition* indiquée par l'identificateur de signal *Signal-identifier*.

Chaque *Expression* contenue dans une expression de codage *Encoding-expression* doit être d'une sorte compatible avec la référence d'identificateur de sorte (par position) *Sort-identifier-reference* correspondante, contenue dans la définition de signal *Signal-definition* indiquée par l'identificateur de signal *Signal-identifier*.

Pour un identificateur de canal *Channel-identifier* sur le trajet de codage *Encoding-path* d'une expression de codage *Encoding-expression* d'agent, le canal de ce trajet doit être accessible au moyen de l'identificateur de signal *Signal-identifier* extrait de l'agent et le trajet de canal *Channel-path* dans le sens d'éloignement de l'agent doit impérativement comprendre l'identificateur de signal *Signal-identifier* dans son ensemble d'identificateurs de signal *Signal-identifier*.

Pour un identificateur de porte *Gate-identifier* du trajet de codage *Encoding-path* d'une expression de codage *Encoding-expression* d'agent, la porte doit être une porte de l'agent ou doit être accessible au moyen d'un canal contenant l'identificateur de signal *Signal-identifier* issu de l'agent et l'ensemble d'identificateurs de signal en sortie *Out-signal-identifier-**set*** de la porte doit comprendre l'identificateur de signal *Signal-identifier*.

L'ex*pression* contenue dans une expression de décodage *Decoding-expression* doit être compatible avec la sorte produite par une expression de codage *Encoding-expression* au moyen du même trajet de codage *Encoding-path* dans un contexte où le contexte de l'expression de décodage *Decoding-expression* est accessible au moyen du trajet de codage *Encoding-path*.

Grammaire concrète

<encoding rules> ::=
encode <rules identifier>

<rules identifier> ::=
<literal>

<encoding expression> ::=
encode { <signal identifier>[(<actual parameters>)] | <expression> }
[<encoding path>]

<encoding path> ::=
as { <channel identifier> | <gate identifier> }

<decoding expression> ::=
decode <expression> [<encoding path>]

L'ensemble des règles de codage du trajet identifié par l'identificateur de canal <channel identifier> ou par l'identificateur de porte <gate identifier> est utilisé.

Si une expression de codage <encoding expression> contient une <expression> (plutôt qu'un signal), le signal effectif est déduit de l'<expression> comme décrit dans le modèle ci-dessous. La sorte de l'<expression> doit être celle du type impliqué CHOICE de données correspondant à l'ensemble de signaux pour le trajet de codage <encoding path>.

Le trajet de codage <encoding path> ne doit être omis à partir de l'expression de codage <encoding expression> que s'il y a exactement un seul trajet avec codage pour sortie du signal dans le contexte de l'expression de codage <encoding expression> et dans ce cas le codage pour ce trajet est utilisé.

Le trajet de codage <encoding path> ne doit être omis à partir de l'expression de codage <encoding expression> que s'il y a exactement un seul trajet avec codage pour entrée dans le contexte de l'expression de décodage et dans ce cas le codage pour ce trajet est utilisé.

Un trajet de codage <encoding path> utilisé comme une sorte <sort> indique les données implicites qui sont définies pour le trajet défini ci-dessous.

Sémantique

L'item de règles de codage *Encoding-rules* détermine l'ensemble des règles utilisé afin de modifier le codage qui dépend de la réalisation pour les données internes:

– soit vers un codage normalisé et indépendant de l'implémentation (pour un seul des littéraux de type de donnée `Encoding` afin de coder de `text` à `EXER`);
– ou vers un codage défini par l'implémentation ou l'application (pour un littéral ajouté à une spécialisation du type de donnée `Encoding`).

Le processus de codage est invoqué chaque fois qu'un signal est émis via un trajet avec un ensemble de règles de codage spécifié et que la procédure de codage correspondante est appelée. Quand un signal est injecté à partir d'un tel trajet, le processus de décodage des données dans le codage interne est invoqué par appel de la procédure de décodage correspondante. Ces processus de codage et de décodage n'ont donc aucun impact sur la sémantique de langage SDL définie dans la Rec. UIT-T Z.100, mais nécessitent un codage spécifique des signaux pour les trajets spécifiés et permet donc d'implémenter différentes parties du système séparément.

Quand une expression de codage *Encoding-expression* ou de décodage *Decoding-expression* est interprétée, la procédure appropriée est appelée.

Le processus de codage relatif à l'ensemble des règles de codage sur un trajet est invoqué dans une expression de codage *Encoding-expression* afin de produire une chaîne de caractères, de bits ou d'octets selon le contexte et l'ensemble des règles de codage utilisé. Cette chaîne de caractères, de bits ou d'octets produite par le processus de codage est décodée par une expression de décodage *Decoding-expression* au moyen du même ensemble de règles de codage sur le même trajet. On

utilise alors l'ensemble de règles de codage du trajet de codage identifié par l'identificateur de canal *Channel-identifier* ou de porte *Gate-identifier* ou d'interface *Interface-identifier*.

Pour une expression de codage *Encoding-expression,* les données sont codées comme si elles allaient être émises sur ce trajet. Le résultat est un type de donnée correspondant à l'ensemble de règles de codage pour le trajet.

Pour une expression de décodage *Decoding-expression*, les données sont décodées comme si elles avaient été reçues sur le trajet spécifié. Le résultat est une expression correspondant au type implicite de donnée relatif aux entrées à partir de ce canal dans le contexte de processus de décodage défini ci-dessous. Si le décodage échoue, l'exception de référence non valide est déclenchée.

Sur un trajet qui contient un codage, un type de donnée qui correspond au langage SDL est implicitement défini:

```
value type Implicitname /* un nom implicite et unique */
{ choice
         signal1 value
         { struct
            1 Sort11 optional;
            2 Sort12 optional;
            3 Sort13 optional;
            /* ... et ainsi de suite pour chaque paramètre de signal1 */
          } ;
         signal2 value
         { struct
            1 Sort21 optional;
            2 Sort22 optional;
            3 Sort23 optional;
            /* ... et ainsi de suite pour chaque paramètre de signal2 */
          } ;
         signal3 NULL;  /* aucun paramètre */
         /* ... et ainsi de suite pour chaque signal */
}
```

où

`signal1`, `signal2`, etc., sont les noms des signaux transportés par le trajet, et où

`Sort11`, `Sort12`, etc., indiquent le type de données correspondant aux paramètres de `signal1`, et où

`Sort21`, `Sort22`, etc., indiquent le type de données correspondant aux paramètres de `signal2`.

Pour un signal qui ne possède aucun paramètre, le type de donnée prédéfini a la valeur `NULL`.

Si le trajet est bidirectionnel et qu'un signal soit transporté dans les deux sens, il n'y a qu'un seul choix dans le type implicite de donnée pour ce signal.

L'identificateur implicite de ce type de donnée est indiqué par l'expression **as** <channel identifier> ou <gate identifier> pour le trajet, de sorte qu'une déclaration de variable légale est:

```
dcl message as user_input;
```

où `user_input` est le nom d'un canal ou d'une porte avec codage et une affectation valide est:

```
message := decode encoded_value as user_input;
```

Le type suivant de données énumérées pour l'ensemble normalisé de règles de codage doit être ajouté au module logiciel "Predefined" qui est défini au § D.3/Z.100 afin de prendre en charge le codage de données SDL:

```
value type Encoding
     { literals text, BER, CER, DER, APER, UPER, CAPER, CUPER, BXER, CXER, EXER }
```

qui sert à indiquer l'ensemble requis de règles de codage comme suit:

`text` pour l'ensemble des règles de codage textuel qui est défini dans la présente Recommandation et qui produit une chaîne de caractères;

`BER` pour l'ensemble des règles de codage de base de la notation ASN.1 qui produit une chaîne d'octets;

`CER` pour l'ensemble des règles de codage canoniques de la notation ASN.1 qui produit une chaîne d'octets;

`DER` pour l'ensemble des règles de codage distinctives de la notation ASN.1 qui produit une chaîne d'octets;

`APER` pour la variante de base alignée des règles de codage compact de la notation ASN.1 qui produit une chaîne d'octets;

`UPER` pour la variante de base non alignée des règles de codage compact de la notation ASN.1 qui produit une chaîne de bits;

`CAPER` pour la variante canonique alignée des règles de codage compact de la notation ASN.1 qui produit une chaîne d'octets;

`CUPER` pour la variante canonique non alignée des règles de codage compact de la notation ASN.1 qui produit une chaîne de bits;

`BXER` pour la variante de base des règles de codage en langage XML de la notation ASN.1 qui produit une chaîne de caractères;

`CXER` pour la variante canonique des règles de codage en langage XML de la notation ASN.1 qui produit une chaîne de caractères;

`EXER` pour la variante étendue des règles de codage en langage XML de la notation ASN.1 qui produit une chaîne de caractères.

Le synonyme "PER" est ajouté au module logiciel **package** "Predefined" comme remplacement du terme APER comme suit:

```
synonym PER Encoding = APER;
```

L'opérateur `last` du type de donnée `Encoding` est redéfini comme un nom inconnu et ne peut donc pas faire l'objet d'un accès, de sorte qu'une application ou implémentation est en mesure d'étendre le type de donnée `Encoding` sans aucun changement lorsque seules les règles ci-dessus sont utilisées. Le type de donnée `Encoding` peut être spécialisé par adjonction de littéraux additionnels.

Modèle

Si une expression de codage <encoding expression> contient une <expression>, le choix présent est déterminé à partir de l'expression et le signal ayant le même nom que le choix est émis avec les valeurs des paramètres de signal indiquées par l'expression.

10.7.1 Ensemble de règles de codage textuel

L'ensemble de règles de codage textuel est fourni de façon que les informations puissent être acheminées sur les trajets de communication au moyen de chaînes textuelles entre éléments contenus dans le système et entre ce système et l'environnement. Le codage des caractères n'est pas défini. Bien que les chaînes textuelles soient en général lisibles par les êtres humains, ce n'est pas l'objet des règles de codage textuel.

Sémantique

Si l'ensemble des règles de codage est spécifié sous forme de texte `text`, le résultat d'un codage est une chaîne de caractères `Charstring`.

La chaîne effective est déterminée comme suit:

> L'ACCOLADE GAUCHE et L'ACCOLADE DROITE { } délimitent les valeurs de type de données et montrent où les valeurs commencent et s'arrêtent, sauf lorsqu'elles apparaissent dans le codage d'une chaîne de caractères `Charstring`, auquel cas elles représentent les caractères effectifs d'accolade gauche <left curly bracket> ou d'accolade droite <right curly bracket> de la Rec. UIT-T Z.100.
>
> Les caractères de VIRGULE servent à délimiter des éléments dans une liste (par exemple dans un codage de création **struct**).
>
> Sinon la chaîne effective de caractères est déterminée pour chaque type de donnée défini ci-dessous et est illustrée comme la chaîne de caractères produite ('Generated `CharString`') dans les exemples.

Un signal complet est codé comme une liste de valeurs et est traité comme un codage de choix (**choice**) du type impliqué de donnée pour le trajet avec codage.

Les détails de codage textuel sont indiqués dans l'Annexe A.

10.7.2 Ensembles de règles de codage normalisés dans la série des Rec. UIT-T X.69x

L'utilisation d'un seul des noms `BER`, `CER`, `DER`, `APER`, `UPER`, `CAPER`, `CUPER`, `BXER`, `CXER` ou `EXER` (correspondant à un ensemble de règles de codage de la série des Recommandations UIT-T X.69x – voir § 10.7 *Sémantique* ci-dessus) ne doit être spécifiée que si les signaux transportés par le trajet sont définis comme un type de donnée ASN.1 CHOICE. Les valeurs sont donc codées conformément à ce type de donnée, traité comme une syntaxe abstraite (ABSTRACT-SYNTAX) en notation ASN.1.

11 Comportement

11.1 Départ

Le départ est défini dans la Rec. UIT-T Z.100.

11.2 Etat

Si, pendant l'analyse des signaux à l'accès d'entrée, le signal analysé ne correspond à aucun des signaux valides dans l'un quelconque des états de l'agent (par exemple parce que le message reçu ne peut pas être décodé en signal valide), l'exception de référence invalide est déclenchée.

11.3 Entrée

Si une entrée codée <encoded input> est indiquée pour un trajet (canal ou porte), les messages qui peuvent être reçus à partir de ce trajet sont reçus et mémorisés dans la variable indiquée, dans leur forme codée. Ces signaux ne peuvent pas être spécifiés dans une quelconque autre entrée ou mémoire pour le même état. Si les mêmes signaux peuvent être reçus à partir d'un autre trajet, ils restent assignés à la variable, qui doit être d'une sorte compatible avec le codage pour le trajet. Bien que le modèle indiqué ci-dessous décrive les signaux comme étant d'abord décodés puis recodés, il est escompté que les réalisations réelles optimiseront cette opération afin de copier la valeur codée vers la variable indiquée dans l'entrée codée <encoded input>.

Grammaire concrète

<input list> ::=
<stimulus> { , <stimulus> }*
| <asterisk input list>
| <encoded input>

<encoded input> ::=
encode <variable> [<encoding path>]

Le trajet de codage <encoding path> ne doit être omis qu'à partir de l'expression <encoded input> s'il y a exactement un seul trajet avec codage conduisant au contexte de l'entrée qui possède un ensemble de règles de codage produisant la sorte (chaîne de caractères, d'octets ou de bits) de la <variable>.

Sémantique

Si le signal spécifié dans l'entrée est reçu via un canal qui possède un ensemble de règles de codage spécifié, ce signal est décodé conformément à cet ensemble de règles de codage.

Modèle

Pour chacun des signaux qui peut être reçu à partir du canal ou de la porte spécifié(e) dans une entrée codée <encoded input>, il y a une entrée impliquée qui est équivalente à l'attribution des valeurs acheminées par signal à des variables locales implicites de types appropriés, suivies par une tâche impliquée. Cette tâche impliquée attribue à la <variable> la valeur d'une expression de codage pour le signal et pour les valeurs reçues à partir des variables implicites au moyen de l'ensemble des règles de codage concernant le trajet de l'entrée codée <encoded input>. Après la tâche impliquée, la transition qui fait suite à l'entrée codée est interprétée.

11.4 Entrée de priorité

Sémantique

Si le signal spécifié dans l'entrée de priorité est reçu via un canal qui possède un ensemble de règles de codage spécifié, ce signal est décodé conformément à cet ensemble de règles de codage.

11.5 Signal continu

Le signal continu est défini dans la Rec. UIT-T Z.100.

11.6 Condition de validation

Sémantique

Si le signal spécifié est reçu via un canal qui possède un ensemble de règles de codage spécifié, ce signal est décodé conformément à cet ensemble de règles de codage.

11.7 Sauvegarde

La sauvegarde est définie dans la Rec. UIT-T Z.100.

11.8 Transition implicite

La transition implicite est définie dans la Rec. UIT-T Z.100.

11.9 Transition spontanée

La transition spontanée est définie dans la Rec. UIT-T Z.100.

11.10 Etiquette

L'étiquette est définie dans la Rec. UIT-T Z.100.

11.11 Automate à états et l'état composite

L'automate à états et l'état composite sont définis dans la Rec. UIT-T Z.100.

11.12 Transition

La transition est définie dans la Rec. UIT-T Z.100.

11.13 Action

11.13.1 Tâche

La tâche est définie dans la Rec. UIT-T Z.100.

11.13.2 Création

La création est définie dans la Rec. UIT-T Z.100.

11.13.3 Appel de procédure

L'appel de procédure est défini dans la Rec. UIT-T Z.100.

11.13.4 Sortie

Si une sortie d'expression <expression output> est indiquée pour un trajet (canal ou porte), cette expression est une valeur de choix servant à construire le signal à envoyer. La sorte de choix correspond à l'ensemble des signaux concernant le trajet.

Si une sortie codée <encoded output> est indiquée pour un trajet (canal ou porte), l'expression indiquée est utilisée comme signal à envoyer. Dans le modèle indiqué ci-dessous, l'expression est décodée afin de vérifier le signal qui est ainsi envoyé.

Grammaire concrète

```
<output body> ::=
                        <output body item> {, <output body item> }*
                        <communication constraints>

<output body item> ::=
                        <signal identifier> [<actual parameters>]
                        <expression output>
                    |   <encoded output>

<expression output> ::=
                        <expression>

<encoded output> ::=
                        encode <expression>
```

Quand une sortie d'expression <expression output> est utilisée, il doit y avoir exactement un seul trajet intermédiaire <via path> dans les contraintes de communication <communication constraints> et la sorte de l'expression <expression> de la sortie d'expression <expression output> doit être la sorte du type impliqué CHOICE de donnée correspondant à l'ensemble de signaux pour ce trajet.

Quand une sortie codée <encoded output> est utilisée, il doit y avoir exactement un seul trajet intermédiaire <via path> dans les contraintes de communication <communication constraints> et ce trajet intermédiaire <via path> doit spécifier une porte ou un canal qui possède un ensemble de règles de codage qui correspond à la sorte (chaîne de caractères, d'octets ou de bits) de l'expression <expression> de la sortie de codage <encoding output>.

Sémantique

Si un signal est émis via un trajet qui possède un ensemble de règles de codage spécifié, ce signal est codé conformément à cet ensemble de règles de codage.

Modèle

Si l'item du corps de sortie <output body item> est une sortie d'expression <expression output>, la présentation du choix est déterminée à partir de l'expression et le signal portant le même nom que ce choix est émis avec les valeurs paramétriques de signal indiquées par cette expression.

Si l'item du corps de sortie <output body item> est une sortie codée <encoded output>, le signal qui est émis est celui qui est obtenu à partir du décodage du contenu de l'expression conformément à la règle de décodage pour la réception des signaux envoyés sur le trajet spécifié. Le signal choisi est celui qui porte le même nom que la présentation du choix dans le décodage. Si le signal n'est pas valide pour le trajet ou si le décodage échoue pour une raison quelconque (par exemple si la chaîne de l'expression ne correspond pas validement à un seul des signaux pour le trajet) l'exception "OutOfRange" est déclenchée et aucun signal n'est envoyé; sinon, la valeur du décodage est émise en tant que signal. Etant donné que l'expression est déjà codée comme une chaîne pour le trajet, la valeur de chaîne peut être envoyée sans autre conversion.

11.13.5 Décision

La décision est définie dans la Rec. UIT-T Z.100.

11.14 Liste d'instructions

La liste d'instructions est définie dans la Rec. UIT-T Z.100.

11.15 Temporisateur

Le temporisateur est défini dans la Rec. UIT-T Z.100.

11.16 Exception

L'exception est définie dans la Rec. UIT-T Z.100.

12 Données

12.1 Définitions de données

Les définitions des données sont reproduites dans la Rec. UIT-T Z.100 sauf que la sorte <sort> est étendue par le trajet de codage <encoding path>.

Grammaire concrète

```
<sort> ::=
                <basic sort> [ ( <range condition> ) ]
        |       <anchored sort>
        |       <expanded sort>
        |       <reference sort>
        |       <pid sort>
        |       <inline data type definition>
        |       <inline syntype definition>
        |       <encoding path>
```

12.2 Utilisation passive de données

La syntaxe abstraite des expressions est étendue de façon à inclure le codage et le décodage, sinon l'utilisation passive des données est définie dans la Rec. UIT-T Z.100.

12.2.1 Expressions

Le codage et le décodage sont ajoutés sous forme d'expressions.

Grammaire abstraite

Active-expression = *Variable-access*
| *Conditional-expression*
| *Operation-application*
| *Equality-expression*
| *Imperative-expression*
| *Range-check-expression*
| *Value-returning-call-node*
| *State-expression*
| *Encoding-expression*
| *Decoding-expression*

Grammaire concrète

<expression0> ::=
<operand>
| <create expression>
| <value returning procedure call>
| <encoding expression>
| <encoding expression>

12.3 Utilisation active de données

La syntaxe abstraite des expressions est étendue de façon à inclure le codage et le décodage; sinon, l'utilisation active de données est définie dans la Rec. UIT-T Z.100.

13 Définition de système générique

La définition de système générique est reproduite dans la Rec. UIT-T Z.100.

Annexe A

Spécification de l'ensemble de règles de codage textuel

Les types de données sont présentés ci-dessous, ceux de l'Annexe D/Z.100 étant placés dans l'ordre de leur apparition et étant suivis par d'autres types de données.

A.1 Booléens

Les valeurs "Faux" (False) et "Vrai" (True) des opérateurs booléens (`Boolean`) doivent être codées respectivement par la LETTRE LATINE MAJUSCULE F et par la LETTRE LATINE MAJUSCULE T.

Exemple

```
dcl Var_Boolean Boolean;
task Var_Boolean := true;
```

Generated `CharString`: T

A.2 Caractères

Les valeurs des caractères (`Character`) doivent être codées comme le caractère réel à l'exception du caractère `ESC`, qui est codé comme deux caractères d'échappement ESCAPE. Une valeur de caractère indéfinie ou absente doit être codée comme un caractère ESCAPE suivi par le caractère NULL.

NOTE – La chaîne de caractères (`CharString`) produite peut contenir des caractères non imprimants.

Exemple

```
dcl Var_Character Character;

task Var_Character := 'M';
```

Generated `CharString`: M

A.3 Chaîne

Une chaîne (`String`) possède un paramètre pour la sorte d'item et doit être codée comme une liste de valeurs de la sorte d'item, englobées entre une ACCOLADE GAUCHE et une ACCOLADE DROITE, les valeurs étant séparées par des caractères de VIRGULE.

Exemple

```
value type IntString inherits String <Integer>;
dcl str IntString;

task str:= ''//mkstring(6)//mkstring(9)//mkstring(1948);
```

Generated `CharString`: {6,9,1948}

A.4 Types de chaîne "Charstring, IA5String, NumericString, PrintableString, VisibleString"

Bien que le type `Charstring` soit fondé sur le type `String` parce qu'il s'agit d'un type prédéfini de donnée qui est couramment utilisé, ce type reçoit un codage spécial. Les types `IA5String` et `Charstring` recouvrent le même jeu de caractères et ont le même codage. Les types `NumericString`, `PrintableString` et `VisibleString` sont des sous-ensembles du type `IA5String`.

Ces types de chaîne doivent être codés comme une chaîne de caractères englobée entre des caractères d'APOSTROPHE (') sans changement sauf que l'<apostrophe> (') doit être codée comme deux caractères APOSTROPHE (').

NOTE 1 – Dans une chaîne de caractères de virgule <comma>, les accolades gauche <left curly brackets> et droite <right curly bracket> sont traitées comme des caractères normaux.

NOTE 2 – La chaîne de caractères (`CharString`) produite peut donc contenir des caractères non imprimants y compris des caractères de fin de fichier ou de fin d'enregistrement.

Exemple

```
dcl Var_Charstring Charstring;

task Var_Charstring := 'Fred''s world;
```

Generated `CharString`: 'Fred''s world'

A.5 Entiers

Les entiers (`Integer`) doivent être codés en notation décimale d'entiers sans caractères initiaux de CHIFFRE ZERO, les nombres négatifs étant immédiatement précédés par un caractère de TRAIT D'UNION-MOINS sans caractères initiaux de CHIFFRE ZERO. Le chiffre zéro ne doit jamais être traité comme un nombre négatif.

Exemple

```
dcl i Integer;

task i := 2-7;
```

Generated `CharString`: -5

A.6 Nombres naturels

Un nombre naturel (`Natural`) est un **syntype** du type `Integer` qui doit donc avoir le même codage que le type `Integer`.

A.7 Nombres réels

La valeur 0.0 doit être codée comme un CHIFFRE ZERO suivi par un POINT suivi par un CHIFFRE ZERO.

Toute valeur négative doit être codée comme un caractère de TRAIT D'UNION-MOINS immédiatement suivi par le codage de la valeur rendue négative (valeur positive d'un nombre réel).

Une valeur positive d'un nombre réel doit être codée comme un simple chiffre dans l'étendue de 1 à 9, suivi par un POINT, suivi par au moins un et jusqu'à 11 chiffres de fraction décimale, suivi(s) par la LETTRE LATINE MINUSCULE 'e', suivie par un exposant. L'exposant est le codage par un entier de la valeur de cet exposant: la puissance en base 10 à appliquer à la première partie du nombre. L'exposant peut être négatif.

Les derniers chiffres zéro après la virgule décimale peuvent être omis. Certaines valeurs de réel ne peuvent pas être précisément codées et il est possible que des valeurs inégales d'un nombre réel, présentant une très petite différence, aient le même codage. En tout cas, la question de savoir si une valeur de réel est précisément correcte dans une application dépendra de la façon dont ce réel a été implémenté. Par exemple, si la valeur de réel 2,0/7,0 est effectivement mémorisée comme le rapport de deux entiers (2 et 7), cette valeur est absolument précise, tandis qu'un codage plus conventionnel pourrait être: 0,2857142857, qui n'est correct qu'avec 10 chiffres après la virgule.

Exemples

```
dcl p, q, r1, r2 Real := 2000.0, 7.0, 0, 0;

task r1:= p/q;
task r2:= q/p;
```

Generated `CharString` for r1: 2.85714285714e2

Generated `CharString` for r2: 3.5e-3

A.8 Tableau

Le type `Array` a deux paramètres: l'un de la sorte des indices, l'autre de la sorte des composants. Ces deux sortes peuvent en principe être d'un type quelconque mais, habituellement, la sorte des indices possède un nombre fini de valeurs.

Lorsque la sorte des indices possède un nombre fini de valeurs ordonnées, un type `Array` doit être codé comme une liste de valeurs de codage d'élément, une pour chaque élément, séparées par des caractères de VIRGULE dans une simple paire de caractères d'ACCOLADE GAUCHE et d'ACCOLADE DROITE.

Exemple

```
value type ABC {literals A, B, C};
value type A1 inherits Array <ABC, Integer>;
dcl avalue, bvalue A1;

task avalue:= (. 3 .);
task bvalue[A]:= 3;
task bvalue[B]:= 5;
task bvalue[C]:= 7;
```

Generated `CharString` for avalue: {3,3,3}

Generated `CharString` for bvalue: {3,5,7}

Il est possible que l'indice ne soit pas ordonné (c'est-à-dire que l'opérateur "<" ne soit pas défini, ou par exemple que la sorte des indices soit de type **structure** ou **choice**). Il est également possible que la sorte des indices ne possède pas un nombre fini de valeurs (c'est-à-dire qu'aucun nombre possible de valeurs ne puisse être infini, par exemple `CharString` ou `Real`). Dans le cas d'une sorte d'indice non ordonné ou infini, le type `Array` doit être codé comme la valeur la plus fréquente, suivie par des paires de valeurs pour chaque élément. Si au moins deux valeurs apparaissent avec la fréquence la plus élevée, une seule de celles-ci est choisie, arbitrairement, comme étant la valeur la plus fréquente. Chaque paire doit être placée entre une ACCOLADE GAUCHE et une ACCOLADE DROITE et être séparée par une VIRGULE: la première valeur est une valeur d'indice et la seconde valeur est une valeur d'élément. Aucune valeur d'indice ne doit être répétée dans les paires.

Exemple

```
value type Dehashing inherits Array <CharString, CharString> := (.''.);
/* note that the default value is an empty string */
dcl hashtable Dehashing;

task htable ('ac'):= 'action';
task htable ('ab'):= 'ability';
task htable ('zzzz'):= 'end of document';
```

Generated `CharString` for `htable`: {'',{'ab','ability'},{'ac','action'},{'zzzz','end of document'}}

A.9 Vecteur

Un vecteur est un cas particulier d'un tableau qui possède toujours une sorte d'indice qui est un sous-ensemble des nombres naturels avec une étendue allant de 1 jusqu'à une valeur maximale spécifiée et toute sorte d'item. Un vecteur doit donc être codé de la même façon qu'un tableau avec un nombre fini de valeurs ordonnées d'indice: c'est-à-dire une liste de valeurs de codage d'élément, une pour chaque élément, séparées par des caractères de VIRGULE et insérées entre une simple paire d'ACCOLADE GAUCHE et d'ACCOLADE DROITE.

A.10 Mode ensembliste

Un type `Powerset` d'une sorte représente un ensemble mathématique dont les éléments sont tous membres de cette sorte. Lorsque la sorte possède un nombre fini de valeurs ordonnées, l'ensemble `Powerset` de la sorte doit être codé comme une chaîne d'éléments binaires (soit une chaîne de caractères de CHIFFRE ZERO et de CHIFFRE 1) englobée entre des caractères d'APOSTROPHE ('). Chaque position binaire dans cette chaîne d'éléments binaires indique si une valeur de la sorte est présente ou absente dans l'ensemble. Un CHIFFRE 1 indique que la valeur est présente et un CHIFFRE ZERO indique que la valeur est absente. Le bit de gauche représente la plus petite valeur de la sorte de l'élément et chaque autre bit représente une valeur de la sorte de l'élément plus grande que les valeurs des bits situés à gauche.

Exemple

```
value type Shortalpha {literals a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t,u,v};
/* note Shortalpha has 22 values */
value type Psa inherits Powerset<Shortalpha>;
dcl letters_used Psa;

task letters_used := (. h, c, u, r .);
```

Generated `CharString`: '0010000100000000010010'

Lorsque la sorte ne possède pas un nombre fini de valeurs ordonnées, le type `Powerset` de la sorte doit être codé comme chacune des valeurs d'élément présentes, séparées par des caractères de VIRGULE et englobées dans une paire d'ACCOLADE GAUCHE et d'ACCOLADE DROITE. Les valeurs d'élément peuvent être placées dans un ordre quelconque.

Exemple

```
value type Pchrstr inherits Powerset<Charstring>;
dcl strings_used Pchrstr;

task strings_used := (. 'me', 'you', 'us', 'me', 'again', 'hey', 'you' .);
```

Generated CharString: {'again','hey','you','me','us'}

A.11 Durée

Le type Duration sert à indiquer 'un intervalle temporel'. Sauf spécification contraire, la valeur par défaut de l'unité de durée (Duration) est 1 s.

Les valeurs de durée doivent être codées comme une paire d'entiers séparés par une VIRGULE et englobés dans une paire d'ACCOLADE GAUCHE et d'ACCOLADE DROITE: le premier entier indique le nombre d'unités et le second entier indique toute partie fractionnaire en milliardièmes (10^{-9}). Par défaut, ces unités sont des secondes et des nanosecondes. La valeur peut être négative, auquel cas le codage de la valeur rendue négative doit être utilisé avec un caractère TRAIT D'UNION-MOINS inséré immédiatement avant le premier entier.

Exemple

```
dcl dvar Duration;

task dvar := -17.00000007;
```

Generated CharString: {-17,700}

A.12 Temps

Le type Time sert à indiquer 'un point dans le temps'. La valeur d'une unité de Time doit être la même que la valeur de l'unité de durée (Duration). L'origine des unités de Time n'est pas spécifiée par la présente Recommandation mais correspond à l'horloge système mise à zéro: c'est-à-dire que la commande "NOW" donne la valeur "0.0". Il est permis que les valeurs de temps soient négatives.

Les valeurs de temps doivent être codées de la même façon que les valeurs de durée.

Exemple

```
dcl tvar Time;

task tvar:= 17.0000017;
```

Generated CharString: {17,17000}

A.13 Sac

Le type Bag doit être codé de la même façon qu'un type Powerset avec une sorte d'élément qui ne possède pas d'ensemble fini de valeurs ordonnées, mais chaque valeur d'élément étant précédée par un entier suivi par un POINT-VIRGULE. L'entier indique le nombre de fois où la sorte d'élément apparaît dans la valeur de type Bag.

Exemple

```
value type B1 inherits Bag <Integer>;

dcl Var_Bag B1;

task Var_Bag := (. 7, 4, 7 .);
```

Generated CharString: {2:7,1:4}

A.14 Bit, chaîne de bits

Le type `Bit` doit être codé comme un simple CHIFFRE ZERO ou un simple CHIFFRE 1 représentant respectivement un bit 0 ou un bit 1.

Exemple

```
dcl Var_Bit Bit;

task Var_Bit := 1;
```

Generated `CharString`: 1

Le type `Bitstring` doit être codé comme une séquence de bits représentée par les caractères de CHIFFRE ZERO et de CHIFFRE 1, englobés entre des caractères d'APOSTROPHE (').

Exemple

```
dcl Var_Bit Bit_String;

task Var_Bit := '01011'B;
```

Generated `CharString`: '01011'

A.15 Octet, chaîne d'octets

Le type `Octet` doit être codé comme deux caractères correspondant à la notation hexadécimale du type `Octet`. Les caractères alphabétiques doivent être représentés par des lettres minuscules (c'est-à-dire la LETTRE LATINE MINUSCULE A jusqu'à la LETTRE LATINE MINUSCULE F).

Exemple

```
dcl oct Octet;

task oct := 62;
```

Generated `CharString`: 3e

Le type `Octetstring` est une séquence de valeurs de type `Octet` et doit être codé comme une chaîne de paires de caractères hexadécimaux englobées entre des caractères d'APOSTROPHE ('). Les caractères alphabétiques doivent être représentés par des lettres minuscules.

Exemple

```
dcl os Octetstring;

task os:= '12B32D'H;
```

Generated `CharString`: '12b32d'

A.16 Identificateurs de processus Pid et sortes d'identificateur pid

Afin d'assurer la flexibilité entre applications, les valeurs d'identificateur pid doivent être codées comme une valeur de type CHOICE correspondant à la définition du choix:

```
{ choice
      0 ApplicationDefined;
      1 Integer;
      2 OctetString;
      3 BitString;
      4 CharString;
      5 { struct
           identity CharString;
           instance Natural;
        };
}
```

Une valeur de la sorte des identificateurs pid doit être codée comme la valeur correspondante de la sorte des identificateurs Pid.

Toutes les valeurs d'identificateur pid doivent être codées au moyen du même champ de choix que défini ci-dessus: c'est-à-dire que si une seule valeur d'identificateur pid dans un modèle SDL est codée au moyen du champ 1du choix (Entier), toutes les autres valeurs d'identificateur pid situées dans le même modèle doivent être codées au moyen du champ 1 du choix: entier. Sinon, la sélection entre les champs de choix ci-dessus dépend de l'application. Le premier champ de choix permet d'utiliser un codage défini par l'application. La sorte du champ `ApplicationDefined` n'est pas définie par la présente Recommandation.

La même instance d'agent d'un modèle doit toujours avoir le même codage de valeur pid. Deux instances d'agent différentes doivent toujours avoir des valeurs codées d'identificateur de processus pid différentes. Sinon, la construction des valeurs codées d'identificateur pid n'est pas définie plus en détail par la présente Recommandation.

Exemple (avec utilisation du champ de choix 5)

```
dcl agent_id Pid; /* in second instance of process IPS */

task agent_id := self;
```

Generated `CharString`: {5, {IPS, 2}}

A.17 Null

La valeur Null (voir la Rec. UIT-T Z.105) doit être codée comme un caractère de CHIFFRE ZERO.

Exemple

```
dcl Var_Null Null;

task Var_Null := Null;
```

Generated `CharString`: 0

A.18 Enumératif (liste de littéraux)

Un type énumératif défini par une liste de littéraux ou comme un type ASN.1 ENUMERATIF doit être codé comme la valeur de nombre naturel qui est indiquée par l'application de l'opérateur `num` du type de donnée au littéral représentant la valeur à coder.

Exemple

```
value type Enum
{ literals e1, e2, e3;
};

dcl evar Enum;

task evar := e2;
```

Generated `CharString`: 1

A.19 Structures

Une valeur de type de donnée concernant une structure doit être codée comme une liste de valeurs des champs contenus dans une paire d'ACCOLADE GAUCHE et d'ACCOLADE DROITE, séparées par des caractères de VIRGULE.

Exemple

```
value type Record { struct
f1 Integer;
f2 Charstring;
f3 Integer;};
dcl locat Record;
task locat:= (. 17, 'mid-field', 230125 .);
```

Generated `CharString`: {1, 'mid-field', 230125}

A.20 Choix

Une valeur de choix doit être codée comme une paire de valeurs séparées par une VIRGULE et englobées dans une paire d'ACCOLADE GAUCHE et d'ACCOLADE DROITE. La première valeur est le champ choisi, indiqué par le nom du choix. La seconde valeur est la chaîne de caractères (`CharString`) pour la valeur de champ conformément au type du champ.

Exemple

```
value type C {choice
cs CharString;
cb Boolean;
};

dcl ChoiceVar C;

task ChoiceVar.cb := true;
```

Generated `CharString`: {cb,T}

A.21 Héritage et syntype

Un **syntype** définit un nouveau nom pour un type de donnée avec une contrainte facultative sur les valeurs de l'ensemble. Le codage textuel est donc identique à celui du type de donnée.

Un type de donnée qui hérite d'un autre type de donnée a le même codage que le type de donnée parent; mais si des littéraux ou des champs supplémentaires sont ajoutés, ils le sont aussi au codage.

SÉRIES DES RECOMMANDATIONS UIT-T

Série A	Organisation du travail de l'UIT-T
Série D	Principes généraux de tarification
Série E	Exploitation générale du réseau, service téléphonique, exploitation des services et facteurs humains
Série F	Services de télécommunication non téléphoniques
Série G	Systèmes et supports de transmission, systèmes et réseaux numériques
Série H	Systèmes audiovisuels et multimédias
Série I	Réseau numérique à intégration de services
Série J	Réseaux câblés et transmission des signaux radiophoniques, télévisuels et autres signaux multimédias
Série K	Protection contre les perturbations
Série L	Construction, installation et protection des câbles et autres éléments des installations extérieures
Série M	Gestion des télécommunications y compris le RGT et maintenance des réseaux
Série N	Maintenance: circuits internationaux de transmission radiophonique et télévisuelle
Série O	Spécifications des appareils de mesure
Série P	Qualité de transmission téléphonique, installations téléphoniques et réseaux locaux
Série Q	Commutation et signalisation
Série R	Transmission télégraphique
Série S	Equipements terminaux de télégraphie
Série T	Terminaux des services télématiques
Série U	Commutation télégraphique
Série V	Communications de données sur le réseau téléphonique
Série X	Réseaux de données, communication entre systèmes ouverts et sécurité
Série Y	Infrastructure mondiale de l'information, protocole Internet et réseaux de prochaine génération
Série Z	**Langages et aspects généraux logiciels des systèmes de télécommunication**

Imprimé en Suisse
Genève, 2005