UNION INTERNATIONALE DES TÉLÉCOMMUNICATIONS

UIT-T Z.109

(11/99)

SECTEUR DE LA NORMALISATION
DES TÉLÉCOMMUNICATIONS
DE L'UIT

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

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

Combinaison du langage SDL avec le langage de modélisation unifié (SDL/UML)

Recommandation UIT-T Z.109

(Antérieurement Recommandation du CCITT)

RECOMMANDATIONS UIT-T DE LA SÉRIE Z

LANGAGES ET ASPECTS INFORMATIQUES GÉNÉRAUX 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
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.399
QUALITÉ DES LOGICIELS DE TÉLÉCOMMUNICATION	Z.400–Z.499
MÉTHODES DE VALIDATION ET D'ESSAI	Z.500–Z.599

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

RECOMMANDATION UIT-T Z.109

COMBINAISON DU LANGAGE SDL AVEC LE LANGAGE DE MODÉLISATION UNIFIÉ (SDL/UML)

Résumé

Objectif

La présente Recommandation définit le sous-ensemble spécialisé du langage UML qui a un mappage direct dans le langage SDL et qui peut être utilisé en combinaison avec ce langage.

Portée

La présente Recommandation présente une définition du mappage du langage UML dans le langage SDL à utiliser pour la combinaison de ces langages.

Application

Le principal domaine d'application de la présente Recommandation est la spécification de systèmes de télécommunication. La combinaison des langages SDL et UML conduit à un outil cohérent permettant de spécifier la structure et le comportement de systèmes de télécommunication, ainsi que les données.

Etat/stabilité

La présente Recommandation constitue le manuel de référence complet décrivant le mappage du langage UML dans le langage SDL à utiliser pour la combinaison de ces langages.

Travaux associés

Recommandation Z.100: SDL.

Source

La Recommandation UIT-T Z.109, élaborée par la Commission d'études 10 (1997-2000) de l'UIT-T, a été approuvée le 19 novembre 1999 selon la procédure définie dans la Résolution n° 1 de la CMNT.

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.

La Conférence mondiale de normalisation des télécommunications (CMNT), qui se réunit tous les quatre ans, détermine les thèmes d'études 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 n° 1 de la CMNT.

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.

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

Recommandation Z.109

COMBINAISON DU LANGAGE SDL AVEC LE LANGAGE DE MODÉLISATION UNIFIÉ (SDL/UML)

(Genève, 1999)

1 Introduction

La *combinaison du langage de description et de spécification (SDL, specification and description language) avec le langage de modélisation unifié (UML, unified modelling language)* est définie par la présente Recommandation ainsi que par la Recommandation Z.100. La présente Recommandation définit un *profil SDL UML* fondé sur le langage UML [1] et sur le langage SDL-2000 [2]. Les parties appropriées de la grammaire graphique du langage SDL et les autres éléments de notation du langage SDL sont définis dans la référence [2].

La présente Recommandation ne traite pas de la combinaison du langage UML et des diagrammes MSC [3].

1.1 Objectif

L'objectif est de tirer parti de la base formelle du langage SDL et de l'expressivité du langage UML, notamment d'utiliser des diagrammes de classe UML avec des associations. L'utilisation du sous-ensemble spécialisé du langage UML défini dans la présente Recommandation permet d'exprimer en langage UML des parties d'une spécification SDL.

1.2 Principes de la combinaison SDL-UML

La présente Recommandation définit la *spécialisation et la restriction du langage UML en vue de le combiner avec le langage SDL* (un *profil SDL UML*). Elle garantit un mappage bien défini entre des parties d'un modèle UML et un modèle SDL. Pour chacun des ModelElements (éléments de modèle) UML qui sont inclus dans le *SDL UML*, il existe un mappage *biunivoque* sur les concepts SDL correspondants. Le mappage est fondé sur le méta-modèle UML et sur la grammaire abstraite du SDL. Un outil qui implémente le *SDL UML* doit absolument prendre en charge ces spécialisations et restrictions et être capable d'assurer ce mappage biunivoque.

Les spécialisations et les restrictions concernant l'UML sont définies sur la base du méta-modèle UML et de la grammaire abstraite du SDL, c'est-à-dire de manière indépendante de la notation.

La présente Recommandation ne donne pas d'indications relatives à la notation en *SDL UML*. Pour certains des éléments UML, le langage SDL contient des éléments dont la notation graphique est semblable à celle du langage UML. Un outil permettant de combiner les langages UML et SDL peut utiliser de facto une norme de notation graphique UML pour le langage UML dont traite la présente Recommandation, mais la partie de cet outil se rapportant expressément au langage SDL doit fournir la grammaire graphique nécessaire pour les éléments définis dans la Recommandation Z.100.

1.3 Restrictions concernant le SDL et l'UML

Il n'y a aucune restriction concernant le SDL. Toutefois, le SDL n'est pas intégralement couvert par le *SDL UML*.

Une restriction générale concernant le *SDL UML* est que seuls les éléments de modèle définis dans le *profil SDL UML* garantissent un mappage biunivoque. Dans une utilisation combinée de l'UML et du

SDL, d'autres parties de l'UML peuvent être utilisées, mais on ne peut pas garantir que le mappage de celles-ci fonctionne de la même façon avec des outils différents.

1.4 Mappage SDL UML

Pour définir le mappage SDL UML, on utilise les mécanismes d'extension UML: Stereotypes (stéréotypes), TaggedValues (valeurs étiquetées) et Constraints (contraintes), on restreint l'utilisation de l'UML et on affecte des significations plus spécifiques à l'UML.

Les *classes* UML représentent généralement des *types* d'entité du SDL. La plupart du temps, le *kind* (genre) d'entité est représenté par des *stereotypes* (stéréotypes). Les éléments de modèle, stéréotypes ou mots clés prédéfinis de l'UML qui ont une signification similaire à la signification qu'ils ont en SDL ont été utilisés.

1.5 Règles de bonne formation

Les deux règles suivantes sont des règles générales de bonne formation concernant le *SDL UML*:

- seuls les éléments de modèle ayant des mappages en SDL (et définis dans la présente Recommandation) peuvent être utilisés;
- le modèle *SDL UML* doit être conforme à la sémantique (statique) du SDL.

Une conséquence de ces règles est que, par exemple, le *ownedElement* (élément détenu) d'une classe qui représente un type SDL doit être une classe *SDL UML* qui représente un type (pouvant être) défini dans l'unité de portée du type SDL.

Pour chacun des éléments de modèle du *SDL UML*, des règles spécifiques de bonne formation sont décrites.

1.6 Conventions

La définition du langage *SDL UML* suit la même organisation que la définition de la sémantique de l'UML. Pour chaque élément de modèle UML, un tableau décrit le mappage en SDL, puis d'éventuelles restrictions et spécialisations sont indiquées. Chaque tableau suit la hiérarchie de généralisation du méta-modèle et comporte une entrée pour chaque élément de modèle, attribut et association qui est considéré ou qui a un mappage. Les éléments de la grammaire abstraite du SDL figurent en italiques avec des traits d'union, par exemple: *Variable-definition*.

Notation

non applicable	non utilisé en *SDL UML*
pas de concept	aucun concept correspondant en SDL
défaut	mappage par défaut
obligatoire	mappage obligatoire
^	super-classes de l'élément de modèle par rapport à un mappage spécial
.	attribut de l'élément de modèle
=	valeur possible de l'attribut
->	association de l'élément de modèle

1.7 Structure de la présente Recommandation

La présente Recommandation est organisée selon le méta-modèle UML, ceci en raison du propos de la Recommandation qui est de savoir "quelles parties du langage UML peuvent être utilisées en combinaison avec le SDL et comment s'effectue alors le mappage sur le SDL". Le mappage entre l'UML et le SDL aurait tout aussi bien pu s'effectuer avec le SDL comme point de départ.

En cas de différences entre la présente Recommandation et la Recommandation Z.100 [2] concernant la description du SDL, la définition donnée dans la Recommandation Z.100 [2] s'applique.

2 Références

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.

[1] Langage de modélisation unifié (UML) (spécification du groupe de gestion d'objets (OMG) sur le langage UML, version 1.3: document OMG ad/99-06-08).

[2] Recommandation UIT-T Z.100 (1999), *Langage de description et de spécification*.

[3] Recommandation UIT-T Z.120 (1999), *Diagrammes de séquences de messages*.

3 Eléments de modèle *SDL UML*

La structure principale du *profil SDL UML* suit le méta-modèle de l'UML. La correspondance avec le SDL est définie par le mappage du méta-modèle UML dans la grammaire du SDL.

3.1 Récapitulatif des éléments de modèle *SDL UML*

Le langage *SDL UML* donne des spécialisations et des restrictions concernant les éléments de modèle suivants:

- gestion de modèle:
  - Package
  - Model
- Classes de stéréotypes:
  - «system»
  - «block»
  - «process»
  - «procedure»
  - «interface»
  - «object»
  - «value»
  - «state»
- «signal» Classifier
- Associations:
  - composition constituant une représentation partielle de contenance entre agents
  - association avec stéréotype représentant une porte avec contrainte de point d'extrémité
  - autres associations représentant les associations correspondantes en SDL
- Generalization

– Dependencies:
  - «import»
  - «create»
– State machine

3.1.1 Stereotypes (stéréotypes)

Les éléments normalisés de l'UML suivants sont utilisés:

Nom	S'applique à	Description
model	Package::Model	spécification SDL (provenant de Model Management)
interface	Classifier::Interface	interface
signal	Classifier	définition de signaux
create	Dependency::Usage	création
import	Dependency::Permission	clause de référence de paquetage

La présente Recommandation définit les stéréotypes suivants:

Nom	S'applique à	Description
system	Class	système (est également un stéréotype de Model en UML, mais ici Class est utilisé)
block	Class	bloc
process	Class	processus
procedure	Class	procédure
object	Class	type d'objet
value	Class	type de valeur
state	Class	type d'état composite
gate	Association	porte avec contrainte de point d'extrémité
signature	Operation	paramètres formels d'agents

3.1.2 TaggedValues (valeurs étiquetées)

La présente Recommandation définit les valeurs étiquetées suivantes:

Nom	S'applique à	Description
encloser	Class	spécifie le type englobant (unité de portée). La valeur est l'identificateur de type.
• virtual • redefined • finalized	Class	spécifie la virtualité <virtuality> d'un type, d'une procédure ou d'un opérateur. • **virtual** • **redefined** • **finalized**
remote	• Attribute • Procedure	• variable distante • procédure distante les valeurs sont booléennes.

3.1.3 Constraints (contraintes)

Nom	S'applique à	Description
atleast	Class	représente une contrainte de virtualité sur un type virtuel
atleast	Parameter	représente une contrainte de virtualité sur un paramètre de contexte

3.2 Eléments de modèle du Core (noyau)

Les restrictions, spécialisations et mappages qui s'appliquent aux éléments généraux du Core du méta-modèle UML sont donnés dans le présent sous-paragraphe. Ils s'appliquent en l'absence de toute spécification pour des éléments de modèle plus spécifiques.

3.2.1 Abstraction

Non applicable.

3.2.2 Association

Dans cette version de la présente Recommandation, des associations sont utilisées pour représenter les concepts SDL suivants:

- agrégation composite correspondant à la contenance, c'est-à-dire des ensembles d'entités contenus dans d'autres entités;
- connexions éventuelles d'ensembles d'agents (par des canaux connectant des portes à des interfaces) dans le cadre de la structure interne d'un agent;
- porte avec contrainte de point d'extrémité par une association dont le stéréotype est "gate".

Des associations autres que celles-ci sont valides (et décrites ci-dessous), mais sont considérées comme des commentaires et sont mappées aux associations correspondantes en SDL.

Les associations *n'*englobent *pas* les ensembles de types connectés par des canaux:

1) les associations sont utilisées pour définir les propriétés de types, tandis que les canaux (dans les diagrammes de structure SDL) sont utilisés pour définir les liens entre ensembles d'instances. Différents canaux peuvent connecter différents ensembles fondés sur le même type. Par conséquent, la connexion d'ensembles d'agents au moyen de canaux est représentée par des collaborations en UML;
2) comme principe de base pour l'utilisation de collaborations, tous les types d'agents ayant des interfaces et des portes pouvant être connectées par des canaux auront, dans le mappage SDL UML, des associations implicites. Ces associations ne supposent pas de propriétés de classes associées.

3.2.2.1 Agrégation composite

Une agrégation composite sert à représenter le fait que les instances d'un type d'agent contiennent des ensembles d'agents fondés d'un autre type. Par exemple, les instances d'un type système contiennent des ensembles de blocs. Les ensembles sont représentés par les points d'extrémité de l'agrégation composite, les noms de rôle étant les noms des ensembles et la multiplicité étant le nombre d'instances.

La signification de l'agrégation composite en UML n'est pas exactement la même que celle de la contenance en SDL. La présente Recommandation impose la signification plus restreinte de la contenance SDL à la composition UML. Un élément d'un ensemble d'agents contenu *ne peut pas* changer d'ensemble, comme c'est le cas en UML. Un ensemble d'agents contenu et ses membres ne peuvent pas exister sans le conteneur, de la même façon qu'en UML.

Il est à noter que l'agrégation composite implique uniquement des propriétés pour le type à l'extrémité composite. Le type à l'autre extrémité n'est pas affecté.

3.2.2.2 Porte avec contrainte de point d'extrémité

Une association de stéréotype gate (porte) représente le fait que le type d'agent correspondant à la classe d'extrémité source a une porte avec une contrainte de point d'extrémité correspondant à la classe d'extrémité cible. Le nom de l'association est le nom de la porte. Seuls les interfaces, les signaux et les procédures distantes associés au sens sortant de la porte sont représentés, les listes de signaux ne le sont pas.

3.2.2.3 Associations générales

Bien que les associations générales en *SDL UML* soient des commentaires en SDL, cette version de la présente Recommandation ne prend en charge qu'un sous-ensemble d'associations UML.

UML	SDL
Association	• contenance: ensemble d'agents faisant partie d'autres entités • porte avec contrainte de point d'extrémité • associations générales: associations en tant que commentaires, mais restreintes par la *non*-prise en charge des éléments d'association suivants: – associations n-aires (n>2) – extrémités d'association qualifiées – classe d'association – agrégation – possibilité de changement – portée de cible
->connection	
^GeneralizableElement	non applicable
^ModelElement	
.name	• contenance: non applicable • porte avec contrainte de point d'extrémité: nom de la porte (*Gate-name*) • autrement: nom <association name>

3.2.3 AssociationClass (classe d'association)

Non applicable.

3.2.4 AssociationEnd (extrémité d'association)

Dans les descriptions suivantes, lorsqu'on parle d'une extrémité d'association dans le cas d'une association binaire, l'extrémité *source* est l'autre extrémité; l'extrémité *target* (cible) est celle dont les propriétés sont examinées.

Une extrémité d'association peut représenter une spécification partielle de deux concepts SDL:

– Contenance:
  - agents contenant d'autres agents: l'association doit être une composition et les classes concernées doivent représenter des types d'agent;
  - états composites contenant des états fondés sur le même type: l'association doit être une composition et les classes concernées doivent représenter des types d'état.

– Porte ave contrainte de point d'extrémité: il faut alors utiliser l'association de stéréotype «gate».

Une extrémité d'association peut aussi constituer simplement l'extrémité d'une association ordinaire. Elle représentera alors simplement l'extrémité d'association de l'association SDL correspondante. Cela signifie qu'une composition qui n'est pas englobée par les deux cas susmentionnés représente simplement l'association de composition correspondante en SDL.

UML	SDL
AssociationEnd	• contenance • porte avec contrainte de point d'extrémité dans les entités à l'extrémité source • autrement: extrémité <association end>
.aggregation = composite	• contenance, à savoir que l'extrémité représente un type d'agent (système/bloc/processus) qui contient des ensembles de blocs et de processus ou que l'extrémité représente un état composite contenant des états fondés sur le même type • autrement: type <association kind>= – type <composition not bound kind>, ou – type <composition part end bound kind>, ou – type <composition composite end bound kind>, ou – type <composition two ends bound kind>
= aggregate	• type <association kind>= – type <aggregation not bound kind>, ou – type <aggregation part end bound kind>, ou – type <aggregation aggregate end bound kind>, ou – type <aggregation two ends bound kind>
= none	• si l'extrémité source de l'association est telle que aggregation = composite: l'extrémité cible représente un ensemble d'agents ou un état composite fondé sur le même type faisant partie d'un type d'état composite. • en cas d'association dont le stéréotype est "gate": porte avec contrainte de point d'extrémité.
.changeability = none = frozen = addOnly	non applicable
.ordering = unordered = ordered	non applicable pour la contenance et pour la porte avec contrainte de point d'extrémité, autrement: • *défaut* • **ordering**
.isNavigable = true = false	non applicable pour la contenance et pour la porte avec contrainte de point d'extrémité, autrement: • navigable • non applicable

UML	SDL
.multiplicity	• pour la contenance: représente le nombre d'instances (*Number-of-instances*) (expressions sous forme d'intervalles) • pour la porte avec contrainte de point d'extrémité: 1 • autrement: multiplicité <multiplicity> de l'extrémité <association end>
.targetScope = instance = classifier	 • entité • non applicable
.visibility = public = protected: default = private	• pour la contenance: non applicable • pour la porte avec contrainte de point d'extrémité: non applicable • autrement: – **exported** – non applicable – local
->qualifier	non applicable
->specification	• pour la contenance: non applicable • pour la porte avec contrainte de point d'extrémité: interfaces, signaux et procédures distantes associés au sens sortant de la porte • autrement: spécificateur <specifier>
->type	• pour la contenance: type d'un ensemble d'agents fondés sur le même type • pour la porte avec contrainte de point d'extrémité: le type de contrainte de point d'extrémité • autrement: type <linked type>
^ModelElement	
.name	• pour la contenance: nom *Agent-name* dans la définition *Agent-definition* • pour la porte avec contrainte de point d'extrémité: non applicable • autrement: nom <role name>

Le *type* ne peut désigner que des éléments de modèle qui sont des classificateurs *SDL UML*, exception faite des interfaces.

3.2.5 Attribute (attribut)

Les attributs représentent des variables d'agents et de procédures, ou des champs de valeurs ou d'objets de données, ou des paramètres de signaux.

UML	SDL
Attribute	variable, champ ou paramètre de signal
.changeability = changeable = frozen = addOnly	 • obligatoire • non applicable • non applicable
.initialValue	initialisation <default initialization> [:= <ground expression>] dans les variables <variables of sort> dans la définition *Variable-definition*
.multiplicity	Note – Peut être utilisé pour les types ASN.1.
.targetScope	

UML	SDL
->type	identificateur *sort-reference-identifier* dans la définition *Variable-definition*, dans le cadre de la définition *Signal-definition* ou de la définition de champ
->associationEnd	non applicable
^Feature	
.ownerScope = instance = classifier	 • obligatoire • non applicable
.visibility = public = protected = private	 • variable **exported** (exportée) ou champ public • variables locales d'agents et champ protégé • non applicable aux variables d'agents, mais applicable aux champs privés
-> owner	l'entité définissant la variable ou le champ
^ModelElement	
.name	nom *Variable-name*/<field name>

Le *type* d'un attribut est restreint: il ne peut être qu'une classe définie par la présente Recommandation.

Les règles suivantes s'appliquent:

- *type* = classe de stéréotype «block» ou «process»: Pid fondé sur le type
- *type* = classe de stéréotype «object»: variable de référence d'objet
- *type* = classe de stéréotype «value»: variable de type de valeur

Les types prédéfinis sont représentés par les classes prédéfinies correspondantes de stéréotype «process», «object» ou «value».

3.2.6 BehaviouralFeature (caractéristique comportementale)

UML	SDL
BehaviouralFeature	opérateur ou procédure
.isquery = true = false	 • *Static-operator/Dynamic-operator* (opérateur) • *Static-operator/Dynamic-operator* (méthode) ou *Procedure-definition*
->Parameter	• *Argument-list* dans *Signature* • *Procedure-formal-parameter**

3.2.7 Binding (rattachement)

Un *rattachement* (binding) est une sous-classe de Dependency (dépendance) qui représente la fourniture de paramètres de contexte effectifs en SDL. Le rattachement en tant que relation explicite n'existe pas en SDL, mais peut être décrit de cette façon en UML.

Pour un *rattachement*, les arguments représentent des éléments de modèle figurant dans la portée (espace de noms) du client qui sont des paramètres effectifs pour les paramètres de gabarit de la source. Les paramètres effectifs doivent avoir un mappage sur les paramètres de contexte valides pour le type SDL qui est représenté par la source.

Les gabarits partiellement instanciés correspondent à des types paramétrés pour lesquels les paramètres de contexte ne sont pas tous fournis.

UML	SDL
Binding	expression <type expression> avec paramètres <actual context parameters>
->argument	paramètre <actual context parameter>
^Dependency	
->client	le type défini au moyen de l'expression <type expression>
->supplier	le type <base type> (dans l'expression <type expression>)
^ModelElement	
.name	non applicable

Le *client* et le *fournisseur* doivent être des classes de même stéréotype.

3.2.8 Class (classe)

Les classes représentent des types et des procédures SDL. Les différents genres de types d'entité en SDL sont représentés par des stéréotypes.

Un type en SDL est complètement défini par une définition ou par un diagramme et éventuellement par une référence (au cas où la définition ou le diagramme fait l'objet d'une référence). Une classe définit des parties du type *et* éventuellement une référence de type. Les parties qu'elle définit dépendent du contenu de ses compartiments et de sa composition. Le choix des propriétés à définir en *SDL UML* est laissé au spécificateur, mais les propriétés spécifiées devront être compatibles avec les propriétés correspondantes définies dans le diagramme ou dans la définition du type SDL.

Les attributs et les associations d'une classe sont restreints/spécialisés et mappés en SDL comme décrit dans le tableau ci-après. Pour chacun des différents genres d'entité SDL, les tableaux qui suivent du 3.2.8.1 au 3.2.8.4 donnent plus de détails. Les attributs définis dans Foundation Package:Auxiliary Elements (éléments auxiliaires) sont également inclus ici.

UML	SDL
Class	types et procédures
.isActive = true = false	 • définition *Agent-type-definition* • définitions *Composite-state-type-definition, Data-type-definition, Procedure-definition, Signal-definition*
^Classifier	
->feature	• dépend du genre d'entité
->participant – agrégation de l'extrémité d'association: = composite = aggregate = none	 • ensembles d'agents contenus • non applicable • si l'autre extrémité de l'association est telle que aggregation = composite: un ensemble d'entités
^GeneralizableElement	spécialisation <specialization>
.isAbstract	**abstract**
.isLeaf	non applicable
.isRoot	peut être dérivé de la définition de type

UML	SDL
->*generalization*	supertype – Le type identifié dans le cadre de l'expression <type expression> de la spécialisation <specialization> pour le type représenté par le classificateur
->specialization	peut être dérivé de *generalization*
^Namespace	unité de portée définie par le type
->*ownedElement*	entités d'un sous-ensemble de l'ensemble de définitions autorisées dans l'unité de portée – dépend du genre de type
^ModelElement	
->*taggedValue* • (virtuality) • virtual • redefined • finalized	 • virtualité <virtuality> – **virtual** – **redefined** – **finalized**
->*constraint*	contrainte <virtuality constraint>
->*supplierDependency* • Dépendance de réalisation pour les interfaces implémentées par la classe • Dépendance d'usage («create») pour les classes pour lesquelles des objets sont créés par des objets de cette classe • Dépendance de permission («import») concernant d'autres paquetages utilisant la classe • Relation de généralisation annotée	Dépendances pour lesquelles ce type est un fournisseur • <interface gate definition> **in with** <interface identifier> • <create line area> • <package use clause>/<definition selection list> • <type expression>
->*clientDependency* • Dépendance d'usage («create») pour les classes à partir desquelles des objets créent des objets de cette classe • Dépendance de permission «import» indiquant quel paquetage cette classe utilise • Relation de généralisation annotée	Dépendances pour lesquelles ce type est un client • <create line area> • <package use clause> concernant la définition de ce type • <type expression>
->*namespace*	l'unité de portée (englobante) détentrice du type d'entité ou de la procédure correspondant à la classe

3.2.8.1 Agent

Les dispositions suivantes s'appliquent aux types d'agents.

La caractéristique d'opération de signature est introduite afin de spécifier les paramètres formels, étant donné que l'UML ne fournit pas de paramètres pour les classes. Elle a le format d'une opération, avec comme nom d'opération celui du processus, et comme stéréotype "signature".

UML	SDL
{ «system» | «block» | «process» } Class	définition *Agent-type-definition* • si le terme *templateParameter* est défini, il s'agit d'un type d'agent paramétré, • si le terme *generalization* est défini, il s'agit d'un sous-type
.isActive = true = false	 • obligatoire • non applicable
^Classifier	
->feature • attributes • operations	 • définition *Variable-definition* • définition *Procedure-definition*, ou • définition *Procedure-definition* (signature) qui représente les paramètres <formal parameters> du type d'agent
->participant	contenance ou autre extrémité d'association à laquelle l'agent peut participer
^GeneralizableElement	
->generalization	la définition *Agent-type-definition* identifiée par l'identificateur *Parent-type-identifier* dans le cadre de ce type d'agent
^Namespace	unité de portée définie par le type d'agent
->ownedElement	entités du sous-ensemble <entity in agent> défini ci-dessous
^ModelElement	
->taggedValue • virtual • redefined • finalized	virtualité <virtuality> du type d'agent, sauf si le type est un type de système
->constraint	contrainte <virtuality constraint> concernant le type d'agent, sauf si le type est un type de système
->namespace	• définition *Package-definition* avec la définition *Agent-type-definition* • définition *Agent-type-definition* définissant le type d'agent

La *généralisation* ne peut être qu'une classe et doit être du même stéréotype que la classe considérée.

Le terme *ownedElement* (élément détenu) est restreint aux éléments de modèle qui représentent des entités du sous-ensemble suivant de <entity in agent>:

- définition *Signal-definition*,
- définition *Variable-definition*,
- définition *Procedure-definition*,
- <remote procedure definition>,
- <remote variable definition>,
- définition *Data-type-definition*,
- définition *Composite-state-type-definition*,
- définition *Interface-definition*,
- définition *Agent-type-definition*, ou
- définition *Agent-definition*.

L'*espace de noms* doit être un paquetage ou une classe de stéréotype «system», «block» ou «process». L'espace d'une classe «system» ou «block» ne peut pas être une classe «process» et l'espace d'une classe «system» ne peut pas être une classe «block» ou «process».

3.2.8.2 Procedure (procédure)

Une *procédure* est un stéréotype de classe qui correspond à une procédure SDL. L'objectif est de permettre la spécialisation des procédures, qui ne fait pas partie de l'UML.

UML	SDL
Classe «procedure»	définition *Procedure-definition* ou définition <remote-procedure-definition> • si un paramètre *templateParameter* est défini, il s'agit d'une procédure paramétrée • si une généralisation *generalization* est définie, il s'agit d'une sous-procédure
.isActive = false = true	 • obligatoire • non applicable
->taggedValue remote	 définition <remote-procedure-definition>
^Classifier	
->feature • attributes • operations	 • définition *Variable-definition* • définition *Procedure-definition*
->participant	non applicable
^GeneralizableElement	
->generalization	la définition *Procedure-definition* identifiée par l'identificateur *Procédure-identifier* facultatif dans le cadre de la définition *Procédure-definition* considérée
^Namespace	unité de portée définie par le type de processus
->ownedElement	entités du sous-ensemble de <entity in procedure> défini ci-dessous
^ModelElement	
->taggedValue • virtual • redefined • finalized	virtualité <virtuality> de la procédure • **virtual** • **redefined** **finalized**
->namespace	• définition *Package-definition* avec la définition *Procedure-definition* • définition *Agent-type-definition* avec la définition *Procedure-definition* • définition *Procedure-definition* avec la définition *Procedure-definition*

Les éléments *ownedElement* sont des éléments de modèle qui représentent des entités du sous-ensemble suivant de <entity in procedure>:

- définition *Data-type-definition,*
- définition *Variable-definition*, ou
- définition *Procédure-definition*.

L'espace *namespace* doit être un paquetage ou une classe de stéréotype «system», «block», «process» ou «procedure».

3.2.8.3 Types de données

Les classes de stéréotype *value* et *object* représentent des types de données: *value* est un stéréotype de classe qui correspond à un type de valeur SDL, et *object* est un stéréotype de classe qui correspond à un type d'objet SDL.

UML	SDL
• «value» Class • «object» Class	• *Value-type* • *Object-type* pour les deux: • si un paramètre *templateParameter* est défini, il s'agit d'un type paramétré • si une généralisation *generalization* est définie, il s'agit d'un sous-type
.isActive = false = true	 • obligatoire • non applicable
^Classifier	
->feature • attributes • operations	 • champ <field> dans la définition <structure-definition> • *Static-operator/Dynamic-operator* (opérateurs et méthodes)
->participant	contenance du type de données ou n'importe quel point d'extrémité d'association auquel le type peut participer
^GeneralizableElement	
->generalization	la définition *Data-type-definition* identifiée par l'identificateur *Data-type-identifier* dans le cadre de la définition *Data-type-definition* considérée
^Namespace	unité de portée définie par la définition *Data-type-definition*
-> ownedElement	les définitions *Data-type-definition* de l'unité de portée du type d'entité représenté par cette classe
^ModelElement	
-> namespace	• définition *Package-definition*, définition *Agent-type-definition*, définition *Procedure-definition*, définition *Data-type-definition* avec la définition *Data-type-definition*

generalization doit être une classe «value» ou «object».

ownedElement doit correspondre aux classes qui ont un mappage sur les définitions *Data-type-definition*.

namespace doit être un paquetage ou une classe de stéréotype «system», «block», «process», «procedure», «state», «object» ou «value».

3.2.8.4 State (état)

Un *état* est un stéréotype de classe qui représente un type d'état composite SDL. Le SDL possède la notion de type d'état composite (en plus de l'état composite) et, comme les machines à états UML ne la possèdent pas, cette notion est représentée, en *SDL UML*, par une classe de stéréotype state.

Les points de connexion (d'entrée et de sortie) d'état sont représentés comme étant des opérations, étant donné qu'il s'agit des propriétés visibles d'un type d'état composite.

UML	SDL
«state» Class	Définition *Composite-state-type-definition* • si un paramètre *templateParameter* est défini, il s'agit d'un type d'état composite paramétré • si une généralisation *generalization* est définie, il s'agit d'un sous-type
.isActive = false = true	 • obligatoire • non applicable
^Classifier	
->feature • attributes • operations	 • définition *Variable-definitions* • définition *Procedure-definitions* • points de connexion d'état tels que définis dans la définition *State-entry-point-definition*
->participant	non applicable
^GeneralizableElement	
->generalization	la définition *Composite-state-type-definition* identifiée par l'identificateur [*Parent-type-identifier*] dans le cadre de la définition *Composite-state-type-definition* considérée
^Namespace	unité de portée définie par la définition *Composite-state-type-definition* considérée
-> ownedElement	entités du sous-ensemble de <entity in composite state type> défini ci-dessous
^ModelElement	
.name	le nom *State-type-name*
->taggedValue • virtual • redefined • finalized	virtualité <virtuality> du type d'état composite
->constraint	contrainte <virtuality constraint>
-> namespace	définition *Package-definition*, définition *Agent-type-definition*, définition *Procédure-definition*, définition *Composite-state-type-definition* avec la définition *Composite-state-type-definition*

generalization doit être une classe «state».

ownedElement doit correspondre aux éléments de modèle qui représentent des entités du sous-ensemble suivant de <entity in composite state>:

- définition *Variable-definition*,
- définition *Data-type-definition*,
- définition *Procedure-definition*,
- <textual procedure definition>, ou
- définition *Composite-state-type-definition*.

namespace doit être un paquetage ou une classe de stéréotype «system», «block», «process», «procedure» ou «state».

3.2.9 Classifier (classificateur)

Un classificateur représente les caractéristiques communes: types d'entité, types d'interface, signaux et procédures. En l'absence de spécification spécifique pour les divers genres, on applique ce qui suit.

UML	SDL
Classifier	types d'entité, interfaces, signaux et procédures
->feature	variables, procédures, méthodes, opérateurs, paramètres de signaux – selon le genre d'entité
->participant	dépend du genre de classificateur
->powertypeRange	non applicable
^GeneralizableElement	dépend du genre
^Namespace	unité de portée
-> ownedElement	entités définies dans l'unité de portée de l'entité considérée
^ModelElement	
.template	le type paramétré si ce type est un paramètre <formal context parameter>
.templateParameter	paramètres <formal context parameters> pour un type paramétré
->constraint	contrainte <virtuality constraint>, si elle est présente
-> supplierDependency	dépend du genre
-> clientDependency	dépend du genre
->taggedValue • virtual • redefined • finalized	virtualité <virtuality>, si elle est présente • **virtual** • **redefined** • **finalized**

3.2.10 Comment (commentaire)

Un commentaire est une annotation attachée à un élément de modèle et représente une note ou un commentaire attaché à l'entité SDL correspondant à l'élément de modèle.

3.2.11 Component (composante)

Non applicable.

3.2.12 Constraint (contrainte)

Les contraintes générales ne sont pas prises en charge, car il n'existe pas de concept correspondant en SDL. Les contraintes de virtualité sont prises en charge. Le texte UML correspond au texte de la contrainte <virtuality constraint>.

UML	SDL
Constraint	
.body = atleast – Class name – Parameter name	• contrainte <virtuality constraint> concernant un type virtuel ou une procédure/méthode virtuelle • contrainte <virtuality constraint> concernant un paramètre <formal context parameter>
->constrainedElement	• un type virtuel si la contrainte est une contrainte de virtualité concernant un type virtuel • un paramètre <formal context parameter> si la contrainte est une contrainte de virtualité concernant un paramètre de contexte
^ModelElement	
.name	non applicable

3.2.13 DataType (type de données)

Non applicable.

3.2.14 Dependency (dépendance)

On utilise les dépendances suivantes:

- une dépendance «create» pour créer des lignes;
- une dépendance «import» pour représenter le fait qu'un paquetage utilise les définitions d'autres paquetages;
- une dépendance «realizes» pour les interfaces prises en charge par la classe.

3.2.15 Element (élément)

Non applicable en lui-même. Toutefois, des attributs sont utilisés. Voir le mappage des éléments plus spécifiques du méta-modèle.

3.2.16 ElementOwnership (détention d'élément)

Non applicable. Tous les éléments contenus font partie de la spécification de l'unité de portée contenante et la visibilité des éléments contenus est assurée par leurs propriétés.

3.2.17 ElementResidence (résidence d'élément)

Non applicable.

3.2.18 Feature (caractéristique)

Classe abstraite. Les attributs et les associations sont explicités plus en détail avec les éléments behaviouralFeature et structuralFeature.

3.2.19 Flow (flux)

Non applicable.

3.2.20 GeneralizableElement (élément généralisable)

Classe abstraite. Les attributs et les associations sont explicités plus en détail avec ses sous-classes.

3.2.21 Generalization (généralisation)

La spécialisation SDL est représentée par la généralisation en UML. Elle s'applique aussi aux procédures. Le méta-modèle UML possède la notion de généralisation, tandis que le SDL possède la notion opposée de spécialisation.

Le terme *Generalization* a le correspondant opposé <specialization> en SDL, mais il reflète la propriété d'héritage correspondante du sous-type.

UML	SDL
Generalization	spécialisation <specialization>
.discriminator	non applicable
->child	un sous-type
->parent	Identificateur *Parent-type-identifier* (type <base type> (dans l'expression <type expression>) du sous-type)
->powertype	non applicable
^ModelElement	
.name	non applicable

Le terme *Generalization* s'applique aux classes dont les stéréotypes correspondent aux sortes suivantes de types: systèmes, blocs, processus, états, objets, valeurs, aux procédures et aux interfaces et signaux (classificateurs de stéréotype «interface» ou «signal»).

Le terme *parent* doit être une seule classe ayant le même stéréotype que le sous-type, à l'exception des interfaces qui peuvent avoir plusieurs parents.

Les contraintes normales ne s'appliquent pas.

3.2.22 Interface

Les interfaces sont des classificateurs qui se mappent sur les interfaces en SDL. Les interfaces UML ne peuvent avoir que des opérations (correspondant aux procédures exportées en SDL), tandis que les interfaces SDL peuvent aussi avoir des variables et des signaux. En *SDL UML*, les interfaces ont donc une liste d'opérations qui représentent des signaux et des variables exportées. Le stéréotype «signal» sert à indiquer des signaux, tandis que les variables et les procédures sont représentées comme étant des opérations.

Les interfaces ne peuvent être appliquées qu'aux classes qui représentent des types d'agents et non aux classes qui représentent des procédures, un type d'état ou des types de données.

UML	SDL
Interface	définition *Interface-definition*
^Classifier	
->feature • *attribute* (*non pris en charge*) • *operation*	 • non applicable • définition *Signal-definition*, définition <remote-variable-definition> et définition <remote-procedure-definition>
->participant	extrémités d'association qui sont des interfaces, auquel cas aggregation = none
^GeneralizableElement	

UML	SDL
.isAbstract	**Abstract**
.isLeaf	non applicable
.isRoot	peut être dérivé de la définition *Interface-definition*
->generalization	super-interfaces, qui sont les interfaces identifiées par la spécialisation <interface specialization> pour la définition *Interface-definition* considérée
->specialization	peut être dérivée de la *generalization*
^Namespace	unité de portée de la définition *Interface-definition*
-> ownedElement	<entity in interface>: les entités définies dans l'unité de portée de la définition *Interface-definition* considérée
^ModelElement	
->supplierDependency	dépendance de réalisation représentant **implements**
->clientDependency	opposé de *supplierDependency*

Le terme *ownedElement* doit correspondre aux éléments de modèle qui représentent les entités suivantes de <entity in interface>:

- définition *Signal-definition*,
- <remote variable definition>, ou
- <remote procedure definition>.

3.2.23 Method (méthode)

UML	SDL
Method	corps de *Procedure-definition*, *Static-operator*/*Dynamic-operator* (opérateur ou méthode)
.body	non applicable
->specification	*Procedure-formal-parameter*, Signature*
^ModelElement	
->constraint	contrainte <virtuality constraint>

La *contrainte* décrit le fait que le corps d'une procédure ou d'un opérateur virtuel hérite de la spécification de comportement de la contrainte. Il s'agit d'une spécialisation de l'association de contrainte de l'élément "Method UML".

3.2.24 ModelElement (élément de modèle)

Un *élément de modèle* représente les propriétés communes des définitions (de type) d'entité SDL. En l'absence de spécification spécifique pour les diverses spécialisations, on applique ce qui suit.

UML	SDL
ModelElement	définition d'entité
.name	nom <name> d'entité
-> clientDependency	dépend du genre d'entité
-> constraint	dépend du genre d'entité
->implementationLocation	non applicable

UML	SDL
-> *namespace*	l'unité de portée (englobante) détentrice où l'entité est définie
-> *presentation*	non applicable
-> *supplierDependency*	dépend du genre d'entité
->*templateParameter*	dépend du genre d'entité

3.2.25 Namespace (espace de noms)

Un espace de noms représente une unité de portée SDL.

UML	SDL
Namespace	unité de portée
-> *ownedElement*	types d'entité définis dans l'unité de portée représentée par cet élément de modèle

Le terme *ownedElement* doit correspondre aux éléments de modèle qui représentent des entités en SDL conformément au *SDL UML*.

3.2.26 Node (nœud)

Non applicable, étant donné que la spécification de déploiement sort du cadre du SDL et par conséquent de celui du langage *SDL UML*.

3.2.27 Operation (opération)

Une opération représente soit une procédure, ou un opérateur ou méthode. Une opération est simplement utilisée pour représenter le fait qu'un type définit une procédure ou bien un opérateur ou méthode, conjointement avec sa signature. Une classe imbriquée de stéréotype procedure représente la procédure proprement dite, son nom étant celui de l'opération. Les opérateurs et les méthodes ne sont pas représentés par des classes avec stéréotype, mais par des méthodes UML. Les opérations stéréotypées par une signature représentent les paramètres formels du type d'agent englobant.

UML	SDL
Operation	*Procedure-definition*, *Static-operator*/*Dynamic-operator* (opérateur ou méthode) ou paramètres formels de type d'agent
.concurrency = sequential = guarded = concurrent	applicable uniquement pour les opérateurs et les méthodes • dépend de la sémantique des données • non applicable • dépend de la sémantique des données
.isAbstract = true = false	 • **abstract** • non **abstract**
.isLeaf = true = false	 • **finalized** • **virtual** ou **redefined**
.isRoot = true = false	non applicable
^Feature	

UML	SDL
.ownerScope = instance = classifier	• obligatoire • non applicable
.visibility = public = protected: default = private	• **exported procedure**, **public** (<visibility>) pour opérateur/méthode • **procedure**s locales, **protected** (<visibility>) pour opérateur/méthode • non applicable aux procédures, **private** (<visibility>) pour opérateur/méthode

Il est à noter que les paramètres formels des agents sont donnés par une opération stéréotypée par une signature portant le même nom que l'agent défini par la classe effective. Les autres opérations d'agents et de procédures représentent des procédures définies localement.

La virtualité d'une procédure est également spécifiée dans le cadre de la classe représentant la procédure. La virtualité des opérateurs et des méthodes est spécifiée dans le cadre des éléments Method.

Il est à noter que le type de valeur retournée par une procédure ou le type de résultat d'un opérateur ne peut pas être représenté directement par une propriété de l'opération *SDL UML* correspondante, mais par un paramètre de genre return (retour).

3.2.28 Parameter (paramètre)

Un paramètre représente un paramètre formel de procédure, d'opérateur ou de méthode. Les paramètres de type d'agent sont représentés par des paramètres (avec *kind* = in) de procédure portant le même nom que le type d'agent et stéréotypée par "signature".

UML	SDL
Parameter	*Procedure-formal-parameter*
.default value	pas de concept
.kind = in = inout = out = return	• *In-parameter* (mot clé **in** dans le genre <parameter kind>) • *Inout-parameter* (mot clé **in/out** dans le genre <parameter kind>) • non applicable • <sort> dans le résultat *Result* ou <opération result>
.name	*Variable-name*
-> type	*Data-type-reference-identifier* ou <sort> (dans le résultat *Result* dans le cadre de la définition *Procedure-définition*)

3.2.29 Permission

La permission «import» sert à représenter des clauses de référence de paquetage SDL, indiquant quels paquetages la classe considérée utilise.

UML	SDL
Permission «import»	clause <package use clause>
^Dependency	
->client	le client est une définition *Package-definition* ou une définition de type qui utilise des types définis dans le paquetage représenté par le fournisseur
->supplier	le fournisseur est un paquetage utilisé par le client
^ModelElement	
.name	non applicable

3.2.30 PresentationElement (élément de présentation)

Non applicable.

3.2.31 Relationship (relation)

Il s'agit d'un concept abstrait non applicable directement.

3.2.32 StructuralFeature (caractéristique structurelle)

Il s'agit d'un concept abstrait, utilisé uniquement pour l'élément Attribute. Voir cet élément pour plus de détails.

3.2.33 TemplateParameter (paramètre de gabarit)

Représente un paramètre de contexte en SDL.

3.2.34 Usage

L'usage de la sorte *create* représente la création de ligne.

3.3 Extension Mechanisms (mécanismes d'extension)

Les mécanismes d'extension de l'UML peuvent aussi être utilisés en SDL UML, mais, comme pour l'UML, il appartient à l'utilisateur de tels mécanismes d'en définir la signification.

3.4 Data Types (types de données)

Non utilisés. Les types prédéfinis du SDL sont supposés être prédéfinis comme des classes avec des stéréotypes et des propriétés appropriées.

3.5 Behavioural Elements (éléments comportementaux)

Il serait possible de fournir un mappage détaillé des éléments comportementaux de l'UML en SDL, mais cette version de la présente Recommandation ne le fait pas, sauf pour les signaux.

3.5.1 Common Behaviour (comportement commun)

On ne prévoit pas que le langage *SDL UML* sera utilisé pour spécifier des actions en détail, car le SDL fournit des mécanismes complets de spécification des actions. Aucun mappage n'est donc spécifié pour le paquetage Common Behaviour. Toutefois, un aperçu général d'un mappage possible est donné dans l'Appendice I.

3.5.2 Signal

Signal est défini en UML comme étant une sous-classe de Classifier. Il correspond à un signal SDL avec les restrictions suivantes.

UML	SDL
«signal»	*Signal-definition* • si un paramètre *templateParameter* est défini, il s'agit d'un signal paramétré • si une généralisation *generalization* est définie, il s'agit d'un sous-type
->context	peut être dérivé: les transitions qui envoient des signaux de ce type
->reception	peut être dérivé: les types d'agent qui ont ce signal dans leur ensemble de signaux d'entrée valides
^Classifier	
->feature	la liste des identificateurs *Data-type-reference-identifier* dans le cadre de la définition *Signal-definition*, c'est-à-dire les paramètres du signal
->participant	non applicable
^GeneralizableElement	
->generalization	la définition *Signal-definition* identifiée dans le cadre de l'expression <type expression> de la spécialisation <specialization> pour la définition *Signal-definition* considérée
^Namespace	unité de portée définie par la définition *Signal-definition*
-> ownedElement	non applicable
^ModelElement	
->taggedValue	pas de concept pour les signaux virtuels
->constraint	non applicable
->supplierDependency	non applicable
->clientDependency	non applicable
-> namespace	définition *Package-definition* ou définition <agent-type-definition> contenant la définition *Signal-definition*

generalization doit être un classificateur «signal».

namespace doit être un paquetage ou une classe d'agents («system», «block» ou «process»).

3.6 Collaborations

Un sous-ensemble de Collaborations sert à représenter la structure interne des automates/agents d'un agent et leurs connexions. Les ensembles d'agents contenus sont représentés par les éléments ownedElement d'une Collaboration:

- les extrémités AssociationEnds pour les ensembles d'agents, dans lesquels le terme *collaborationMultiplicity* représente le nombre <number of instances> et les rôles ClassifierRoles les types d'ensembles d'agents;
- les rôles AssociationRoles pour les canaux connectant les ensembles d'agents.

La notion d'Interaction n'est pas utilisée. En tant que parties de Collaborations, les Interactions équivalent simplement à des modèles de propriété de comportement et correspondent aux modèles décrits par le diagramme MSC [3].

Le langage UML n'est pas très clair sur les conséquences des Collaborations pour les instances du classificateur Classifier qui est représenté par la Collaboration (*representedClassifier*). Il est précisé dans la présente Recommandation qu'une Collaboration définit (*defines*) le contenu des instances du classificateur Classifier représenté.

3.6.1 AssociationEndRole (Rôle d'extrémité d'association)

Un rôle AssociationEndRole peut représenter un des éléments suivants d'un ensemble ***set*** {<interaction area> | <agent body area>}:

- un ensemble d'agents contenu,
- l'automate de l'agent contenant,
- une porte de l'agent contenant.

UML	SDL
AssociationEndRole	le classificateur d'interface associé à une porte d'un ensemble d'agents auquel un canal est connecté
.collaborationMultiplicity	• dans le cas d'un ensemble d'agents contenu: *Number-of-instances* • s'il s'agit de l'automate de l'agent: non applicable • dans le cas d'une porte de l'agent contenant: non applicable
->availableQualifier	l'ensemble des signaux et des procédure distantes utilisés sur le canal
->base	le qualificateur d'interface représenté par l'interface de la porte à laquelle le canal est connecté
^AssociationEnd	
^GeneralizableElement	non applicable
^ModelElement	
.name	nom *Gate-name*

3.6.2 AssociationRole (Rôle d'association)

Les rôles d'association représentent les canaux reliant entre eux les ensembles d'agents, l'automate de l'agent contenant et les portes de l'agent contenant.

UML	SDL
AssociationRole	une définition *Channel-definition*
.multiplicity	Le nombre de connexions découlant de la multiplicité de rôles AssociationEndRoles aux deux extrémités
->base	• l'association implicite entre les types des ensembles d'agents, • les associations composites entre le type d'agent englobant et les types des ensembles d'agents, • les associations implicites entre le type d'état composite de l'automate et le type d'agent contenant et les types des ensembles d'agents.
^Association	
^GeneralizableElement	les canaux suivent la généralisation du type d'agent contenant, c'est-à-dire que les connexions d'un type d'agent sont héritées
^ModelElement	
.name	noms *Channel-names*

3.6.3 ClassifierRole (Rôle de classificateur)

Un rôle de classificateur représente une interface d'une porte d'un ensemble d'agents.

UML	SDL
ClassifierRole	le type d'un ensemble d'agents contenu dans un agent
.multiplicity	identifique à la multiplicité du rôle AssociationEndRole correspondant, c'est-à-dire le nombre *Number-of-instances*
->availableContents	les signaux et les procédures exportées auxquels donne accès la porte de l'interface
->availableFeature	les signaux et les procédures exportées auxquels donne accès la porte de l'interface
->base	le type de l'ensemble d'agents
^Classifier	
^GeneralizableElement	l'interface suit la généralisation du type d'agent contenant, c'est-à-dire qu'elle est héritée
^ModelElement	
.name	nom de l'ensemble d'agents

3.6.4 Collaboration

Une collaboration représente la structure interne de l'automate/agent d'un agent et leurs connexions. Les propriétés définies par une collaboration sont les propriétés de toutes les instances de la classe représentée par le classificateur *representedClassifier*.

L'association *representedOperation* n'est pas utilisée, c'est-à-dire que les entités SDL qui seraient représentées par des opérations ont une structure interne qui n'est pas représentée par une collaboration.

UML	SDL
Collaboration	l'ensemble ***set*** { <interaction area> | <agent body area> } } dans le cadre du contenu <agent diagram content>
->constrainingElement	non applicable
->interaction	non applicable
->ownedElement	les éléments de l'ensemble ***set*** { <interaction area> | <agent body area> } }
->representedClassifier	le type d'agent avec l'ensemble ***set*** { <interaction area> | <agent body area> } } représenté par la collaboration considérée
->representedOperation	non applicable
^GeneralizableElement	suit la propriété correspondante du type d'agent conteneur, c'est-à-dire que toutes les propriétés d'un type d'agent spécifiées par une collaboration sont héritées
^Namespace	l'unité de portée du type d'agent avec l'ensemble ***set*** { { <interaction area> | <agent body area> } } representée par la collaboration considérée. Alors que l'élément ownedElement d'une collaboration ne représente que l'ensemble ***set*** { { <interaction area> | <agent body area> } }, l'élément ownedElement de l'espace Namespace du type d'agent représente toutes les entités figurant dans le type d'agent
^ModelElement	les attributs et les associations de l'élément ModelElement sont les mêmes que pour la classe représentant le type d'agent dont l'ensemble ***set*** {<interaction area> | <agent body area>}} est représenté par la collaboration considérée

3.6.5 Interaction

Non applicable.

3.6.6 Message

Non applicable.

3.7 Use Cases (cas d'utilisation)

Les cas d'utilisation ne sont pas utilisés dans la présente Recommandation. Ils décrivent un système sous une forme qui sort du cadre du SDL. Toutefois, ils pourront toujours être utilisés en combinaison avec le langage *SDL UML*.

3.8 State Machines (automates)

3.8.1 CallEvent (événement d'appel)

Un événement d'appel représente l'entrée d'une procédure distante en SDL.

UML	SDL
CallEvent	entrée d'appel de procédure distante
->operation	identificateur <remote procedure identifier>
^Event	
->parameters	paramètres <procedure formal parameters>

Il est à noter qu'en SDL, une entrée de procédure distante ne peut pas être ignorée, elle ne peut être que différée si elle n'est pas exécutée.

3.8.2 ChangeEvent (événement de modification)

Un événement de modification représente l'entrée d'un signal continu en SDL.

UML	SDL
ChangeEvent	signal *Continuous-signal*
.changeExpression	expression *Continuous-expression*
^Event	
->parameters	non applicable

En SDL, le signal *Continuous-signal* n'est évalué que lorsque aucun autre stimulus ne se trouve dans le port d'entrée.

3.8.3 CompositeState (état composite)

Un état composite représente un état composite SDL.

Le fait qu'un type d'état composite peut être un type virtuel ou un sous-type est représenté dans la classe de stéréotype «state».

UML	SDL
CompositeState	partie *Composite-state-part*
->subvertex	nœuds *State-start-nodes*, nœuds *States-nodes* et nœuds *Return-nodes* du graphe *Composite-state-graph*
.isConcurrent = false = true	 état composite autre qu'un état *Multi-state* état *Multi-state*
.isRegion = false = true	 état composite dans le cadre d'une définition *State-machine-definition* partition *State-partition*

3.8.4 Event (événement)

Un événement n'a pas d'équivalent en SDL. Toutefois, les sous-classes d'un événement ont un mappage adapté au SDL.

3.8.5 FinalState (état final)

Un FinalState représente un retour à partir d'un état composite SDL.

UML	SDL
FinalState	nœud *Return-node* non étiqueté à partir d'un état composite

3.8.6 Guard (garde)

Une garde représente une condition de validation en SDL.

UML	SDL
Guard	expression *Provided-expression*
.expression	expression *Boolean-expression*

Il est à noter qu'en UML, si une garde est évaluée à faux, l'événement d'entrée est ignoré, tandis qu'en SDL, l'événement d'entrée est différé (sauvegardé).

3.8.7 PseudoState (pseudo-état)

Un pseudo-état représente une abstraction qui englobe différents types de nœuds transitoires dans un graphe d'automates.

UML	SDL
PseudoState	Nœuds figurant dans une définition *State-machine-definition*
.kind = initial = deepHistory = shallowHistory = join = fork = junction = choice	 Nœud *State-start-node* non étiqueté d'un état composite pas de concept pas de concept pas de concept pas de concept nœud *Join-node* pour une fusion, nœud *Decision-node* pour un branchement pas de concept

3.8.8 SignalEvent (événement de signal)

Un événement de signal représente l'entrée d'un signal en SDL.

UML	SDL
SignalEvent	entrée d'un signal
->signal	instance de classe «signal»
^Event	
->parameters	paramètres du signal
.kind = in = inout = return	 obligatoire non applicable non applicable

3.8.9 SimpleState (état simple)

Un état simple représente un état de base en SDL.

UML	SDL
SimpleState	nœud *State-node* sans partie *Composite-state-part*

3.8.10 State (état)

Un état représente la combinaison d'un état SDL et du contenu d'un état composite SDL. Les procédures d'entrée et de sortie et les transitions internes font partie du contenu de l'état composite SDL, tandis que le terme *deferrableEvent* (événement pouvant être différé) correspond à la sauvegarde de l'état tel quel.

UML	SDL
State	nœud *State-node* et contenu de la partie *Composite-state-part*
->deferrableEvent	partie <save part> associée au nœud *State-node*
->entry	définition *Entry-procedure-definition* (de la partie *Composite-state-part*)
->exit	définition *Exit-procedure-definition* (de la partie *Composite-state-part*)
->doActivity	pas de concept
->internalTransition	transition *Transitions* internes (de la partie *Composite state-part*)

3.8.11 StateMachine (automate)

Un automate représente l'automate d'un agent ou d'un état composite.

UML	SDL
StateMachine	définition *State-machine-definition*
->context	définition *Agent-type-definition* avec la définition *State-machine-definition* ou la définition *Procedure-definition* avec le graphe *Procedure-graph*
->top	l'état composite englobant de la définition *State-machine-definition*
->transition	le graphe *State-transition-graph* de la définition *State-machine-definition*

3.8.12 StateVertex (sommet d'état)

Un sommet d'état représente l'abstraction d'un nœud dans un graphe d'automate. Les nœuds possibles en SDL sont (dans ce contexte): *State-node*, *Nextstate-node*, *Start-node*, *Return-node*, *Stop-node*, *Decision-node* et <join>.

UML	SDL
StateVertex	*Transition*
->outgoing	*Transition*s allant du nœud à des terminateurs *Terminator* de transition
->incoming	*Transition*s ayant le nœud comme terminateur *Terminator* de transition
->container	le graphe *State-transition-graph*

3.8.13 StubState (état souche)

Un état souche n'a pas de correspondant direct en SDL. Le concept similaire en SDL est le point de connexion d'entrée/sortie d'état. En SDL, il est considéré comme essentiel de définir les points d'entrée/sortie possibles en tant que partie de l'interface d'un état composite et de ne pas autoriser d'entrée dans un quelconque sous-état.

3.8.14 SubmachineState (état de sous-machine)

La sémantique de l'UML n'indique pas clairement si l'état de sous-machine est fondé sur des classes, même si la réutilisation est mentionnée. Le correspondant le plus proche en SDL est l'état composite fondé sur les types. La référence d'état composite pourrait constituer un autre correspondant en SDL et le fait que le même état de sous-machine peut faire l'objet de plusieurs références dans l'automate contenant permet de la prendre en charge, mais l'expansion de type macro décrite en UML indique qu'il ne s'agit pas d'une référence.

3.8.15 SynchState (état synchrone)

Un état synchrone n'a pas de correspondant en SDL.

3.8.16 TimeEvent (événement temporel)

Un événement temporel représente l'entrée d'un temporisateur en SDL.

UML	SDL
TimeEvent	entrée de temporisateur
.when	expression *Time-expression*
^Event	
->parameters *.kind* = in = inout = return	*Variable-identifier**s dans *Input-node* obligatoire non applicable non applicable

3.8.17 Transition

Une transition représente une transition en SDL. La seule différence est qu'une transition en SDL est associée à la *source*, alors qu'il s'agit d'une association de *transition*.

UML	SDL
Transition	*Transition*
->trigger	nœud *Input-node* ou signal *Continuous-signal*
->guard	expression *Provided-expression*
->effect	nœud *Graph-node** de *Transition*
->source	nœud *State-node* auquel la transition est associée
->target	terminateur *Terminator*

Lorsque la *source* et la cible sont situées dans des automates différents, l'effet sera exécuté dans l'automate détenteur de la *source*. En mappant sur le SDL, un *effect* traversant une frontière d'état pour aller d'un automate intérieur à un automate extérieur introduira un point <state exit point> dans l'automate intérieur. En revanche, un *effect* traversant une frontière d'état pour aller d'un automate extérieur à un automate intérieur introduira un point <state entry point> dans l'automate intérieur.

3.9 Activity Graphs (graphes d'activité)

Les graphes d'activité ne sont pas utilisés dans la présente Recommandation.

3.10 Model Management (gestion de modèle)

Dans le paquetage Model Management, on utilise seulement les méta-classes Package (paquetage) et Model (modèle); la méta-classe Subsystem (sous-système) n'est pas utilisée. Model et Package sont des classes de base de méta-modèle en UML. Aucun stéréotype particulier n'est défini, mais l'utilisation doit être restreinte comme décrit dans le présent sous-paragraphe.

3.10.1 ElementImport (importation d'élément)

Non applicable.

3.10.2 Model (modèle)

Un modèle représente une spécification *SDL-specification*. Cela implique qu'un modèle ne peut contenir qu'un ensemble de paquetages et une seule classe system.

NOTE – Eventuellement aussi des classes dans l'environnement, des types imbriqués, des interfaces et des associations.

UML	SDL
Model	spécification *SDL-specification*
^Package	
-> importedElement	pas de concept
^GeneralizableElement	non applicable
^Namespace	unité de portée Scope
-> ownedElement	ensemble *Package-definition*-**set** [définition *Agent-definition*]
^ModelElement	
.name	pas de concept (mais à étudier)
-> constraint	pas de concept
-> namespace	non applicable – une spécification SDL n'est pas définie dans une unité de portée

UML	SDL
.template	non applicable
.templateParameter	non applicable
-> supplierDependency	non applicable
->clientDependency	non applicable

Seuls les éléments ModelElements qui ont un mappage sur des entités contenues figurant dans la spécification *SDL-specification* peuvent être des éléments *ownedElement* d'un Model.

3.10.3 Package (paquetage)

Un paquetage représente une définition *Package-definition* en SDL.

UML	SDL
Package	définition *Package-definition*
-> importedElement	définitions rendues visibles pour ce paquetage par l'intermédiaire de la clause <package use clause> et de la liste <definition selection list>
^GeneralizableElement	non applicable
^Namespace	unité de portée de paquetage
-> ownedElement	entités du sous-ensemble de <entity in package> défini ci-dessous
^ModelElement	
.name	
-> constraint	pas de concept
-> namespace	l'unité de portée englobante qui contient le paquetage, par exemple la spécification *SDL-specification* ou la définition *Package-definition*
.template	non applicable
.templateParameter	non applicable
-> supplierDependency	pour un paquetage de la clause <package use clause>, le fait que ce paquetage soit utilisé par un certain nombre de paquetages
->clientDependency	pour un paquetage de la clause <package use clause>, le fait que ce paquetage utilise d'autres paquetages

Le *namespace* doit être Model ou Package.

L'élément *ownedElement* doit correspondre aux éléments de modèle qui représentent les éléments suivants du paquetage <entity in package>:

- définition *Package-definition*,
- définition *Agent-type-definition*,
- définition *Signal-definition*,
- <remote-variable-definition>,
- définition *Data-type-definition*,
- définition *Procedure-definition*,
- définition *Composite-state-type-definition*,
- définition *Interface-definition*.

Un paquetage SDL est complètement défini la définition *Package-definition*. Un Package définit des parties du paquetage SDL. Les parties qu'il définit dépendent de son contenu. Il appartient à l'utilisateur de choisir ce contenu, mais les parties doivent être compatibles avec les propriétés correspondantes définies dans la définition *Package-definition* SDL.

L'interface <interface> d'un en-tête <package heading> n'est pas directement couverte par cette extension. Indirectement, elle peut être représentée par l'attribut visibility (visibilité) de chaque Feature (caractéristique) dans les éléments *ownedElement* du Package.

Les dépendances entre Packages représentent des clauses <package reference clause> (dans la zone <package reference area>). La liste <definition selection list> n'est pas représentée. Pour un paquetage d'une clause <package use clause>, *supplierDependency* dénote les dépendances correspondant à l'utilisation du paquetage par un certain nombre d'autres paquetages. Pour un paquetage d'une clause <package use clause>, *clientDependency* dénote les dépendances correspondant à l'utilisation d'autres paquetages.

3.10.4 Subsystem (sous-système)

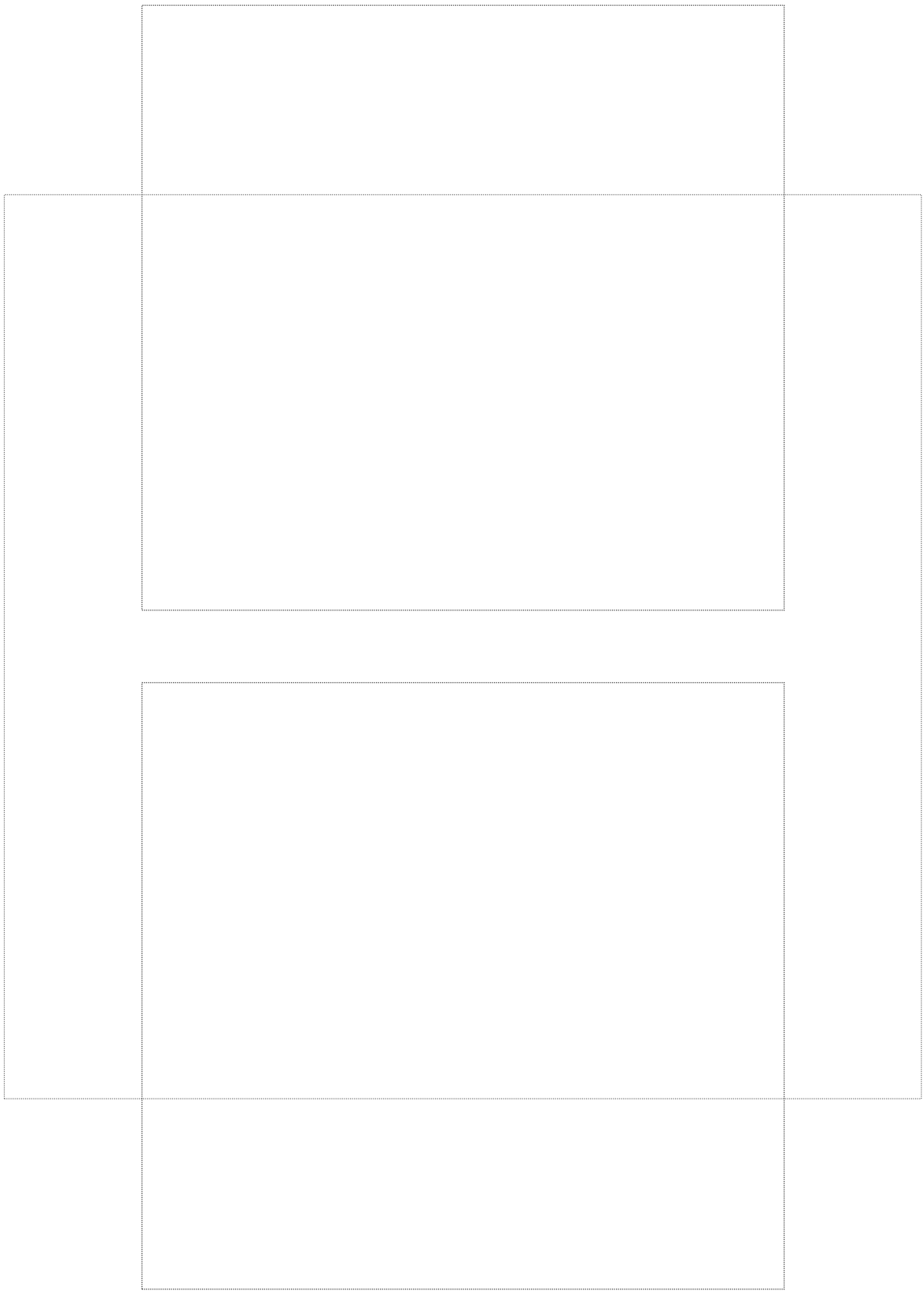
Non applicable.

APPENDICE I

Comportement commun

Le tableau suivant donne un aperçu général d'un mappage possible du paquetage Common Bahaviour UML.

UML	SDL
Action	Action
ActionSequence	Chaîne de transition
Argument	Paramètres de signal de sortie
AssignmentAction	Assignation
AttributeLink	Rattachement d'un identificateur à la définition de la variable ou du champ
CallAction	Appel de procédure distante ou invocation de méthode
ComponentInstance	Non applicable
CreateAction	Demande de création
DestroyAction	Pas de concept
DataValue	Valeur
Exception	Instance d'exception
Instance	Instance/entité
Link	Pas de concept
LinkEnd	Pas de concept
LinkObject	Pas de concept
NodeInstance	Pas de concept
Object	Entité fondée sur un type
Reception	Entrée
ReturnAction	Retour d'une procédure retournant une valeur
SendAction	Sortie
Signal	Signal – Voir le tableau séparé au 3.5.2
Stimulus	Consommation d'un signal ou exécution d'une procédure distante
TerminateAction	Arrêt
UninterpretedAction	Action informelle



SERIES DES RECOMMANDATIONS UIT-T

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

Imprimé en Suisse

Genève, 2000