UNION INTERNATIONALE DES TÉLÉCOMMUNICATIONS

UIT-T X.904

(12/97)

SECTEUR DE LA NORMALISATION
DES TÉLÉCOMMUNICATIONS
DE L'UIT

SÉRIE X: RÉSEAUX DE DONNÉES ET
COMMUNICATION ENTRE SYSTÈMES OUVERTS

Traitement réparti ouvert

Technologies de l'information – Traitement réparti ouvert – Modèle de référence: sémantique architecturale

Recommandation UIT-T X.904

(Antérieurement Recommandation du CCITT)

RECOMMANDATIONS UIT-T DE LA SÉRIE X

RÉSEAUX DE DONNÉES ET COMMUNICATION ENTRE SYSTÈMES OUVERTS

RÉSEAUX PUBLICS DE DONNÉES	
Services et fonctionnalités	X.1–X.19
Interfaces	X.20–X.49
Transmission, signalisation et commutation	X.50–X.89
Aspects réseau	X.90–X.149
Maintenance	X.150–X.179
Dispositions administratives	X.180–X.199
INTERCONNEXION DES SYSTÈMES OUVERTS	
Modèle et notation	X.200–X.209
Définitions des services	X.210–X.219
Spécifications des protocoles en mode connexion	X.220–X.229
Spécifications des protocoles en mode sans connexion	X.230–X.239
Formulaires PICS	X.240–X.259
Identification des protocoles	X.260–X.269
Protocoles de sécurité	X.270–X.279
Objets gérés de couche	X.280–X.289
Tests de conformité	X.290–X.299
INTERFONCTIONNEMENT DES RÉSEAUX	
Généralités	X.300–X.349
Systèmes de transmission de données par satellite	X.350–X.399
SYSTÈMES DE MESSAGERIE	X.400–X.499
ANNUAIRE	X.500–X.599
RÉSEAUTAGE OSI ET ASPECTS SYSTÈMES	
Réseautage	X.600–X.629
Efficacité	X.630–X.639
Qualité de service	X.640–X.649
Dénomination, adressage et enregistrement	X.650–X.679
Notation de syntaxe abstraite numéro un (ASN.1)	X.680–X.699
GESTION OSI	
Cadre général et architecture de la gestion-systèmes	X.700–X.709
Service et protocole de communication de gestion	X.710–X.719
Structure de l'information de gestion	X.720–X.729
Fonctions de gestion et fonctions ODMA	X.730–X.799
SÉCURITÉ	X.800–X.849
APPLICATIONS OSI	
Engagement, concomitance et rétablissement	X.850–X.859
Traitement transactionnel	X.860–X.879
Opérations distantes	X.880–X.899
TRAITEMENT RÉPARTI OUVERT	**X.900–X.999**

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

NORME INTERNATIONALE 10746-4

RECOMMANDATION UIT-T X.904

TECHNOLOGIES DE L'INFORMATION – TRAITEMENT RÉPARTI OUVERT – MODÈLE DE RÉFÉRENCE: SÉMANTIQUE ARCHITECTURALE

Résumé

La présente Recommandation | Norme internationale fait partie intégrante du modèle de référence du traitement réparti ouvert (ODP, *open distributed processing*). Elle contient une formalisation des concepts de modélisation ODP définis dans les articles 8 et 9 de la Rec. UIT-T X.902 | ISO/CEI 10746-2. La formalisation est obtenue par l'interprétation de chaque concept en fonction des constructions des différentes techniques de description formelle normalisées.

Source

La Recommandation X.904 de l'UIT-T a été approuvée le 12 décembre 1997. Un texte identique est publié comme Norme internationale ISO/CEI 10746-4.

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

Avant-propos

La présente Recommandation | Norme internationale fait partie intégrante du modèle de référence du traitement réparti ouvert (ODP, *open distributed processing*). Elle contient une formalisation des concepts de modélisation ODP définis dans les articles 8 et 9 de la Rec. UIT-T X.902 | ISO/CEI 10746-2. La formalisation est obtenue par l'interprétation de chaque concept en fonction des constructions des différentes techniques de description formelle normalisées.

La présente Recommandation | Norme internationale est accompagnée d'un amendement et d'un rapport technique. L'amendement s'intéresse à la formalisation du langage du point de vue informatique contenu dans la Rec. UIT-T X.903 | ISO/CEI 10746-3. Le rapport technique associé présente des exemples sur la manière possible d'appliquer des formalismes du modèle de référence ODP au développement de spécifications.

NORME INTERNATIONALE

RECOMMANDATION UIT-T

TECHNOLOGIES DE L'INFORMATION – TRAITEMENT RÉPARTI OUVERT – MODÈLE DE RÉFÉRENCE: SÉMANTIQUE ARCHITECTURALE

0 Introduction

La croissance rapide des applications réparties a fait naître le besoin d'un cadre pour coordonner la normalisation du traitement réparti ouvert (ODP, *open distributed processing*). Le modèle de référence du traitement réparti ouvert fournit ce cadre. Il établit une architecture qui permet d'intégrer la répartition, l'interfonctionnement, l'interopérabilité et la portabilité.

Le modèle de référence de base du traitement réparti ouvert (RM-ODP, *reference model of open distributed processing*) (voir les Rec. UIT-T X.901 à X.904 | ISO/CEI 10746) repose sur des concepts précis issus des développements récents dans le domaine du traitement réparti et s'appuie, dans la mesure du possible, sur l'utilisation des techniques de description formelle pour la spécification de l'architecture.

Le modèle RM-ODP se compose:

- de la Rec. UIT-T X.901 | ISO/CEI 10746-1: **Vue d'ensemble:** elle contient un aperçu général du modèle RM-ODP, en précise les motivations, le champ d'application et la justification, et propose une explication des concepts clés, ainsi qu'une présentation de l'architecture des systèmes ODP. Ce texte n'est pas normatif;
- de la Rec. UIT-T X.902 | ISO/CEI 10746-2: **Fondements:** elle contient la définition des concepts ainsi que le cadre analytique et la notation à utiliser pour la description normalisée de systèmes de traitement réparti (arbitraires). Elle s'en tient à un niveau de détail suffisant pour étayer la Rec. UIT-T X.903 | ISO/CEI 10746-3 et pour établir les prescriptions applicables à de nouvelles techniques de spécification. Ce texte est normatif;
- de la Rec. UIT-T X.903 | ISO/CEI 10746-3: **Architecture:** elle contient la spécification des caractéristiques requises pour qu'un système de traitement réparti puisse être qualifié d'ouvert. Il s'agit des contraintes que les normes ODP doivent respecter. Ce texte, qui utilise les techniques descriptives de la Rec. UIT-T X.902 | ISO/CEI 10746-2. Ce texte est normatif;
- de la Rec. UIT-T X.904 | ISO/CEI 10746-4: **Sémantique architecturale:** elle contient une formalisation des concepts de modélisation ODP définis dans les articles 8 et 9 de la Rec. UIT-T X.902 | ISO/CEI 10746-2 et une formalisation des langages de point de vue définis dans la Rec. UIT-T X.903 | ISO/CEI 10746-3. La formalisation est obtenue par l'interprétation de chaque concept en fonction des constructions des différentes techniques de description formelle normalisées. Ce texte est normatif.

La présente Recommandation | Norme internationale a pour objet de fournir une sémantique architecturale pour les systèmes ODP, ce qui se traduit par une interprétation des concepts de modélisation de base et de spécification définis dans la Rec. UIT-T X.902 | ISO/CEI 10746-2 et des langages de point de vue définis dans la Rec. UIT-T X.903 | ISO/CEI 10746-3; elle utilise les diverses caractéristiques de différents langages de spécification formelle. Une sémantique architecturale est élaborée pour quatre différents langages de spécification formelle: LOTOS, ESTELLE, SDL et Z, ce qui conduit à une formalisation de l'architecture des systèmes ODP. Un processus d'élaboration itérative et de retour a permis d'améliorer la cohérence des Rec. UIT-T X.902 | ISO/CEI 10746-2 et UIT-T X.903 | ISO/CEI 10746-3.

La mise au point d'une sémantique architecturale présente les avantages supplémentaires suivants:

- favoriser l'élaboration harmonieuse et uniforme des descriptions formelles de systèmes ODP;
- permettre une comparaison uniforme et cohérente des descriptions formelles de la même norme dans différents langages de spécification formelle.

La présente Recommandation | Norme internationale contient une interprétation, non pas de tous les concepts définis dans la Rec. UIT-T X.902 | ISO/CEI 10746-2, mais uniquement des concepts les plus fondamentaux. Pour les concepts architecturaux de niveau supérieur, une sémantique est fournie indirectement par la définition de ces concepts en fonction des concepts ODP fondamentaux.

Des exemples d'utilisation de certains des langages de spécification formelle dont il est question dans la présente Spécification sont donnés dans TR 10167 (Principes directeurs pour l'application d'ESTELLE, LOTOS et SDL).

Dans les articles suivants, la numérotation des paragraphes relatifs aux concepts est conforme à la numérotation utilisée dans la Rec. UIT-T X.902 | ISO/CEI 10746-2.

1 Domaine d'application

La présente Recommandation | Norme internationale, qui contient la définition d'une sémantique architecturale pour les systèmes ODP, permet:

- de fournir une formalisation des concepts de modélisation ODP;
- de favoriser une élaboration harmonieuse et uniforme des descriptions formelles des normes applicables aux systèmes répartis;
- de relier les concepts de modélisation ODP et les modèles sémantiques des langages de spécification LOTOS, SDL, ESTELLE et Z;
- de constituer une base pour une comparaison uniforme et cohérente des descriptions formelles de la même norme dans les langages de spécification utilisés pour élaborer une sémantique architecturale.

Ce texte est normatif.

2 Références normatives

Les Recommandations et les Normes internationales suivantes contiennent des dispositions qui, par suite de la référence qui y est faite, constituent des dispositions valables pour la présente Recommandation | Norme internationale. Au moment de la publication, les éditions indiquées étaient en vigueur. Toutes Recommandations et Normes sont sujettes à révision et les parties prenantes aux accords fondés sur la présente Recommandation | Norme internationale sont invitées à rechercher la possibilité d'appliquer les éditions les plus récentes des Recommandations et Normes Internationales indiquées ci-après. Les membres de la CEI et de l'ISO possèdent le registre des Normes Internationales en vigueur. Le Bureau de la normalisation des télécommunications de l'UIT tient à jour une liste des Recommandations de l'UIT-T en vigueur.

- ISO/CEI 8807:1989, *Systèmes de traitement de l'information – Interconnexion de systèmes ouverts – LOTOS – Technique de description formelle basée sur l'organisation temporelle de comportement observationnel.*
- Recommandation UIT-T Z.100 (1993), *Langage de description et de spécification du CCITT.*
- ISO/CEI TR 10167:1991, *Technologies de l'information – Interconnexion de systèmes ouverts – Principes directeurs pour l'application d'Estelle, LOTOS et SDL.*
- ISO/CEI 13568[1)], *Information technology – Programming Languages their Environments and System Software Interfaces, Z Specification language.*
- The Z Notation, *A Reference Manual, J.M. Spivey, International Series in Computer Science, Second Edition, Prentice-Hall International*, 1992.
- ISO/CEI 9074:1997, *Technologies de l'information – Interconnexion de systèmes ouverts (OSI) – Estelle: Technique de description formelle basée sur un modèle de transition d'état étendu.*

1) Actuellement à l'état de projet.

3 Définitions

3.1 Termes définis dans l'ISO/CEI 8807

La présente Recommandation | Norme internationale utilise les termes ci-après dont les équivalents anglais sont définis dans l'ISO/CEI 8807:

activation, actualisation de paramètres, choix, composition parallèle, conformité, définition de processus, définition de type, définition de type paramétré, dénotation d'action, désactivation, enrichissement, entrelacement, équation, événement, événement interne observable, expression de comportement, extension, garde, instanciation, liste de paramètres formels, liste de paramètres de valeurs, liste de portes formelles, prédicat de sélection, occultation de porte, opération, porte, réduction, sorte, synchronisation.

3.2 Termes définis dans la Recommandation UIT-T Z.100

La présente Recommandation | Norme internationale utilise les termes ci-après définis dans la Rec. UIT-T Z.100:

actif, action statement, arrêt, call, canal, clause "atleast", condition de validation, content parameter, créer, entrée, état suivant, export, finalized, import, initialisation, maintenant ("now"), procédure, procédure distante, procédure exportée, porte, "provided", réinitialisation, retard, retour, route de signaux (acheminement de signal), signal, signal continu, sortie, tâche, temporisateur, temps, transition, type de bloc, type de processus, type de service, type de système, type redéfini, type virtuel, variable exportée, variable importée, variable révélée, variable visualisée, visualisation.

3.3 Termes définis dans "The Z Base Standard"

La présente Recommandation | Norme internationale utilise les termes ci-après dont les équivalents anglais sont définis dans "The Z base standard":

affinement de données, affinement d'opération, calcul de schéma, composition de schéma, conjonction, description axiomatique, invariant, postcondition, précondition, remplacement, schéma (opération, état, encadrement).

3.4 Termes définis dans l'ISO/CEI 9074

La présente Recommandation | Norme internationale utilise les termes ci-après dont les équivalents anglais sont définis dans l'ISO/CEI 9074:

activité, activité-système, bloc de transition, canal, clause DELAY, clause de transition, clause FROM, clause PROVlDED, clause TO, clause WHEN, connexion, déconnexion, définition de canal, définition de corps de module, définition d'en-tête de module, détachement, état de contrôle, fonction, initialisation, instance de module, instance mère, instanciation, instruction d'affectation, interaction, libération, point d'interaction, point d'interaction externe, procédure, procédure primitive, processus-système, rattachement, rôle, sortie, transition, variable exportée.

4 Interprétation des concepts de modélisation

4.1 Sémantique architecturale en LOTOS

Le LOTOS est un langage de spécification formelle (FSL, *formal specification language*) normalisé (ISO/CEI 8807). Un didactitiel est donné dans la norme.

Le présent paragraphe contient une explication de la manière dont les concepts de modélisation de base peuvent être exprimés en LOTOS (voir ISO/CEI 8807). Il convient de signaler qu'il existe, en LOTOS, deux principales façons de modéliser les concepts définis dans la Rec. UIT-T X.902 | ISO/CEI 10746-2. Elles sont basées sur les parties algèbre de processus et typage de données ACT ONE du langage. Comme la formalisation ACT ONE des concepts s'applique aussi au langage SDL-92, elle est donnée dans un paragraphe indépendant. Voir 4.2.

Afin d'éviter toute confusion entre les terminologies ODP et LOTOS, les termes propres à la terminologie LOTOS sont en *italique* dans les paragraphes qui suivent.

4.1.1 Concepts de modélisation de base

4.1.1.1 Objet

Un objet est une *instanciation* de *définition de processus* LOTOS qui peut être identifiée de manière univoque.

4.1.1.2 Environnement (d'un objet)

L'environnement d'un objet est la partie d'un modèle qui ne fait pas partie de l'objet. En LOTOS, l'environnement d'un objet dans une spécification à un certain instant est donné par l'environnement de la spécification et par les autres *expressions de comportement* qui sont composées avec cet objet dans la spécification concernée à l'instant en question.

NOTE – L'environnement d'une spécification est vide si la spécification n'est pas paramétrée.

4.1.1.3 Action

En LOTOS, les actions sont modélisées sous forme d'*événements internes* ou d'*événements observables*. En LOTOS, tous les événements sont atomiques. Une action interne peut être donnée explicitement par le symbole d'*événement interne*, **i**, ou par l'occurrence d'un événement dont la *porte* associée est *cachée* du côté environnement.

En LOTOS, une interaction est représentée par une *synchronisation* entre deux *expressions de comportement* ou plus associées à des objets en un point d'interaction commun *(porte)*. Les interactions peuvent être du type:

- *synchronisation* pure en une *porte* commune, sans offre: il ne se produit aucun passage de valeurs entre les objets;
- ! et ! pour une *synchronisation* pure: aucune valeur n'est échangée entre les objets;
- ! et ? pour un passage de valeur sous réserve que l'événement ? contienne l'événement !; autre façon de voir: l'événement ! choisit une valeur parmi un ensemble de valeurs associées à l'événement;
- ? et ? pour un établissement de valeur: ici, une valeur est déterminée d'un commun accord à partir de l'intersection des ensembles de valeurs. Si cette intersection est l'ensemble vide, il ne se produit aucune *synchronisation* et donc aucune interaction.

Si une granularité non atomique des actions est exigée, on peut avoir recours à l'affinement d'événement, ce qui permet de modéliser les actions non instantanées et les actions se chevauchant. Il convient de noter que l'affinement d'événement est une opération complexe, en particulier lorsqu'il s'agit de maintenir la compatibilité de comportement.

En LOTOS, il n'existe aucune construction qui permette d'exprimer les relations de cause à effet, même si une représentation non formelle est parfois possible.

4.1.1.4 Interface

Une interface est une abstraction du comportement d'un objet, constituée par un sous-ensemble des actions observables de cet objet. En LOTOS, toutes les actions observables d'un objet exigent des *portes* qui leur permettent de se synchroniser avec l'environnement, on obtient donc généralement un sous-ensemble d'actions observables par une subdivision de l'ensemble des *portes* données dans la *définition de processus* associée à l'objet. Pour obtenir une interface, il est possible de *cacher* les *portes* non requises par l'interface considérée. Autre solution: il est possible d'effectuer une *synchronisation* uniquement sur un sous-ensemble des *portes* associées à un objet. Dans ce cas, les actions se produisant aux *portes* de la *définition de processus* qui n'appartiennent pas à l'ensemble sur lequel a porté la synchronisation, peuvent être considérées comme des actions internes à l'objet dans la mesure où l'environnement sur lequel sont synchronisées les portes constituant l'interface est concerné.

Il convient de noter que dans le cadre de cette définition, les interfaces d'un objet doivent utiliser des noms de porte différents, car il est impossible de faire la distinction entre des interfaces utilisant la même porte.

4.1.1.5 Activité

Une activité est un graphe d'actions acyclique orienté à racine unique, chaque noeud du graphe représentant un état du système et chaque arc une action. Pour qu'une action puisse se produire, elle doit satisfaire les préconditions de l'état du système.

4.1.1.6 Comportement (d'un objet)

Le comportement d'un objet est défini par l'*expression de comportement* LOTOS associée à la *définition de processus* qui constitue le squelette d'objet. Une *expression de comportement* peut être composée d'une séquence d'offres d'événements visibles de l'extérieur et d'*événements internes*. Le comportement effectif d'un objet, tel qu'il pourrait être enregistré dans une trace, dépend de l'*expression de comportement* associée à l'objet et du mode de configuration avec l'environnement. Le comportement effectif que l'objet présente dépend de l'*expression de comportement* de l'objet et du mode de synchronisation avec l'environnement. Un objet peut présenter un comportement non déterministe.

4.1.1.7 Etat (d'un objet)

C'est la condition d'un objet déterminant l'ensemble de toutes les séquences d'actions auxquelles l'objet peut participer. Cette condition est fonction de l'*expression de comportement* définie dans le squelette d'objet à partir duquel l'objet a été créé et éventuellement des relations courantes entre d'éventuelles variables locales existantes.

4.1.1.8 Communication

Une communication est une transmission d'information (par passage de valeur) entre deux objets en interaction ou plus. Il est impossible d'écrire directement des relations de cause à effet. Il convient aussi de signaler que la *synchronisation* peut être interprétée comme une communication.

4.1.1.9 Position dans l'espace

En LOTOS, l'abstraction correspondante est très éloignée de la notion de position dans l'espace. On peut uniquement faire correspondre l'espace à la structure du modèle de spécification. Pour les interactions en LOTOS, la position d'un événement – la position structurelle par rapport au modèle de spécification – est donnée par une *porte*. En LOTOS, l'abstraction correspondant à la notion de position dans l'espace à laquelle un *événement interne* peut se produire est très éloignée de cette notion. Cette abstraction est obtenue implicitement au moyen de la construction LOTOS *hide ... in* (cacher ... dans) qui permet une utilisation interne des *portes* dans un processus invisible pour l'environnement du processus, ou explicitement au moyen du symbole d'*événement interne*, **i**.

En LOTOS, il est possible qu'une même position dans l'espace soit employée pour plusieurs points d'interaction, car différentes *dénotations d'action* peuvent être associées à une même porte.

La position d'un objet est donnée par la réunion des positions des *portes* associées à cet objet, c'est-à-dire la réunion de toutes les positions des actions auxquelles l'objet participe.

4.1.1.10 Position dans le temps

En LOTOS, l'abstraction correspondante est très éloignée du concept de temps, seul l'ordre temporel est pris en considération, il n'existe donc pas de *position absolue dans le temps*. Toutefois, un positionnement dans le temps serait possible, si on utilisait une forme étendue de LOTOS dans laquelle des aspects temporels seraient incorporés.

4.1.1.11 Point d'interaction

Un point d'interaction est une *porte* à laquelle est associée un ensemble éventuellement vide de valeurs.

NOTE – Dans une spécification, des changements de position peuvent se traduire par des modifications des valeurs associées.

4.1.2 Concepts de spécification

4.1.2.1 Composition

- **d'objets:** un objet composite est un objet décrit au moyen d'un ou de plusieurs opérateurs de combinaison LOTOS, qui comprennent:
  - *l'opérateur d'entrelacement* (|||);
  - *les opérateurs de composition parallèle* (|| et |[liste-de-portes]|);
  - *l'opérateur d'activation* (>>);
  - *l'opérateur de désactivation* ([>);
  - *l'opérateur de choix* ([]).
- **de comportements:** il s'agit de la composition des *expressions de comportement* associées aux objets composants qui entrent dans la création d'un objet composite par composition. Les opérateurs servant à la composition de comportements sont les mêmes que ceux qui servent à la composition d'objets.

4.1.2.2 Objet composite

Un objet composite est un objet décrit au moyen d'un ou de plusieurs des *opérateurs* LOTOS d'*entrelacement*, de *composition parallèle*, de *désactivation*, d'*activation* et de *choix*.

4.1.2.3 Décomposition

- **d'objets:** il s'agit de l'expression d'un objet donné sous forme d'*objet composite*. Toutefois, il peut exister plusieurs façons de décomposer un objet composite.

- **de comportements:** il s'agit de l'expression d'un comportement donné sous forme de comportement composite. Il peut exister plusieurs façons de décomposer un comportement composite.

NOTE – On pourrait aussi considérer que la notion de décomposition de comportements est fournie de manière inhérente par les *opérations* et les équations ACT ONE associées à une *sorte*. En effet, ces *opérations* et *équations* fournissent toutes les combinaisons possibles de comportement. Par conséquent, une composition séquentielle pourrait par exemple être générée par des *opérations* appliquées séquentiellement. Chaque *opération* de la séquence doit satisfaire les équations nécessaires pour pouvoir se produire. Mais le fait de parler de composition de comportements est discutable, étant donné que les *opérations* et les équations existent déjà et définissent tous les comportements possibles.

4.1.2.4 Compatibilité de comportements

En LOTOS, des théories spécifiques ont été établies pour vérifier la compatibilité de comportement. Il n'existe pas d'éléments syntaxiques propres au LOTOS pour construire et garantir une compatibilité de comportement de manière générale. Toutefois, la norme LOTOS définit la notion de *conformité* qui constitue une base pour la prise en considération de la compatibilité de comportement.

Pour pouvoir déterminer si les comportements de deux objets sont compatibles, il faut introduire la notion de *conformité*. La *conformité* a pour objet l'évaluation de la fonctionnalité d'une réalisation par rapport à sa spécification; on peut considérer ici que le terme réalisation désigne une description moins abstraite d'une spécification.

Si **P** et **Q** sont deux processus LOTOS, l'instruction "**Q** est conforme à **P**" (s'écrivant **Q conf P**) signifie que **Q** est une réalisation valable de **P**. Autrement dit, si **P** peut exécuter une certaine trace σ et ensuite se comporter comme un certain processus **P'** et si **Q** peut aussi exécuter la trace σ et ensuite se comporter comme **Q'**, alors les conditions suivantes sur **P'** et **Q'** doivent être satisfaites: chaque fois que **Q'** peut refuser d'exécuter tous les événements d'un ensemble donné A d'actions observables, alors **P'** doit aussi être pouvoir refuser d'exécuter tous les événements de A.

Par conséquent, **Q conf P** si et seulement si, placé dans un environnement dont les traces sont limitées à celles de **P**, **Q** ne peut pas parvenir à un blocage lorsque **P** ne peut pas parvenir à un blocage. Autre définition possible: **Q** présente les blocages de **P** dans un environnement dont les traces sont limitées à celles de **P**.

Un objet peut être rendu compatible sur le plan du comportement avec un second objet après une certaine modification de son comportement, qui peut consister en une *extension* (ajouter un comportement supplémentaire) ou en une *réduction* (restreindre le comportement). Ce processus de modification d'un objet est appelé affinement (voir 4.1.2.5).

4.1.2.5 Affinement

L'affinement est le processus permettant de modifier un objet, par l'extension ou la réduction de son comportement ou encore par une combinaison des deux, de sorte qu'il soit conforme à un autre objet. Soient **P** et **Q** deux processus LOTOS, une *extension* de **P** par **Q** (s'écrivant **Q extends P**) signifie que **Q** a autant ou plus de traces que **P**, mais que dans un environnement dont les traces sont limitées à celles de **P**, **Q** présente les mêmes blocages. Une *réduction* de **P** par **Q** (s'écrivant **Q reduces P**) signifie que **Q** a autant ou moins de traces que **P**, mais que dans un environnement dont les traces sont limitées à celles de **Q**, **P** présente les mêmes blocages.

4.1.2.6 Trace

Une trace du comportement d'un objet depuis son état instancié initial jusqu'à un certain autre état est un enregistrement de la séquence finie d'interactions *(événements observables)* entre l'objet et son environnement.

4.1.2.7 Type d'un <X>

Les types qui peuvent être consignés explicitement en LOTOS pour des objets et des interfaces sont des types de squelette. En LOTOS, il n'existe aucune construction explicite qui permette de modéliser les types d'action. Une spécification LOTOS est constituée par une *expression de comportement* elle-même composée de *dénotations d'actions* (squelettes d'action). Soit ces squelettes d'action se produisent dans le cadre du comportement du système, auquel cas leur occurrence peut abusivement être considérée comme l'instanciation du squelette d'action, soit ils ne se produisent pas, auquel cas le squelette d'action reste non instancié. Les squelettes d'action peuvent être donnés par le symbole d'*événement interne*, **i**, ou par des offres d'événement en des *portes* qui peuvent ou non comporter une séquence finie de déclarations de valeurs ou de variables.

Le LOTOS ne permet pas de caractériser les actions directement; toutefois, une forme limitée de caractérisation des actions est intégrée dans l'élément *synchronisation* du LOTOS. En effet, on pourrait considérer que les *dénotations d'actions* (squelettes d'action) synchronisées doivent satisfaire le même type d'action pour que l'action puisse se produire. Toutefois, le LOTOS ne classe pas les éléments de caractérisation de ces *dénotations d'action* arbitraires et il est donc impossible d'attribuer un type formel à une action donnée. Un rôle de cause à effet pourrait être attribué de manière non formelle aux offres d'événement intervenant dans une interaction, mais ce n'est généralement pas le cas. Voir 4.1.1.8.

Le symbole d'*événement interne* peut servir à représenter un type d'action, la caractéristique commune de cet ensemble d'actions étant que ces actions n'ont pas de caractéristique.

Il convient de noter que si on affirme qu'en LOTOS, le seul prédicat possible pour les objets (et les interfaces) est qu'ils satisfassent à leur type de squelette, les concepts de type et de type de squelette donnés dans la Rec. UIT-T X.902 | ISO/CEI 10746-2 en sont réduits à la même technique de modélisation en LOTOS. Il n'existe donc pas de distinction en LOTOS entre un type dans sons sens de caractérisation général et un type de squelette dans son sens plus restrictif d'instanciation de squelette.

4.1.2.8 Classe d'un <X>

La notion de classe dépend du prédicat de type de caractérisation auquel les membres de la classe satisfont. Les objets, les interfaces et les actions peuvent satisfaire à de nombreux prédicats de type de caractérisation arbitraires. Un type pouvant être consigné est un type de squelette. Lorsque c'est le cas, la classe d'objets, d'interfaces ou d'actions associée à ce type est la classe de squelette.

NOTE – Il convient de noter que, si on affirme qu'en LOTOS, on ne peut classer les objets, interfaces et actions que par rapport à leur type de squelette, les concepts de classe et de classe de squelette donnés dans la Rec. UIT-T X.902 | ISO/CEI 10746-2 en sont réduits à la même technique de modélisation en LOTOS. Il n'existe donc pas de distinction en LOTOS entre une classe dans son sens de classification général et une classe de squelette dans son sens plus restrictif d'ensemble d'instances d'un type de squelette donné.

4.1.2.9 Sous-type/super-type

Etant donné qu'en LOTOS, les types qui peuvent être consignés pour les objets, les interfaces et dans une moindre mesure les actions sont des types de squelette, une relation de sous-typage en LOTOS est une relation qui peut exister entre types de squelette. Toutefois, il n'existe pas en LOTOS d'élément direct permettant de consigner directement des relations de sous-typage. Si le sous-typage est exigé, on peut alors avoir recours à une *extension* pour donner une relation de sous-typage sur la base d'une capacité de substitution; toutefois, ce n'est pas un élément explicitement prévu en LOTOS.

4.1.2.10 Sous-classe/superclasse

Etant donné qu'en LOTOS, les types qui peuvent être consignés pour les objets, les interfaces et dans une moindre mesure les actions sont des types de squelette, une relation de sous-classe existe entre deux classes lorsqu'une relation de sous-typage existe entre leurs types de squelette correspondants.

4.1.2.11 Squelette de <X>

– **Squelette d'objet:** c'est une *définition de processus* avec un certain moyen permettant de l'identifier de manière univoque une fois instanciée. Si aucune liste de paramètres de valeurs n'est donnée, l'identification d'objet sera alors impossible si plusieurs objets sont instanciés à partir du squelette d'objet.

  Concernant la combinaison de squelettes d'objet en LOTOS, il n'existe pas d'opérateur de combinaison sauf pour une forme limitée de domaine d'application utilisant le terme LOTOS *where*.

– **Squelette d'interface:** il s'agit de tout comportement obtenu à partir d'une *définition de processus* en ne considérant que les interactions au niveau d'un sous-ensemble des *portes* associées à la *définition de processus*. On obtient ce sous-ensemble de *portes* en *cachant* les *portes* qui ne sont pas requises pour les interactions considérées.

  Concernant la combinaison de squelettes d'interface en LOTOS, il n'existe pas d'opérateur de combinaison sauf pour une forme limitée de domaine d'application utilisant le terme LOTOS *where*.

– **Squelette d'action:** c'est une *dénotation d'action*, qui peut être un symbole d'*événement interne*, un identificateur de porte ou un identificateur de porte suivi par une séquence finie de déclarations de valeurs ou de variables.

NOTE – La définition de *dénotation d'action* donnée ici est imaginée car le LOTOS ne prend pas vraiment en charge le concept de squelette d'action. En LOTOS, les comportements possibles sont spécifiés par la combinaison de *dénotations d'action* sous une certaine forme. En LOTOS, tout ce qu'on puisse faire, c'est relier un squelette à une *dénotation d'action*. Toutefois, d'après le texte de la Rec. UIT-T X.902 | ISO/CEI 10746-2, un squelette d'action doit regrouper les caractéristiques des actions. Ceci n'est pas vérifié en LOTOS car les offres d'événement *(dénotations d'action)* existent de façon isolée et il est impossible de les rassembler et de leur appliquer un squelette qui les caractérise.

La composition de squelettes d'action peut abusivement être comparée à une *synchronisation* avec passage de valeur ou génération de valeur. Dans ce cas, deux squelettes d'action (ou plus) déterminent un squelette d'action commun pour que la *synchronisation* puisse se produire, c'est-à-dire un squelette d'action avec les caractéristiques communes à tous les squelettes d'action intervenant dans la *synchronisation* (composition).

4.1.2.12 Signature d'interface

La signature d'une interface, définie comme étant un ensemble de squelettes d'action associés aux interactions de l'interface, est représentée en LOTOS par un ensemble de *dénotations d'action*. Les membres de cet ensemble sont les *dénotations d'action* qui nécessitent une *synchronisation* avec l'environnement pour pouvoir se produire.

4.1.2.13 Instanciation (d'un squelette de < *X* >)

– **d'un squelette d'objet:** c'est le résultat d'un processus qui utilise un squelette d'objet pour créer un nouvel objet dans son état initial. Ce processus met en jeu l'*actualisation* de la *liste de portes formelles* et des *paramètres formels* d'une *définition de processus* par un réétiquetage au cas par cas à partir d'une liste de portes spécifiées et d'une liste de paramètres effectifs. Les éléments de l'objet créé sont fonction du squelette d'objet et des éventuels paramètres utilisés pour l'instancier.

– **d'un squelette d'interface:** c'est le résultat d'un processus permettant de créer une interface à partir d'un squelette d'interface. L'interface créée peut ensuite être utilisée par l'objet auquel elle est associée en vue d'une interaction avec l'environnement. Les éléments de l'interface créée sont fonction du squelette d'interface et des éventuels paramètres utilisés pour l'instancier.

– **d'un squelette d'action:** il s'agit de l'occurrence d'une action en LOTOS, pouvant impliquer la réécriture d'expressions ACT ONE.

4.1.2.14 Rôle

Un rôle est un nom associé à une *définition de processus* dans le squelette d'un objet composite (c'est-à-dire une composition LOTOS d'*expressions de comportement*). En tant que tels, les rôles ne peuvent être utilisés comme des paramètres. Toutefois, il est possible d'assigner des valeurs de données à chacun des rôles d'une composition afin de les distinguer ou d'y accéder de manière spécifique.

4.1.2.15 Création (d'un < *X* >)

– **d'un objet:** c'est l'instanciation d'un squelette d'objet, entrant dans le cadre du comportement d'un objet existant.

– **d'une interface:** les objets et les interfaces étant modélisés de la même manière en LOTOS (via les *définitions de processus)*, la création d'objet correspond à la création d'interface. La définition de la création d'interface est donc donnée par la définition de la création d'objet.

4.1.2.16 Introduction (d'un objet)

Il s'agit de l'*instanciation* du comportement associé à une spécification LOTOS.

4.1.2.17 Suppression (d'un < *X* >)

- **d'un objet:** c'est la terminaison de l'*instanciation* d'un processus. Cela peut se faire au moyen de l'*opérateur de désactivation* LOTOS, de l'*expression de comportement (arrêt)* d'inaction LOTOS pour laquelle le passage de commande est impossible ou de l'*expression de comportement (sortie)* de terminaison effective pour laquelle le passage de commande est possible via l'*opérateur d'activation.*

- **d'une interface:** c'est le processus selon lequel le futur comportement d'un objet est limité au comportement pour lequel la participation de l'interface à supprimer n'est pas requise.

4.1.2.18 Instance d'un type

- **de squelette d'objet:** une instance d'un squelette d'objet est représentée en LOTOS par une instanciation de ce squelette d'objet ou par une substitution acceptable d'une instanciation de ce squelette d'objet. Ici le substitut acceptable doit prendre les caractéristiques qui identifient le type en question. Un substitut acceptable pourrait donc être un autre squelette qui est compatible sur le plan du comportement avec le premier. Cette compatibilité pourrait être obtenue par *extension* (voir 4.1.2.4). L'utilisation de cette relation permet de garantir que toutes les caractéristiques du type considéré sont incluses. On pourrait toutefois trouver une forme plus faible de relation de satisfaction de type pour laquelle il ne serait pas nécessaire d'inclure toutes les caractéristiques associées à un squelette donné, mais seulement une partie.

- **de squelette d'interface:** étant donné qu'un squelette d'interface est représenté de la même manière qu'un squelette d'objet (via une *définition de processus* en LOTOS), le texte ci-dessus s'applique aussi (après remplacement du terme objet par le terme interface) à l'instance d'un squelette d'interface.

- **de squelette d'action:** une instance de squelette d'action *(dénotation d'action)* est représentée en LOTOS par une autre *dénotation d'action* proposant une offre d'événement équivalente.

4.1.2.19 Type de squelette (d'un < *X* >)

Le type de squelette d'un < *X* > est un prédicat exprimant qu'un < *X* > est une instance d'un squelette donné, où un < *X* > désigne un objet, une interface ou une action.

4.1.2.20 Classe de squelette (d'un < *X* >)

La classe de squelette d'un < *X* > est l'ensemble de tous les < *X* > qui sont des instances de ce squelette de < *X* >, où un < *X* > désigne un objet, une interface ou une action.

NOTE – L'application de la notion de classe de squelette d'une action au LOTOS est limitée, car les squelettes d'action, instanciations de squelette d'action et types de squelette d'action ne sont pas explicitement prévus en LOTOS.

4.1.2.21 Classe dérivée/classe de base

Si le squelette d'une classe correspond au squelette d'une seconde classe modifié de façon incrémentielle, la première classe est alors une classe dérivée de la seconde et la seconde une classe de base de la première.

Les squelettes LOTOS peuvent être modifiés de façon incrémentielle par *extension*, *enrichissement* et modification des *types de données* ou par modification du comportement. Toutefois, les modifications de comportement peuvent causer des problèmes, concernant en particulier:

- le sous-typage: un non-déterminisme peut être introduit dans le système lorsque les initiales du squelette hérité et celles du squelette modifié sont identiques, le sous-typage ne peut donc pas être garanti;

- la nécessité d'une réorientation d'auto-référence: toute référence à un squelette dérivé d'un squelette père doit être réorientée vers le squelette dérivé, ce qui n'est pas toujours possible.

Le LOTOS standard n'apporte pas de solution satisfaisante à ces problèmes.

4.1.2.22 Invariant

En LOTOS, les seuls invariants pouvant être consignés sont les *définitions de processus*. Il est impossible d'attacher à une *définition de processus* un invariant autre que la *définition de processus* elle-même.

4.1.2.23 Précondition

Une précondition est un prédicat qui doit être vrai pour qu'une action puisse se produire et qui peut être exprimé directement en LOTOS par un ou plusieurs des moyens suivants:

- séquences d'actions;
- *gardes* et *prédicats de sélection.*

4.1.2.24 Postcondition

En LOTOS, l'occurrence d'une action est indépendante de l'état du système après l'occurrence de l'action. En tant que tel, le LOTOS ne permet pas d'exprimer directement des postconditions.

4.2 Sémantique architecturale en ACT ONE

Le présent paragraphe contient une formalisation des concepts de modélisation de base et de spécification définis dans la Rec. UIT-T X.902 | ISO/CEI 10746-2. ACT ONE n'est pas en lui-même un langage FSL normalisé, mais il est utilisé dans les langages FSL normalisés LOTOS et SDL-92 et permet de formaliser d'une autre manière les concepts mentionnés plus haut. La formalisation ACT ONE des concepts définis dans la Rec. UIT-T X.902 | ISO/CEI 10746-2 est donc présentée dans un paragraphe distinct.

4.2.1 Concepts de modélisation de base

4.2.1.1 Objet

Un objet est une instance de *sorte* pouvant être identifiée de manière univoque. Il convient de signaler que les objets modélisés de cette façon doivent être spécifiés pour avoir une certaine forme d'existence. Cela peut se faire par un style de spécification d'algèbre de processus. On peut citer comme exemple de ce style la récurrence dans les *définitions de processus*, où l'objet est un élément de la *liste de paramètres de valeurs* associée à la *définition de processus* considérée. Autre exemple: on peut utiliser des clauses **let ... in** pour modéliser les objets avec une forme d'existence. Dans ces deux styles, des *gardes* ou des *prédicats de sélection* sont nécessaires pour garantir que les instanciations des définitions de *sorte* sont uniques.

4.2.1.2 Environnement d'un objet

L'environnement d'un objet n'est pas prévu en ACT ONE. Cette notion ne peut être prise en considération qu'au moyen de l'algèbre de processus et des expressions ACT ONE qui existent dans cette algèbre. En réalité, on peut considérer que l'environnement d'un objet correspond à tout ce qui constitue l'algèbre de processus autre que les expressions ACT ONE représentant l'objet en question et les *opérations* sur cet objet. En effet, l'environnement sert à engendrer l'occurrence d'*opérations* sur un objet. Pour cette notion d'environnement, il n'est pas nécessaire que les *opérations* sur un objet soient invoquées par d'autres objets, d'où certaines conséquences sur des notions comme l'interaction; à savoir qu'ici, l'interaction n'a pas lieu entre objets mais entre un objet et une certaine agence extérieure – ici l'algèbre de processus.

4.2.1.3 Action

Une action correspond à l'occurrence d'une *opération*. Il convient de noter qu'en général, il n'y a pas de distinction inhérente entre une interaction et une action interne d'un point de vue purement ACT ONE. En effet, les éventuelles actions sont modélisées au moyen d'*opérations* dans la signature d'une *sorte* ACT ONE et ces *opérations* peuvent se produire ou non, en fonction de l'occurrence des expressions ACT ONE existant dans l'algèbre de processus. Les actions internes ne sont donc pas explicitement prévues en ACT ONE. On pourrait toutefois modéliser une forme d'action interne au moyen de *sortes* définies localement dans l'algèbre de processus. Sinon, toutes les *opérations* déclarées dans l'algèbre de processus peuvent être considérées comme des interactions. Les opérations utilisées pour satisfaire ces *opérations*, c'est-à-dire dans les *équations* associées aux *opérations* envisagées, peuvent être considérées comme des actions internes. Par exemple, si des processus appellent une *opération pop2* qui supprime deux éléments dans une file d'attente et qui utilise l'*opération pop* deux fois dans ses *équations* associées, alors *pop2* peut être considérée comme une interaction, tandis que *pop* peut être considérée comme une action interne. Toutefois ce traitement des actions internes pose le problème suivant: il n'y a pas de notion de transitions spontanées en tant que telles, comme par exemple avec le symbole d'*événement interne* **i** dans l'algèbre de processus.

Il convient de signaler que pour cette forme d'interaction, il n'est pas nécessaire que deux objets ou plus interagissent au sens de l'algèbre de processus, c'est-à-dire par une *synchronisation* en une *porte* commune. Au contraire, ici l'interaction peut être interprétée comme quelque chose qui est causé indirectement par l'environnement et qui n'est pas nécessairement causé par un objet, c'est-à-dire qui n'est pas causé par une autre instance de *sorte* modélisant un objet. Il se pourrait donc que l'occurrence d'une offre d'événement qui ne fait pas intervenir d'expressions ACT ONE entraîne l'exécution d'une interaction, par exemple par une occurrence d'offre d'événement qui se traduit par l'*instanciation* d'une *définition de processus* dont la *liste de paramètres de valeurs* contient une *opération* (interaction) sur un objet (ou sur des objets).

4.2.1.4 Interface

Une interface est constituée par les *opérations* et *équations* associées à l'objet.

4.2.1.5 Activité

Une activité est une séquence d'*opérations* appliquées à une *sorte* donnée. Ces *opérations* doivent satisfaire les *équations* associées à la *sorte*. Chaque *opération* de la séquence qui se produit, c'est-à-dire chaque *opération* de l'activité, doit avoir des préconditions qui satisfont aux postconditions correspondant à l'occurrence de l'*opération* précédente. Les préconditions et les postconditions associées aux *opérations* sont respectivement définies aux 4.2.2.23 et 4.2.2.24.

4.2.1.6 Comportement (d'un objet)

Le comportement d'un objet modélisé en ACT ONE dépend des *opérations* associées au squelette d'objet et de la valeur courante de l'état de l'objet. En effet, cette valeur peut servir à limiter les éventuelles *opérations* qui peuvent avoir lieu, on peut par exemple empêcher à certaines *opérations* dont les *équations* ne sont pas valables pour cette valeur de l'état de se produire.

En ACT ONE, aucune contrainte n'est prévue explicitement concernant l'occurrence d'*opérations* comme les contraintes de séquence, de non-déterminisme, de parallélisme et de temps réel. Mais ACT ONE fournit des *opérations* et des *équations* qui intègrent toutes les contraintes possibles, c'est-à-dire tous les comportements possibles avec leurs contraintes associées.

ACT ONE ne contient pas l'élément permettant de modéliser les actions internes de manière spécifique. En effet, il n'y a pas de notion de transitions spontanées comme cela pourrait être le cas dans l'algèbre de processus avec le symbole d'*événement interne* **i** par exemple.

Il convient de signaler qu'il n'y a pas vraiment de notion d'occurrence du comportement seule en ACT ONE. La notion d'occurrence du comportement en ACT ONE est normalement associée aux expressions ACT ONE qui sont évaluées dans l'algèbre de processus.

4.2.1.7 Etat (d'un objet)

L'état d'un objet est la valeur courante qui est associée à une instance de *sorte* modélisant un objet. Il convient de signaler qu'une *sorte* modélisant un objet doit contenir un identificateur servant à faire la distinction entre les différentes instances de la *sorte*. Toutefois, la valeur de cet identificateur ne fait pas partie de l'état; en effet, cette valeur doit rester inaltérable dans les *opérations* et *équations* associées à la *sorte*.

4.2.1.8 Communication

Cette notion n'est pas prise en charge en ACT ONE. On pourrait modéliser une forme abstraite de communication au moyen d'un style de spécification d'algèbre de processus. Toutefois, cela ne reflète pas le texte de la Rec. UIT-T X.902 | ISO/CEI 10746-2, dans le sens où un objet ne transmet pas d'information à un autre objet. Ici la communication se fait avec l'environnement (l'algèbre de processus) et non avec d'autres objets.

4.2.1.9 Position dans l'espace

La notion de position dans l'espace n'est pas explicitement prise en charge en ACT ONE. Si cette notion est exigée, elle peut être intégrée dans le modèle de spécification, par exemple au moyen d'une *sorte* modélisant une position dans l'espace utilisée dans les *opérations* dont la position dans l'espace doit être contrôlée.

4.2.1.10 Position dans le temps

La notion de position dans le temps n'est pas explicitement prise en charge. Toutefois, si la notion de temps est reliée à l'état courant d'un objet donné, c'est-à-dire aux changements d'état qui se sont produits et à ceux qui peuvent se produire, la position dans le temps à laquelle une action donnée peut se produire peut alors être déterminée dans une certaine mesure par l'état courant de l'objet.

La position dans le temps à laquelle une action peut se produire peut également être intégrée dans la spécification, par exemple au moyen d'une *sorte* modélisant une position dans le temps utilisée dans les *opérations* dont la position dans le temps doit être contrôlée.

4.2.1.11 Point d'interaction

Cette notion n'est pas directement prise en charge en ACT ONE. Toutefois, elle pourrait être intégrée dans une spécification; par exemple, au moyen d'une *sorte* utilisée dans les *opérations* qui figurent dans l'ensemble des interfaces situées à la même position. Cela signifie que toutes les *opérations* comprises dans cet ensemble nécessitent un paramètre d'entrée *(sorte)* indiquant leur position. Si c'est nécessaire, plusieurs points d'interaction peuvent être modélisés de manière qu'ils existent à la même position, par exemple au moyen d'une *opération* qui crée une *sorte* de position nécessitant plusieurs *sortes* de point d'interaction en entrée. Les *opérations* et *équations* associées à ces *sortes* doivent permettre d'identifier des positions et des points d'interaction distincts.

4.2.2 Concepts de spécification

4.2.2.1 Composition

– **d'objets:** deux objets arbitraires ne peuvent généralement pas être combinés en ACT ONE et conduire à un résultat signifiant, c'est-à-dire à un objet composite avec son propre comportement, etc. La forme de composition la plus vraisemblable en ACT ONE est basée sur une *opération* de construction qui a deux objets ou plus comme paramètres d'entrée et un moyen permettant d'identifier de manière univoque l'objet créé. Considérons l'*opération* de construction ACT ONE suivante*:*

makeCO: Id, Ob1, Ob2 -> CO

Ici, un objet composite est créé à partir de deux autres objets qui ont leurs propres *opérations* et *équations* associées, c'est-à-dire leurs propres comportements. L'objet *co: CO = makeCO(id1,ob1,ob2)* est composé des objets *ob1* et *ob2*, mais l'objet *co* n'a pas de comportement en tant que tel. En effet, les comportements associés aux objets *ob1* et *ob2* ne sont pas applicables aux instances de *CO*.

Pour résoudre ce problème, on peut spécifier des *opérations* et des *équations* supplémentaires pour l'objet composite. La forme de ces *opérations* et *équations* et leur relation avec les objets composants déterminent alors la forme de la composition. Par exemple, si les *opérations* et les *équations* de l'objet composite permettent simplement un accès aux objets composants isolés, on obtient alors une forme de délégation, la composition des objets composants représentant une agrégation. Si les *opérations* et les *équations* de l'objet composite sont spécifiées de telle manière qu'elles modifient le comportement des objets composants, on obtient alors une forme de composition qui se rapproche plus du texte de la Rec. UIT-T X.902 | ISO/CEI 10746-2. Il convient de signaler, toutefois, que la notion de composition donnée dans la Rec. UIT-T X.902 | ISO/CEI 10746-2 ne nécessite pas de spécification supplémentaire, c'est-à-dire qu'elle englobe la composition d'objets et de comportements existants en vue de la génération de nouveaux objets et comportements, mais qu'elle n'englobe pas la spécification d'un comportement supplémentaire permettant une composition signifiante d'objets composants.

– **de comportements:** ACT ONE n'intégrant pas d'opérateurs de composition spécifiques, la notion de composition de comportements n'est pas prévue explicitement. Mais il convient de signaler qu'il existe une forme de composition en ACT ONE: celle de l'*enrichissement*. Toutefois, cette forme ne peut généralement pas être classée comme une composition, car elle ne permet pas en elle-même une composition de comportements (ou d'objets) explicite. L'*enrichissement* en lui-même, c'est-à-dire sans autre spécification, n'offre aucun élément de composition. Tous les *types de données* existent indépendamment les uns des autres lorsque l'*enrichissement* est appliqué. C'est uniquement lorsque les *opérations* et *équations* qui utilisent les *sortes* rendues disponibles par *enrichissement* sont spécifiées, que la notion de composition peut être appliquée. Toutefois, cela peut ne pas refléter convenablement le texte de la Rec. UIT-T X.902 | ISO/CEI 10746-2 dans le sens où deux comportements ne sont pas simplement combinés. Une autre spécification est nécessaire pour combiner les comportements. Il convient d'ajouter cependant que l'*enrichissement* rend disponibles toutes les *opérations* et *équations* existantes des *sortes* combinées. La spécification supplémentaire nécessaire pour combiner les comportements n'inclut donc pas les *opérations* et *équations* des *sortes* introduites par *enrichissement.*

On pourrait aussi obtenir une forme de composition de comportements par *actualisation* de *types de données paramétrés*. Pour cela, l'*actualisation d'un type de données* doit satisfaire les *sortes formelles*, les *opérations* et les *équations* du *type de données actualisé*. C'est ce qui permet, en ACT ONE, de se rapprocher le plus de la notion de composition de comportements telle qu'elle est définie dans la Rec. UIT-T X.902 | ISO/CEI 10746-2. Cependant, le fait de parler de composition de comportements est discutable, car il ne peut pas y avoir de comportement d'un *type de données paramétré* tant que ce type n'a pas été *actualisé*; c'est-à-dire qu'une instance de cette *sorte* ne peut pas se produire dans l'algèbre de processus et que les *opérations* ne peuvent pas être appliquées.

4.2.2.2 Objet composite

Un objet composite est le résultat d'une composition d'objets.

4.2.2.3 Décomposition

– **d'objets:** des objets peuvent être décomposés en ACT ONE sous réserve que des *opérations* existent dans la signature associée à l'objet pour que la décomposition soit possible. Par exemple, le *type de données* ci-après permet à un objet composite d'être décomposé en ses objets composants.

```
type Z is X, Y, IdType
    sorts Z
    opns makeZ: Id, X, Y -> Z
        getX: Z -> X
        getY: Z -> Y
    eqns forall x: X, y: Y, id: ID
        ofsort X
            getX (makeZ(id,x,y)) = x;
        ofsort Y
            getY (makeZ(id,x,y)) = y;
endtype (* Z *)
```

Par conséquent, étant donné Z: Z = *makeZ(id1,x,y)*, où *x* et *y* sont des instances de *sortes* modélisant des objets et *id1* est un identificateur unique, *z* peut être décomposé en *x* et *y*, c'est-à-dire en ses objets composants, respectivement par *getX(Z)* et *getY(Z)*.

NOTE – Dans cette interprétation, on suppose possible la subdivision d'un objet composite en ses objets composants. Toutefois, d'après le texte de la Rec. UIT-T X.902 | ISO/CEI 10746-2, il suffit que la décomposition spécifie un objet donné comme une combinaison de deux objets ou plus, c'est-à-dire une composition. En ACT ONE, les objets composites sont toujours spécifiés à partir de combinaisons d'objets composants. La distinction faite dans la Rec. UIT-T X.902 | ISO/CEI 10746-2 entre composition et décomposition est donc quelque peu floue en ACT ONE.

– **de comportements:** la notion de décomposition de comportements dépend de la spécification de la composition de comportements. Ce concept n'est pas explicitement prévu en ACT ONE (voir 4.2.2.1). En effet, les comportements sont représentés par des *opérations* et des *équations* agissant sur une *sorte*. Deux comportements de *sortes* arbitraires ne peuvent pas être combinés et déboucher sur un nouveau comportement.

NOTE – On pourrait aussi considérer que la notion de décomposition de comportements est fournie de manière inhérente par les *opérations* et *équations* ACT ONE associées à une *sorte*. En effet, ces *opérations* et *équations* permettent toutes les combinaisons possibles de comportements. Par conséquent, une composition séquentielle pourrait par exemple être générée au moyen d'*opérations* appliquées séquentiellement. Chaque *opération* de la séquence doit satisfaire aux *équations* nécessaires pour pouvoir se produire. Toutefois, le fait de parler de composition de comportements est discutable, car les *opérations* et *équations* existent déjà et définissent tous les comportements possibles.

4.2.2.4 Compatibilité de comportements

En LOTOS, l'équivalence de *types de données* sur le plan du comportement est basée sur l'équivalence de nom des *sortes* et éventuellement aussi sur la valeur associée à ces *sortes*. Par conséquent, un objet ne peut généralement pas en remplacer un autre dans un environnement donné si les objets proviennent de squelettes d'objet différents, c'est-à-dire si ce sont des instances de différentes *sortes*. Toutefois, il peut parfois être possible de remplacer un objet par un autre objet dérivé d'un squelette d'objet différent. Pour cela, l'environnement doit offrir des *opérations* qui soient applicables aux deux *sortes* et les résultats de ces *opérations* doivent être identiques. Par exemple, des *sortes* représentant une pile d'entiers et une file d'attente d'entiers peuvent être compatibles sur le plan du comportement dans un environnement donné si l'environnement n'offre qu'une *opération top* et si la file d'attente et la pile contiennent le même nombre d'entiers. En effet, dans les deux cas, le résultat sera un entier. Si un environnement offre une *opération pop* ou une *opération push*, la compatibilité de comportements n'existera pas entre les objets, car les *opérations* renvoient des *sortes* de pile et des *sortes* de file d'attente. Comme l'environnement d'un objet peut généralement invoquer n'importe quelle *opération* dans la signature, cette forme de compatibilité de comportements est limitée.

4.2.2.5 Affinement

La notion d'affinement étant explicitement prévue dans l'algèbre de processus du LOTOS, par exemple par des tests de *conformité* et des relations d'équivalence, on s'est très peu intéressé à l'affinement en ACT ONE. Intuitivement cependant, l'affinement en ACT ONE pourrait prendre de nombreuses formes, par exemple celle de l'extension de la signature d'une *sorte* donnée, c'est-à-dire la fourniture d'un plus grand nombre d'*opérations*. Avec cette forme d'affinement, une compatibilité de comportements naturelle devrait être générée; en effet, les *opérations* et *équations* existantes restent inchangées. D'autres formes d'affinement pourraient aussi être possibles, par exemple la modification des *équations* associées aux *opérations* sur une *sorte*. Dans ce cas, il ne devrait pas être facile de garantir la compatibilité de comportements.

4.2.2.6 Trace

Les interactions n'étant pas explicitement prévues en ACT ONE, la notion de trace est limitée, c'est-à-dire qu'il est impossible de garantir l'absence d'actions internes. Si on considère que les interactions sont les *opérations* qui se produisent dans l'algèbre de processus et que les actions internes sont les *opérations* servant à évaluer les *équations* associées à ces *opérations*, on peut alors modéliser une trace dans une faible mesure. Dans ce cas, elle correspond à la séquence d'*opérations* appliquées à une instance de *sorte* modélisant un objet. Il convient de signaler que si les *équations* associées aux *opérations* modélisant des interactions sont réécrites, l'enregistrement des interactions d'un objet, à savoir la trace, sera vraisemblablement incorrect. Par exemple, l'application de l'*opération push* puis de l'*opération pop* à une *file d'attente* fera vraisemblablement l'objet d'une réécriture sous la forme: *file d'attente* soumise à l'expression *pop(push(x,q))*. La notion de trace est donc limitée en ACT ONE.

4.2.2.7 Type d'un <X>

Les objets, interfaces et actions spécifiés en ACT ONE peuvent satisfaire de nombreux prédicats de caractérisation arbitraires différents. Les types pouvant être consignés explicitement sont des types de squelette.

4.2.2.8 Classe d'un <X>

La notion de classe dépend du prédicat de type de caractérisation auquel satisfont les membres de la classe. Les objets, interfaces et actions peuvent satisfaire de nombreux prédicats de type de caractérisation arbitraires. Un type pouvant être consigné est un type de squelette. Lorsque c'est le cas, la classe d'objets, d'interfaces ou d'actions associée à ce type est la classe de squelette.

NOTE – Il convient de noter que, si on affirme qu'en LOTOS, on ne peut classer les objets, interfaces et actions que par rapport à leur type de squelette, les concepts de classe et de classe de squelette donnés dans la Rec. UIT-T X.902 | ISO/CEI 10746-2 en sont réduits à la même technique de modélisation en LOTOS. Il n'existe donc pas de distinction en LOTOS entre une classe dans son sens de classification général et une classe de squelette dans son sens plus restrictif d'ensemble d'instances d'un type de squelette donné.

4.2.2.9 Sous-type/supertype

Les notions de sous-type et de supertype ne sont généralement pas prises en charge en ACT ONE car LOTOS utilise l'équivalence de nom pour la vérification de type. Par exemple, deux types d'objet ne sont identiques que lorsqu'ils sont représentés par la même *sorte*. Par conséquent, une *sorte* ne peut généralement pas être remplacée par une autre *sorte*. Toutefois, une forme limitée de sous-typage pourrait exister entre deux *sortes* différentes si le prédicat de type de caractérisation est basé sur un aspect des *sortes* autres que leur nom, par exemple: cette *opération* est valable sur cette *sorte* et renvoie ce résultat. Il s'agit toutefois d'une forme limitée de typage qui n'existera vraisemblablement pas dans la plupart des cas.

4.2.2.10 Sous-classe/superclasse

Les notions de sous-type et de supertype ne sont prises en charge que dans une très faible mesure en ACT ONE, à savoir lorsque le prédicat de type de caractérisation est basé sur un certain aspect de la *sorte*. En conséquence, les notions de sous-classe et de superclasse ne sont pas entièrement prises en charge en ACT ONE. Si une relation de sous-type/ supertype existe entre deux *sortes*, une relation de sous-classe/superclasse existe aussi entre les instances des *sortes* dans l'algèbre de processus.

4.2.2.11 Squelette de <X>

- **Squelette d'objet:** il s'agit de la définition d'une *sorte* et des *opérations* et *équations* associées modélisant un objet.

- **Squelette d'interface:** c'est l'ensemble des *opérations* et *équations* associées à une définition de *sorte* modélisant un objet. Il convient de noter que les notions de squelettes d'interface et d'objet peuvent être considérées comme identiques puisque les *opérations* et *équations* doivent toujours agir sur une définition de *sorte*. Par ailleurs, les *opérations* et *équations* spécifiées dans la partie ACT ONE de la spécification sont implicitement associées à la déclaration d'une instance de *sorte* dans l'algèbre de processus.

- **Squelette d'action:** il s'agit d'une *opération* et des *équations* qui lui sont associées.

4.2.2.12 Signature d'interface

La signature d'interface est constituée par les *opérations* qui s'appliquent à une variable déclarée comme une instance de *sorte* modélisant un objet.

4.2.2.13 Instanciation (d'un squelette de < *X* >)

- **d'un squelette d'objet:** pour instancier un squelette d'objet, il faut initialiser dans un état initial valide une *sorte* modélisant un objet. Cette initialisation doit permettre de garantir que l'instance de *sorte* peut être identifiée de manière univoque.

- **d'un squelette d'interface:** un squelette d'interface est instancié lorsqu'un squelette d'objet est instancié. En tant que tel, un objet a une seule interface donnée par les *opérations* et *équations* agissant sur la *sorte* à partir de laquelle l'objet est instancié.

- **d'un squelette d'action:** il s'agit de l'occurrence d'une *opération* ACT ONE dans l'algèbre de processus. Cette *opération* doit satisfaire aux *équations* qui lui sont associées.

4.2.2.14 Rôle

La meilleure façon de modéliser la notion de rôle en ACT ONE est d'utiliser une *sorte*, car un rôle représente un identificateur de comportement. En effet, la déclaration d'une *sorte* permet de rendre accessibles les *opérations* et *équations* qui s'appliquent à cette *sorte*.

4.2.2.15 Création (d'un < *X* >)

- **d'un objet/d'une interface:** étant donné que les objets et les interfaces n'ont une forme d'existence que lorsqu'ACT ONE est utilisé conjointement avec l'algèbre de processus, ACT ONE ne peut pas être utilisé seul pour modéliser la création. ACT ONE conjointement avec l'algèbre de processus peuvent servir à modéliser la création d'objets et d'interfaces dans une faible mesure, sous réserve qu'un certain style de spécification soit suivi; par exemple, une *opération* associée à une *sorte* modélisant un objet, c'est-à-dire une *opération* sur un objet existant, qui entraîne la génération d'un nouvel objet. Il devrait être possible d'identifier de manière univoque l'objet nouvellement généré. Ce nouvel objet devrait aussi être utilisé dans l'algèbre de processus de manière à avoir une certaine forme d'existence (voir 4.2.1.1 pour de plus amples détails sur la façon de réaliser cela.)

4.2.2.16 Introduction (d'un objet)

L'introduction d'un objet peut se faire de diverses manières en ACT ONE lorsque ce langage est utilisé conjointement avec l'algèbre de processus; par exemple, par l'occurrence d'offres d'événements dont les *dénotations d'action* conduisent à une nouvelle instance de *sorte* modélisant un objet généré. Ces nouvelles instances doivent être dans un état initial valide, on doit pouvoir les identifier de manière univoque et elles doivent avoir une certaine forme d'existence dans l'algèbre de processus. Autre solution: les objets peuvent être introduits par des clauses **let ... in**. Ici aussi, les objets doivent avoir un état initial valide, être utilisés dans l'algèbre de processus de manière à avoir une forme d'existence et il doit être possible de les identifier de manière univoque.

4.2.2.17 Suppression (d'un < *X* >)

- **d'un objet/d'une interface:** la suppression d'un objet ou d'une interface en ACT ONE – lorsque ce langage est utilisé conjointement avec l'algèbre de processus – peut se faire par la réécriture qui se produit avec les *équations* associées aux *opérations* sur les *sortes* modélisant des objets. Par exemple, une *opération* qui supprime un élément d'un ensemble peut servir à modéliser la suppression, un objet possédant un certain identificateur peut par exemple être supprimé de l'ensemble des objets instanciés qui existent dans la *liste de paramètres de valeurs* d'une *définition de processus* récurrente.

4.2.2.18 Instance d'un type

- **de squelette d'objet/d'interface/d'action:** une instance de type dépend du prédicat de caractérisation définissant le type. Si le prédicat de type correspond au type de squelette pour les objets et les interfaces, une instance de type d'objet ou d'interface correspond à une occurrence de la *sorte* modélisant les objets et les interfaces considérés dans l'algèbre de processus. De même, si le prédicat de type correspond au type de squelette pour les actions, une instance de type d'action est donnée par l'occurrence d'une *opération* modélisant le type d'action considéré dans l'algèbre de processus.

4.2.2.19 Type de squelette (d'un < *X* >)

- **d'un objet/d'une interface:** prédicat sur les instanciations de la *sorte* utilisée pour modéliser un objet. Toutes les instanciations (occurrences) du squelette (*sorte*) dans l'algèbre de processus comportent les *opérations* et *équations* qui sont associées à cette *sorte*. Etant donné que les squelettes d'objet et d'interface sont modélisés de la même manière, à savoir par la définition d'une *sorte* et par les *opérations* et *équations* associées, le type de squelette d'un objet et le type de squelette de l'interface avec cet objet sont des synonymes en ACT ONE. En effet, ils correspondent tous deux à l'occurrence d'une *sorte* dans l'algèbre de processus.
- **d'une action:** prédicat sur les occurrences d'*opération* dans l'algèbre de processus. En effet, toutes les instanciations (occurrences) du squelette (*opération*) dans l'algèbre de processus doivent respecter les spécifications du squelette – c'est-à-dire qu'elles doivent avoir les entrées et produire les résultats donnés dans la définition d'*opération* – et l'évaluation de l'*opération* est fonction des *équations* associées à cette *opération.*

4.2.2.20 Classe de squelette (d'un < *X* >)

- **d'un objet:** c'est l'ensemble des instances d'une *sorte* donnée modélisant un objet dans l'algèbre de processus.
- **d'une interface:** c'est l'ensemble des instances d'une *sorte* donnée modélisant une interface avec un objet dans l'algèbre de processus.
- **d'une action:** c'est l'ensemble des instances d'une *opération* donnée dans l'algèbre de processus.

4.2.2.21 Classe dérivée/classe de base

Les classes dérivées et les classes de base ne sont pas prises en charge en ACT ONE, car dans ce langage, les classes ne sont normalement données que par des classes de squelette, c'est-à-dire, pour un objet, par l'ensemble des instances d'une *sorte* donnée modélisant un objet dans l'algèbre de processus. Les *sortes* ne peuvent pas être modifiées de manière incrémentielle. En effet, les *sortes* et les étiquettes qui leur sont rattachées – c'est-à-dire le nom de la *sorte* – ne permettent pas de faire référence à une autre *sorte*, c'est-à-dire que l'autoréférence existe toujours. Les *opérations* et *équations* associées à une *sorte* donnée ne sont donc applicables qu'à cette *sorte* et à aucune autre.

Il convient aussi de signaler que la notion d'*actualisation* de classes paramétrées, tout en présentant intuitivement les éléments de la relation classes dérivées/classes de base, ne représente pas une telle relation en réalité, car les instances d'une classe paramétrée ne peuvent pas se produire dans l'algèbre de processus, c'est-à-dire qu'elles doivent être *actualisées* pour qu'une classe puisse exister.

4.2.2.22 Invariant

Cette notion est implicite en ACT ONE, c'est-à-dire que les objets doivent toujours satisfaire aux *opérations* et *équations* qui s'appliquent à ces objets.

4.2.2.23 Précondition

En ACT ONE, toutes les *opérations* doivent satisfaire à toutes les *équations* (et aux éventuels *gardes* associés) qui s'appliquent à ces opérations pour qu'elle puissent se produire.

4.2.2.24 Postcondition

Cette notion est implicite en ACT ONE, c'est-à-dire que pour qu'une *opération* (action) donnée se produise, il faut que les *équations* associées soient définies (et soient vraies).

4.3 Sémantique architecturale en SDL-92

Le SDL-92 est un langage FSL normalisé. Le tutorial figure en annexe à la Rec. UIT-T Z.100. Un certain nombre d'ouvrages existent sur le SDL ainsi que des outils commerciaux, qui prennent en charge différents aspects du SDL depuis le traitement des graphiques jusqu'à l'analyse et la production d'un code de programmation basé sur le SDL.

En SDL, un *système* est modélisé sous la forme d'un ensemble de machines à états finis étendues communiquant par messages appelés *signaux*. Le concept de données du SDL est basé sur ACT ONE. Les machines à états sont étendues dans le sens où elles peuvent définir des variables locales permettant de garder une partie de leur historique. Les signaux sont transmis de manière asynchrone, ce qui permet un couplage lâche entre les composants d'un système réparti.

Le présent paragraphe expose une façon dont les concepts de modélisation peuvent être exprimés en SDL. La représentation n'est pas considérée comme unique. Toutefois, elle montre que presque tous les concepts fondamentaux peuvent être exprimés en SDL. Il convient de signaler qu'il existe une autre méthode – basée sur l'utilisation de ACT ONE – permettant de modéliser de nombreux concepts parmi ceux qui sont donnés ici. Des renseignements sur cette méthode figurent au 4.2.

On utilise la version SDL-92, définie dans la Rec. UIT-T Z.100. Le SDL-92 contient un certain nombre d'extensions par rapport au SDL-88. Les plus importantes sont les suivantes:

- constructions orientées objet;
- possibilité de *canaux* sans retard;
- non-déterminisme;
- inclusion possible d'autres concepts de données et appels de *procédure distante*.

L'élément le plus important du typage d'autres données est qu'il est possible d'utiliser de manière combinée ACT ONE et l'ASN.1 avec le SDL. La sémantique de la combinaison du SDL-92 avec l'ASN.1 est définie dans la Rec. UIT-T Z.105.

Afin d'éviter toute confusion, les concepts SDL ont été mis en *italique*, chaque fois que cela a été jugé nécessaire.

4.3.1 Concepts de modélisation de base

4.3.1.1 Objet

En SDL, les objets sont des instances de *type de système*, de *type de bloc*, de *type de processus*, de *type de service*, de *temporisateur*, de *canal* et de *route de signaux*. Ces instances sont caractérisées par un état et un comportement.

Chaque instance est encapsulée, c'est-à-dire que son état ne peut être modifié qu'à la suite d'une action interne ou d'une interaction avec son environnement.

Pour certains types d'objets, des références doivent être fournies explicitement.

4.3.1.2 Environnement (d'un objet)

L'environnement d'un objet dépend du type d'objet. Voir le Tableau 1.

4.3.1.3 Action

En SDL, une action est une *entrée* ou une *mise en réserve* simple, une *instruction d'action* simple, une *transition* complète ou l'exécution complète d'une *procédure*. Citons des *instructions d'action* simples:

- *tâche, importation, exportation, visualisation;*
- *sortie;*
- *création;*
- *initialisation, réinitialisation, actif;*
- *appel de procédure;*
- *arrêt/retour;*
- *état suivant.*

Tableau 1 – Environnement des objets

Type d'objet	Contraintes concernant l'environnement
système	– *signaux* entrants de canaux
bloc	– types de données globaux
	– *signaux* entrants de canaux
	– appels de *procédures exportées*
	– variables importées
processus	– types de données globaux
	– *signaux* entrants de *routes de signaux* implicites ou explicites
	– appels de *procédures exportées*
	– variables visualisées/importées
	– contraintes temporelles pour les actions d'*entrée*
service	– variables globales/*temporisateurs*/types de données, tampon de signaux partagé (possédé par l'instance de *processus* englobante)
	– *signaux* entrants de *routes de signaux* implicites ou explicites
	– appels de *procédures exportées*
	– variables visualisées/importées
	– contraintes temporelles pour les actions d'*entrée*
temporisateur	– appels d'*initialisation* et de *réinitialisation, arrêt* du *processus* propriétaire
canal	– *signaux* entrants provenant des *blocs* connectés (ou de l'environnement du *système*)
route de signaux	– *signaux* entrants provenant des instances de *processus/service* connectées

La transmission d'un *signal* par un *canal* ou par une *route de signaux* est aussi une action, tout comme la génération d'un *signal de temporisateur*. Les interactions correspondent aux *entrées/sorties* d'un *signal*, aux actions d'*appel* et de *retour* d'une *procédure distante* et à l'utilisation de variables partagées (variables de *processus* globales – *révélées/visualisées* – associées aux *services* et *importation/exportation* de variables de *processus*). La séquence d'actions consistant à envoyer, acheminer et éventuellement recevoir un *signal (sortie-entrée)* peut être considérée comme une interaction.

4.3.1.4 Interface

Suivant le type d'objet, les interfaces peuvent être décrites de différentes façons en SDL (voir le Tableau 2).

Dans le cas d'un objet *bloc*, l'ensemble des *procédures distantes exportées/importées* vers/depuis l'extérieur du *bloc* ainsi que l'ensemble des *signaux* envoyés/reçus par les *processus* de ce *bloc* doivent être encapsulés dans un ou plusieurs processus. Ces *processus* jouent alors le rôle d'interfaces pour l'objet *bloc*. Avec ces descriptions d'interface, seules les interactions potentielles d'un objet sont définies, où chaque interface correspond à la description d'un sous-ensemble des interactions potentielles d'un objet.

Tableau 2 – Interfaces avec les objets

Type d'objet	L'interface est caractérisée par
système	– les *portes du système* et leurs listes de *signaux*
	– les *canaux* entrants/sortants et leurs listes de *signaux*
bloc	– les *portes du bloc* et leurs listes de *signaux*
	– les *canaux* entrants/sortants et leurs listes de *signaux*
	– les *routes de signaux* entrantes/sortantes et leurs listes de *signaux*
	– l'ensemble des *procédures distantes (exportées/importées* vers/depuis l'extérieur du *bloc)*
	– l'ensemble des variables distantes *(exportées/importées* vers/depuis l'extérieur du *bloc)*
processus	– les *portes du processus* et leurs listes de *signaux*
	– l'ensemble de tous les *signaux d'entrée/sortie* valides
	– l'ensemble de toutes les *procédures exportées/importées*
	– l'ensemble des variables partagées *(révélées/visualisées* vers/depuis l'extérieur du *processus)*
service	– les *portes du service* et leurs listes de *signaux*
	– l'ensemble de tous les *signaux d'entrée/sortie* valides
	– l'ensemble de toutes les *procédures exportées/importées*
temporisateur	– l'identification du *temporisateur*
canal	– les ensembles de tous les *signaux* transportés dans chaque sens
route de signaux	– les ensembles de tous les *signaux* transportés dans chaque sens

4.3.1.5 Activité

Une activité ne peut généralement pas être indiquée explicitement, car elle peut couvrir plusieurs objets.

Citons le cas particulier d'activité suivant: l'exécution d'une *procédure distante* ou locale, l'action d'*appel* étant l'action de tête de l'activité et les actions de retour potentielles étant les actions de queue.

4.3.1.6 Comportement (d'un objet)

Le comportement d'un *processus/service* est l'ensemble de toutes les transitions de ce *processus/service*. Les actions d'*entrée* imposent des contraintes quant aux circonstances dans lesquelles les transitions peuvent avoir lieu. On peut introduire d'autres contraintes au moyen de la construction *provide* ou *signal continu*. Un objet peut présenter un comportement non déterministe.

Le comportement d'un *bloc* est constitué à partir du comportement des *processus* contenus dans ce *bloc*. Le comportement d'un *système* est constitué à partir du comportement des *blocs* contenus dans ce *système.*

Le comportement d'un *canal* ou d'une *route de signaux* correspond à l'acheminement des *signaux* (instantané ou retardé, lorsque c'est spécifié).

Le comportement d'un *temporisateur* est prédéfini en SDL sous forme d'une horloge à alarme.

4.3.1.7 Etat (d'un objet)

L'ensemble de toutes les séquences d'actions auxquelles un objet peut participer à un instant donné est déterminé, dans le cas d'un *processus/service*, par l'état SDL courant à cet instant, les valeurs des variables locales et le contenu du port d'entrée.

L'état d'un *bloc* ou d'un *système* est donné par l'ensemble des états de tous les *processus, blocs* qu'il contient et de tous les *canaux* ou *routes de signaux* qu'il contient.

L'état d'un *canal* est donné implicitement ou explicitement par un *bloc*. L'état dépend du fait que le *canal* ait une propriété de *retard* ou pas.

L'état d'une *route de signaux* est toujours donné implicitement.

L'état d'un *temporisateur* est actif ou inactif. L'état d'un temporisateur actif est déterminé par la durée restante au bout de laquelle un *signal* de fin de temporisation est envoyé.

4.3.1.8 Communication

L'acheminement d'information entre deux objets ou plus se fait par des *canaux* ou des *routes de signaux* explicites ou implicites (en cas de *procédures* distantes ou de variables *importées/exportées*). L'information est transportée par des *signaux.*

4.3.1.9 Position dans l'espace

Les actions ont lieu dans des instances de *processus* et dans des instances de *service*. Les actions de transmission ont lieu dans des *canaux* ou dans des *routes de signaux.*

4.3.1.10 Position dans le temps

Chaque action est caractérisée par une date de début de l'action et une date de fin de l'action. Les actions peuvent être instantanées. La durée d'une action ne peut pas être indiquée explicitement. Les actions peuvent être programmées pour une certaine date ou après un certain délai.

NOTE –

a) Il existe un temps absolu en SDL accessible par l'expression *now*. Aucune précision n'est donnée concernant les unités de temps.

b) Pour deux actions consécutives *a1* et *a2*, on a la relation *now(a1) <= now(a2).*

c) Pour accéder explicitement au temps, on ne peut qu'utiliser l'action d'*initialisation* pour un *temporisateur* ou appliquer l'expression *now* dans les *conditions de validation* et les *signaux continus*. Il convient d'éviter de programmer des actions pour un instant fixe.

4.3.1.11 Point d'interaction

Les points d'interaction correspondent aux portes des instances de *bloc/processus/service* et aux points d'extrémité des *canaux* et *routes de signaux* (éventuellement implicites). Les variables partagées sont aussi des points d'interaction. Les points d'interaction ont une position. Un objet peut avoir plusieurs points d'interaction.

4.3.2 Concepts de spécification

4.3.2.1 Composition

- **d'objets**
  a) un objet *système* peut être une composition parallèle d'objets *bloc*, qui peuvent être interconnectés par des *canaux*;
  b) un objet *bloc* peut être une composition parallèle d'objets *bloc* (qui peuvent être interconnectés par des *canaux*) ou une composition parallèle d'objets *processus*, qui peuvent être connectés par des *routes de signaux*;
  c) un objet *processus* peut être une composition d'objets *service* avec entrelacement;
  d) un *canal* peut être composé de *blocs* interconnectés par des *canaux*.
- **de comportements**
  a) le comportement d'un *système* est une composition parallèle du comportement de ses *blocs*;
  b) le comportement d'un *bloc* est une composition parallèle du comportement de ses *sous-blocs* ou de ses *processus*;
  c) le comportement d'un *processus* est une composition du comportement de ses *services* avec entrelacement ou une composition séquentielle des actions de son graphe de *processus*;
  d) le comportement d'un *service* est une composition séquentielle des actions de son graphe de *service*;
  e) le comportement d'un *canal* peut être composé du comportement des *blocs* qui le composent et des *canaux* qui les interconnectent.

4.3.2.2 Objet composite

D'après le 4.3.2.1, les objets suivants peuvent être exprimés sous forme de composition:

- *système;*
- *bloc;*
- *processus;*
- *canal.*

4.3.2.3 Décomposition

- **d'objets:** c'est la spécification d'un objet donné sous forme de composition.
- **de comportements:** c'est la spécification d'un comportement donné sous forme de composition.

4.3.2.4 Compatibilité de comportements

Il n'existe pas de méthode générale permettant de décrire explicitement la compatibilité de comportements en SDL, toutefois la base sémantique du langage en termes de systèmes de transition permet de définir la compatibilité de comportements et de la vérifier.

On peut considérer qu'une instance de classe dérivée présente une compatibilité de comportement restreinte avec une instance de la classe de base correspondante et qu'une instance de type *redéfini* présente une compatibilité de comportement restreinte avec une instance du type *virtuel* correspondant. On peut recourir à des clauses *atleast* pour exiger une compatibilité de comportement restreinte.

4.3.2.5 Affinement

Il existe deux moyens pour affiner une spécification SDL d'un objet:

- sous-structuration (au niveau du *bloc* ou du *système*);
- utilisation des éléments orientés objet (héritage, virtualité et paramètres génériques).

4.3.2.6 Trace

Le SDL ne permet pas de spécifier explicitement les traces. Celles-ci peuvent être obtenues comme résultat de l'interprétation d'une spécification SDL conformément à la sémantique dynamique du SDL.

NOTE – Les diagrammes de séquences de messages (MSC, *message sequence charts*) fournissent une syntaxe et une sémantique appropriées pour la représentation des traces des spécifications SDL. Il existe une relation étroite entre d'une part la syntaxe et la sémantique des diagrammes MSC et d'autre part la syntaxe et la sémantique du SDL. Les diagrammes MSC sont définis et normalisés dans la Rec. UIT-T Z.120.

4.3.2.7 Type (d'un <X>)

Il n'existe pas de prédicat explicite général en SDL.

4.3.2.8 Classe (de <X>)

Ce concept n'est généralement pris en charge que pour les types de squelette.

4.3.2.9 Sous-type/supertype

Ce concept n'est généralement pas pris en charge.

4.3.2.10 Sous-classe/superclasse

Ce concept n'est généralement pris en charge que pour les types de squelette.

4.3.2.11 Squelette de <X>

- **Squelette d'objet:** les squelettes d'objet sont des définitions de type correspondant au type d'objet approprié *(système, bloc, processus, service)*. Pour les *temporisateurs*, les *canaux* et les *routes de signaux*, les squelettes d'objet sont les déclarations correspondantes.
- **Squelette d'interface:** suivant le type d'interface, un squelette d'interface peut être donné implicitement par une déclaration *(canal, routes de signaux)* ou explicitement par une définition de type correspondant à un *processus* (voir 4.3.1.4).
- **Squelette d'action:** un squelette d'action est spécifié par la définition d'un graphe de *processus/procédure/service*. Les squelettes d'actions atomiques sont les suivants: *entrée, sortie, mise en réserve, initialisation, réinitialisation, création, tâche, arrêt, retour, état suivant, appel, importation, exportation, visualisation.*

Les squelettes peuvent être spécifiés au moyen de paramètres (paramètres formels ou paramètres de contexte formel). Des contraintes peuvent être associées aux paramètres. Les squelettes peuvent être combinés (c'est-à-dire qu'une définition de type peut contenir d'autres définitions de type).

4.3.2.12 Signature d'interface

Il s'agit de l'ensemble des *types de signaux* et des *types de procédures distantes* qui sont valables pour l'interface.

4.3.2.13 Instanciation (d'un squelette de <X>)

En SDL, il existe deux moyens permettant d'instancier les squelettes:

- instanciation implicite *(système, bloc, canaux, routes de signaux, processus, services)* qui se fait par déclaration d'objet;
- instanciation explicite au moyen de l'action de *création* (uniquement pour les *processus*).

Les instanciations sont toujours le résultat de l'instanciation d'un squelette par une action. Les paramètres de contexte formel doivent être actualisés avant que l'instanciation puisse se produire (par la spécialisation d'un type de *processus* ou par la déclaration d'un *processus*).

4.3.2.14 Rôle

Il n'existe pas de moyen général permettant de spécifier les rôles.

Les rôles peuvent être décrits sous forme de paramètres de contexte formel.

NOTE – On peut utiliser les clauses *atleast* pour qualifier un rôle avec plus de détails.

4.3.2.15 Création (d'un <X>)

Il existe deux types de création (voir 4.3.2.13):

- instanciation implicite;
- instanciation explicite – interprétation d'une action de *création*.

4.3.2.16 Suppression (d'un <X>)

Seuls les objets *processus* peuvent être supprimés. Un *processus* ne peut supprimer que lui-même, ce qu'il fait par l'interprétation de l'action *arrêt*. Si un *service* interprète une action *arrêt*, ce *service* est alors supprimé, de même que tous les autres *services* appartenant au même *processus* ainsi que le *processus.*

NOTE –

a) La suppression d'un *processus* par un autre *processus* peut être modélisée au moyen de la *sortie* d'un *signal* spécial dont la consommation par le récepteur provoque l'interprétation d'un *arrêt* par le récepteur.

b) La suppression de tous les *processus* d'un *bloc* peut être considérée comme la suppression de ce *bloc.*

4.3.2.17 Introduction (d'un objet)

L'instanciation implicite (voir 4.3.2.13) peut être considérée comme une introduction.

4.3.2.18 Instance d'un type

Un objet est une instance de *type système*, de *type bloc*, de *type processus* or de *type service* X, s'il existe une instanciation explicite ou implicite de cet X ou un substitut de X. Un substitut est une instance d'un squelette de type qui est une spécialisation de type en SDL.

4.3.2.19 Type de squelette (d'un <X>)

Le fait qu'un <X> est une instanciation d'un squelette de <X> peut être exprimé pour les *processus*, *services*, *blocs* et *systèmes* par l'indication que l'objet est une instance SDL de la définition du type.

4.3.2.20 Classe de squelette (d'un <X>)

Il n'existe pas de notation explicite générale pour caractériser la classe de squelette d'un <X>, toutefois la classe de squelette est l'ensemble de toutes les instances de *processus*, *blocs*, *services* ou *systèmes* provenant de la définition, d'un *type de processus*, d'un *type de bloc*, d'un *type de service* ou d'un *type de système* respectivement.

4.3.2.21 Classe dérivée/classe de base

Une définition de type peut être dérivée d'une autre définition de type par *spécialisation*, qui peut consister:

- en une liaison des paramètres de contexte et en une adjonction de nouveaux paramètres de contexte;
- en un héritage de définitions;
- en une redéfinition de composants virtuels;
- en une adjonction d'autres définitions.

Des contraintes peuvent être appliquées au moyen des *constructions atleast* et *finalized*.

NOTE – L'héritage multiple n'est pas pris en charge.

4.3.2.22 Invariant

Il n'existe pas de notation pour les invariants en SDL.

4.3.2.23 Précondition

On peut utiliser des *conditions de validation*, des *signaux continus* et des *signaux* pour spécifier les préconditions d'une transition.

4.3.2.24 Postcondition

Il n'existe pas de notation pour les postconditions en SDL.

4.4 Sémantique architecturale en Z

La notation Z est une notation de spécification basée sur une théorie des ensembles comportant de nombreux types et sur un calcul de prédicat de premier ordre. Z n'est pas encore un langage FSL ISO mais une norme est en cours d'élaboration au sein de l'ISO SC22 WGl9. La dernière version de la norme figure dans "The Z-Base Standard". Voir l'article 2.

Il existe une norme de facto pour Z ("The Z Notation" de Spivey). Voir l'article 4. Cette version de la notation est stable et très proche de la norme "The Z-Base Standard"; des outils permettent de prendre en charge la vérification de la syntaxe et de la sémantique statique. Cette version de la notation sera utilisée pour la sémantique architecturale du modèle RM-ODP tant qu'une version ISO ne sera pas largement disponible.

Afin d'éviter toute confusion dans les terminologies ODP et Z, les termes propres à la notation Z sont mis en italique dans les paragraphes qui suivent.

4.4.1 Concepts de modélisation de base

4.4.1.1 Objet

En Z, un objet peut être décrit par un ensemble de fragments de spécification. Ces fragments doivent contenir un ensemble de *schémas d'opération* (représentant l'interface avec l'objet) faisant référence à un ou plusieurs *schémas d'état* (représentant l'état de l'objet). Les fragments de spécification doivent pouvoir être identifiés de manière univoque (il est question ici de l'identité de l'objet). Un fragment peut être identifié au moyen d'un identificateur figurant dans le ou les *schémas d'état* de l'objet, cet identificateur restant constant dans toutes les opérations définies pour cet objet. Enfin, il doit exister un état initial valide pour cet objet. Cet état peut être déterminé par un schéma d'initialisation qui fournit des liaisons légales avec les variables déclarées dans le *schéma d'état* ainsi qu'un prédicat garantissant que l'objet est unique dans la spécification.

NOTE – Il faut faire attention au moment de la spécification d'objets en Z, car le langage Z ne possède pas d'éléments d'encapsulation (essentiels pour la description d'objets), comme il est dit dans la Note en 4.4.1.3.

4.4.1.2 Environnement (d'un objet)

Dans une spécification Z, l'environnement d'un objet est décrit en termes de ses entrée et sortie. L'entrée d'un objet vient de l'environnement. La sortie d'un objet va vers l'environnement. L'environnement d'un objet peut être spécifié directement ou ne pas être spécifié. S'il n'est pas spécifié, des *schémas d'opération* produisant des sorties ou exigeant des entrées peuvent avoir lieu, l'environnement de la spécification tout entier fournissant les entrées ou recevant les sorties. Toutefois, si l'environnement d'un objet est spécifié, alors pour chaque *schéma d'opération* associé à l'objet, il existe un autre *schéma d'opération* (éventuellement associé à un autre objet) qui exige des entrées ou des sorties du même type que celles de l'objet considéré. Ces deux *schémas d'opération* sont alors réunis, les entrées et les sorties de l'opération considérée devenant alors respectivement les sorties et les entrées de l'opération représentant l'environnement.

L'environnement d'un objet peut aussi être donné par des variables auxquelles un objet fait référence et qui ont un domaine d'application global, par exemple les variables figurant dans les *descriptions axiomatiques.*

4.4.1.3 Action

En Z, une action est modélisée par l'exécution d'une opération spécifiée dans un *schéma d'opération*. Le résultat en est un changement instantané (ou aucun changement) de l'état des objets auxquels l'action est associée. Une action peut produire un résultat non déterministe.

Comme il n'existe pas de notation explicite pour l'encapsulation en Z, il est inhabituel de déterminer si une action est observable ou interne en Z, la distinction entre interaction et action interne n'est donc pas clairement définie. La présente Recommandation | Norme internationale utilise la convention suivante: un *schéma d'opération* représentant une action qui a des entrées, des sorties ou des variables globales sur le plan de la spécification interagit avec son environnement. L'environnement peut être spécifié ou non (voir 4.4.1.2). Les actions qui exigent des entrées venant d'un environnement non spécifié et qui ne produisent pas de sortie peuvent être considérées comme des actions non observables invoquées en externe. Les actions produisant des sorties vers un environnement non spécifié peuvent être considérées comme des actions (spontanées) observables invoquées en interne. Les actions qui exigent des entrées venant d'un environnement non spécifié et qui produisent des sorties vers cet environnement peuvent être considérées comme des actions observables invoquées en externe.

Toutefois, si l'environnement d'un objet est spécifié, alors pour chaque *schéma d'opération* qui exige des entrées ou des sorties et qui est associé à une interface avec un objet, c'est-à-dire pour chaque action observable, il existe un autre *schéma d'opération* (éventuellement associé à un autre objet) qui exige des entrées ou des sorties du même type que celles de l'objet considéré. Ces deux *schémas d'opération* sont alors réunis, les entrées et les sorties de l'opération considérée devenant respectivement les sorties et les entrées de l'opération représentant l'environnement.

Par ailleurs, l'occurrence de chacune des opérations qui font référence à des variables qui sont globales sur le plan de la spécification peut être considérée comme une interaction.

En Z, toutes les opérations sont atomiques, ce qui signifie que les *schémas d'opération* se produisent en totalité ou ne se produisent pas du tout. Les actions sont donc forcément atomiques en Z.

Un objet interagissant avec lui-même peut être modélisé de manière non formelle par la composition de *schémas d'opération Z*. Par exemple, l'opération OpA avec la sortie *a!: A* peut être composée avec l'opération OpB, avec l'entrée *b?: A* et une *conjonction* de prédicat peut être ajoutée pour préciser que *a! = b?*.

La notion de relation de cause à effet n'entre pas strictement dans le cadre de la notation Z. Toutefois, si une opération exige qu'une entrée ait lieu, on pourrait considérer que c'est l'environnement qui provoque l'occurrence de cette opération, c'est-à-dire que l'environnement joue le rôle de producteur et le *schéma d'opération* celui de consommateur. De même, si un *schéma d'opération* produit une sortie, on pourrait considérer que l'environnement joue le rôle de consommateur et l'opération celui de producteur. Si un *schéma d'opération* donné exige des entrées et donne des sorties, ou n'a ni entrée ni sortie, il est impossible d'associer une relation de cause à effet à cette action particulière.

NOTE – Il convient de noter que cette convention syntaxique pour la distinction entre action interne et action observable est limitée car il n'y a pas de distinction sémantique entre les opérations devant être interprétées comme spontanées ou internes et celles qui exigent une participation de l'environnement; cela doit être précisé dans les observations en langage naturel qui doivent accompagner toutes les spécifications Z. En conséquence, d'après la définition ci-dessus, une file d'attente "Lossy" est traitée comme étant un sous-type de file d'attente. Cependant, il est clair que lorsque l'opération "Lose" est appliquée dans une file d'attente "Lossy", le but est que cette opération se produise de manière non déterministe.

4.4.1.4 Interface

Il s'agit d'une abstraction du comportement d'un objet obtenue par identification des opérations associées à cet objet qui doivent constituer la substance de l'interface. Dans tous les *schémas d'opération* restants, toutes les entrées et sorties sont cachées et l'occurrence de chacune des opérations définies dans ces *schémas d'opération* est considérée comme une action interne, c'est-à-dire qu'elle n'exige pas la participation de l'environnement de l'objet ou qu'elle ne fait pas participer cet environnement. Le texte Z résultant qui représente cet objet est un squelette d'interface. Toute instance du squelette d'interface est une interface.

4.4.1.5 Activité

La notion d'activité définie comme un graphe d'actions acyclique orienté à racine unique n'existe pas directement dans le langage Z. Toutefois, on peut modéliser le concept d'activité dans une certaine mesure en notant que si l'action *x* précède l'action *y* dans une certaine activité, alors la *postcondition* de l'action *x* doit déterminer la *précondition* pour l'action *y*.

4.4.1.6 Comportement (d'un objet)

Le comportement d'un objet dans un état donné correspond à l'ensemble de toutes les activités possibles qui peuvent se produire à partir de cet état. L'environnement de l'objet et les contraintes exprimées dans les *préconditions* peuvent avoir une incidence sur la séquence effective des actions qui peuvent se produire.

4.4.1.7 Etat (d'un objet)

Il s'agit d'une liaison des variables d'état déclarées dans le ou les *schémas d'état* associés au squelette d'objet utilisé dans le calcul des *préconditions*.

4.4.1.8 Communication

En Z, la communication peut être modélisée au moyen d'entrées et de sorties par rapport à des opérations. Les entrées et les sorties par rapport aux *schémas d'opération* sont normalement considérées comme des communications avec l'environnement d'un objet. Comme la communication a lieu entre objets, l'environnement d'un objet (voir 4.4.1.2) doit être spécifié pour modéliser la communication. Pour obtenir une communication, il faut donc d'abord normaliser les *schémas d'opération* associés aux objets en interaction puis les réunir, les sorties d'une opération devenant les entrées de l'autre *schéma d'opération*. Pour cette modélisation de la communication, les entrées et sorties des *schémas d'opération* associés doivent être du même type.

Par ailleurs, l'occurrence de chacun des *schémas d'opération* qui font référence à des variables globales sur le plan de la spécification représente une communication, la valeur de la variable globale résultant de l'occurrence d'opération étant communiquée à tous les autres *schémas d'opération* faisant référence à cette variable.

4.4.1.9 Position dans l'espace

En Z, le concept d'espace n'est pas considéré comme primitif. La position dans l'espace à laquelle une action se produit ne peut être donnée en Z qu'en termes du modèle de spécification et non en termes du système réel modélisé. Une position dans l'espace pourrait donc être introduite sous forme de type Z. Cela étant, on peut spécifier des relations qui associent des *schémas d'opération* à des positions données dans l'espace. Il est alors possible de raisonner en termes de positions dans l'espace auxquelles des actions peuvent se produire.

4.4.1.10 Position dans le temps

En Z, le concept de temps n'est pas considéré comme primitif. La position dans le temps à laquelle une action se produit ne peut être donnée en Z qu'en termes du modèle de spécification et non en termes du système réel modélisé. Une position dans le temps pourrait donc être introduite sous forme de type Z pouvant être associé à des actions données, par exemple au moyen d'une certaine relation. Il est alors possible de raisonner en termes de positions dans le temps auxquelles des actions peuvent se produire.

4.4.1.11 Point d'interaction

Le concept de point d'interaction dépend des définitions de l'interaction et des positions dans l'espace et dans le temps. Voir 4.4.1.3, 4.4.1.9 et 4.4.1.10.

4.4.2 Concepts de spécification

4.4.2.1 Composition

- **d'objets:** la composition d'objets n'est pas explicitement offerte par le langage Z, en raison, entre autres, de l'absence d'encapsulation. Toutefois, il est possible de modéliser certaines caractéristiques de composition par l'inclusion de schémas et la redéfinition d'opérations par promotion.
- **de comportements:** étant donné que, dans le cas le plus dégénéré, un comportement peut être considéré comme une action et qu'une action en Z correspond à l'exécution d'une opération définie par un *schéma d'opération*, la composition d'actions est équivalente à la combinaison de *schémas d'opération* en Z. Les schémas d'opération peuvent être combinés de diverses manières en Z, par exemple:
  - *par calcul de schéma;*
  - *par composition de schémas* (;);
  - *par remplacement* (⊕).

En général, les caractéristiques du comportement résultant peuvent être dérivées de la composition qui ne découle pas nécessairement de chacun des comportements combinés. En outre, les détails non pertinents des comportements combinés peuvent être supprimés.

4.4.2.2 Objet composite

Voir ci-dessus l'interprétation relative à la composition.

4.4.2.3 Décomposition

- **d'objets:** voir ci-dessus l'interprétation relative à la composition d'objets.
- **de comportements:** voir ci-dessus l'interprétation relative à la composition de comportements.

4.4.2.4 Compatibilité de comportements

La compatibilité de comportements est basée sur la notion de capacité de substitution dans un environnement donné. L'extension est l'un des moyens permettant de parvenir à cette compatibilité. Une extension de squelette de base peut avoir un *schéma d'état* avec des composants supplémentaires, un invariant d'état plus puissant, des conditions initiales plus robustes et un plus grand nombre de *schémas d'opération*. Les *schémas d'opération* associés à l'extension du type de squelette peuvent avoir des *préconditions* plus faibles et des *postconditions* plus robustes que les *schémas d'opération* correspondants du type de squelette de base.

4.4.2.5 Affinement

L'affinement est le processus de transformation d'une spécification en une spécification plus détaillée. Etant donné que le langage Z traite d'abstractions de systèmes où les données et les opérations effectuées sur ces données servent à représenter le système considéré, deux formes principales d'affinement ont été identifiées:

- l'*affinement d'opération*; et
- l'*affinement de données*.

L'affinement d'une spécification doit garantir la compatibilité de comportement entre la spécification et l'affinement. Pour cela, il existe certaines conditions permettant de garantir que l'affinement d'une spécification Z produit une spécification plus détaillée valable. Il s'agit des conditions de *sécurité* et d'*existence*. La condition de sécurité sur l'affinement d'une spécification est la suivante: toute circonstance acceptable pour la spécification doit être acceptable pour l'affinement. La condition d'existence sur l'affinement d'une spécification est la suivante: pour toute circonstance acceptable pour la spécification, le comportement de l'affinement doit être autorisé par la spécification.

Il faut appliquer les conditions de sécurité et d'existence à la fois à l'affinement d'opération et à l'affinement de données.

4.4.2.6 Trace

La modélisation d'une trace en Z est limitée pour deux raisons. Premièrement, on ne peut pas enregistrer d'action d'objet et deuxièmement, il n'y a pas de distinction sur le plan sémantique entre les actions internes et les actions observables comme il est dit dans la Note en 4.4.1.3.

4.4.2.7 Type (d'un < *X* >)

Un objet, une interface ou une action peut avoir de nombreux types ODP différents. Les types ODP correspondent à des ensembles en Z, où le prédicat de caractérisation est donné par les membres de l'ensemble.

4.4.2.8 Classe (des < *X* >)

Il s'agit de l'ensemble de tous les < *X* > tels que le prédicat associé aux membres de l'ensemble, c'est-à-dire le type ODP, est vrai.

4.4.2.9 Sous-type/Supertype

Les sous-types et les supertypes de l'ODP correspondent respectivement aux sous-ensembles et aux superensembles en Z.

4.4.2.10 Sous-classe/Superclasse

Les sous-classes et les superclasses de l'ODP correspondent respectivement aux relations de sous-ensemble et de superensemble en Z.

4.4.2.11 Squelette de < *X* >

- **Squelette d'objet:** il s'agit des fragments d'une spécification qui représentent un état et qui ont une identité (immuable) unique à laquelle il peut être fait référence ainsi que de l'ensemble associé de *schémas d'opération* qui agissent sur cet état. Si le squelette d'objet est générique, la forme précise du squelette ne sera donnée que lorsque le type des paramètres de paramétrage sera donné.
- **Squelette d'interface:** il s'agit d'un ensemble de *schémas d'opération* qui sont dérivés du texte Z représentant un squelette d'objet de la manière décrite au paragraphe traitant de l'interprétation du concept d'interface (voir 4.4.1.4). Si le squelette d'objet est générique, la forme précise du squelette d'interface ne sera donnée que lorsque le type des paramètres de paramétrage sera donné. Des squelettes d'interface peuvent être combinés au moyen des opérations Z pour la combinaison de schémas.
- **Squelette d'action:** il s'agit d'un *schéma d'opération*. Des squelettes d'action peuvent être combinés au moyen des opérations Z pour la combinaison de schémas. Si le squelette d'action est générique, la forme précise du squelette d'action ne sera donnée que lorsque le type des paramètres de paramétrage sera donné.

4.4.2.12 Signature d'interface

Il s'agit de l'ensemble des squelettes d'action associés aux interactions d'une interface.

NOTE – Il convient de noter que dans le texte de la Rec. UIT-T X.902 | ISO/CEI 10746-2, la signature d'interface est définie comme un ensemble de squelettes d'action associés aux interactions d'une interface. Etant donné qu'un squelette d'action est constitué par les éléments communs d'un ensemble d'actions, cette définition est vraisemblablement incorrecte. En effet, tout squelette d'action est susceptible d'inclure des informations sémantiques tout comme des informations syntaxiques. Toutefois, les interprétations communes de signature d'interface se situent d'abord au niveau syntaxique. Si une notion syntaxique de signature d'interface est prise en considération, elle correspond à l'ensemble des squelettes d'action normalisés associés aux interactions d'une interface, toutes les variables de non-entrée/sortie déclarées dans la signature du *schéma d'opération* étant quantifiées dans la partie prédicat du *schéma d'opération.*

4.4.2.13 Instanciation (d'un squelette de < *X* >)

- **d'un squelette d'objet:** il s'agit de l'initialisation du texte Z représentant le squelette d'objet. Cette instanciation est souvent prise en charge explicitement en Z au moyen d'un schéma d'initialisation. L'exécution de ce schéma d'initialisation doit satisfaire à la condition selon laquelle un état valide doit exister pour l'objet après cette exécution, c'est-à-dire un état qui satisfasse à tous les *invariants* qui peuvent être présents.

  NOTE – Ces *invariants* peuvent se rapporter à d'autres objets.

- **d'un squelette d'interface:** il s'agit de contraindre des fragments de squelette d'interface de la spécification Z par la spécification des paramètres génériques et par l'imposition de prédicats appropriés aux variables auxquelles se réfère le squelette.

- **d'un squelette d'action:** il s'agit de l'exécution d'une opération spécifiée dans un *schéma d'opération.*

4.4.2.14 Rôle

Un rôle peut être représenté par un nom qui identifie un objet individuel, par exemple par un identificateur figurant dans le *schéma d'état* associé à l'objet. Ce nom peut alors être utilisé avec un *schéma d'encadrement* pour promouvoir les opérations de l'objet individuel en vue de réaliser tout le système (spécification).

NOTE – Cette description peut ne pas recouvrir totalement les éléments du texte associé dans la Rec. UIT-T X.902 | ISO/CEI 10746-2.

4.4.2.15 Création (d'un < *X* >)

- **d'un objet:** pour créer un objet en Z, il faut fournir un état initial valide pour le texte Z associé au squelette d'objet, c'est-à-dire qu'il faut établir une liaison entre les variables données dans le *schéma d'état* de l'objet et les valeurs initiales qui leur sont associées. Cette création est souvent prise en charge explicitement en Z au moyen d'un schéma d'initialisation. Dans ce cas, l'action de création est représentée par l'exécution de l'opération donnée dans le schéma d'initialisation.

- **d'une interface:** la création d'une interface en Z est liée de manière inhérente à la création d'objets. En effet, lorsque le texte associé à un squelette d'objet est initialisé, tous les squelettes d'interface susceptibles d'être présents sont aussi initialisés.

Pour la création, il est nécessaire de garantir que l'identité de l'objet créé est unique dans la spécification. Cela peut se faire au moyen d'un *schéma d'encadrement* avec un prédicat approprié garantissant que les identités de tous les objets créés sont uniques. Ce *schéma d'encadrement* peut alors servir à promouvoir l'initialisation d'un objet en vue de réaliser toute la spécification.

NOTE –

a) D'après le texte donné ici, à partir du moment où un schéma d'initialisation est donné dans une spécification, un objet est créé. Toutefois, dans une spécification Z, il n'y a pas de notion appliquant réellement ce schéma d'initialisation. Il s'agit donc en réalité de l'introduction d'un objet, c'est-à-dire qu'un objet est instancié par un mécanisme non couvert par le modèle. Il semblerait que la notion d'initialisation d'une spécification Z soit en partie de la création (dans le sens où un schéma d'initialisation est donné) et en partie de l'introduction (dans le sens où l'application du schéma d'initialisation n'est pas couverte par le modèle).

b) Normalement, après l'application d'un schéma d'initialisation, il faut une preuve afin de garantir que l'objet est dans un état initial valide.

4.4.2.16 Introduction (d'un objet)

Voir création (d'un objet) (4.4.2.15).

4.4.2.17 Suppression (d'un < *X* >)

Il peut être possible d'avoir une représentation abstraite de la suppression avec utilisation d'un *schéma d'encadrement*. La suppression peut alors être modélisée comme la promotion d'une opération visant à supprimer l'état et l'identité d'un objet dans tout le système.

On pourrait aussi modéliser une forme de suppression sur la base de l'inactivité. Par exemple, un objet dont les comportements futurs associés sont susceptibles de ne plus se produire en raison de la transgression d'*invariants* peut dans une certaine mesure être considéré comme supprimé. Toutefois, cette forme de suppression ne répond peut-être pas précisément à la définition donnée dans la Rec. UIT-T X.902 | ISO/CEI 10746-2, dans le sens où il ne se produit pas réellement de destruction.

4.4.2.18 Instance d'un type

En Z, une instance de type ODP est représentée par un élément de l'ensemble dont les membres satisfont le prédicat, c'est-à-dire le type ODP. Si on considère une notion de type plus spécifique, comme le type de squelette, une instance de type d'objet ou d'interface correspond à l'initialisation du texte Z (ou d'une extension de ce texte) représentant l'objet ou l'interface considéré, telle qu'il existe un état initial valide pour cet objet ou cette interface. Ici, le prédicat de caractérisation est donné par le texte de spécification et les prédicats associés sur les éventuelles liaisons légales (*invariants*), ce prédicat devant être satisfait par le schéma d'initialisation pour que la classification comme instance du type d'objet ou d'interface soit possible.

De même qu'un *schéma d'opération* donne les caractéristiques d'un type d'action, une instance de type d'action est donnée par l'occurrence de ce *schéma d'opération* ou par l'occurrence d'un autre *schéma d'opération* qui est une extension de ce *schéma d'opération*, car l'extension comprend les éléments de caractérisation du type d'action.

4.4.2.19 Type de squelette (d'un < *X* >)

En Z, un type de squelette d'objet ou d'interface est un prédicat exprimant qu'un schéma d'initialisation laisse l'objet et les interfaces associées dans un état initial valide. Par conséquent, toutes les variables d'état doivent être liées et tous les prédicats nécessaires (*invariants*) satisfaits. En cas de vérification fréquente d'un type de squelette d'objet ou d'interface, il faut établir une preuve.

Un type de squelette d'action correspond à un prédicat exprimant qu'un squelette d'action, tel qu'il est donné par un *schéma d'opération*, peut se produire, autrement dit que c'est une instanciation d'un *schéma d'opération* donné. Toutes les instanciations doivent donc satisfaire les prédicats sur les liaisons légales des variables, donnés dans les prédicats du *schéma d'opération*.

4.4.2.20 Classe de squelette (d'un < *X* >)

Une classe de squelette d'un < *X* > est l'ensemble de tous les < *X* > qui sont des instances de ce squelette de < *X* >, où un < *X* > désigne un objet, une interface ou une action.

4.4.2.21 Classe dérivée/classe de base

Etant donné, en Z, deux squelettes A et B où A est une modification incrémentielle de B et les instances de A et B sont dans une relation classe dérivée/classe de base, les modifications incrémentielles apportées à B pour produire A peuvent comprendre:

- l'ajout ou la suppression de paramètres d'état;
- l'ajout, la suppression ou la modification d'opérations;
- le renforcement ou l'affaiblissement des *invariants.*

4.4.2.22 Invariant

Il s'agit d'un prédicat qui doit toujours être vrai. En Z, il est possible de consigner les *invariants* directement dans les schémas et les *descriptions axiomatiques.* Les *invariants* imposent souvent des restrictions aux éventuelles liaisons que les variables peuvent présenter dans les schémas ou les *descriptions axiomatiques*. En tant que tel, un *invariant* sert souvent à restreindre les comportements possibles donnés dans une spécification.

4.4.2.23 Précondition

Il s'agit de la condition sur l'état du système avant l'occurrence d'une opération définie par un *schéma d'opération* et sur ses entrées, cette condition étant telle qu'il existe un état possible après l'exécution de l'opération et des sorties qui satisfassent les *postconditions*. En Z, il est possible de consigner directement les *préconditions*.

4.4.2.24 Postcondition

Il s'agit d'un prédicat qui décrit l'ensemble des états dans lesquels un système donné peut se trouver après l'exécution d'une opération définie par un *schéma d'opération*. En Z, il est possible de consigner directement les *postconditions*.

4.5 Sémantique architecturale en ESTELLE

ESTELLE est un langage FSL normalisé (voir ISO/CEI 9074). Un tutorial figure dans l'ISO. Il existe divers outils prenant en charge la simulation ainsi que la réalisation répartie des spécifications ESTELLE.

ESTELLE est basé sur des automates à états finis étendus. Un système est modélisé par un ensemble – dont la structure est hiérarchique – d'instances de module, communiquant en mode asynchrone par des messages échangés sur des canaux. La syntaxe du langage et la définition des types de données et des variables sont basés sur le langage Pascal ISO.

Dans une spécification ESTELLE, l'interface visible de l'extérieur associée à un module est définie dans l'*en-tête du module*, tandis que le *corps du module* décrit la structure interne et le comportement du module. Les *instances de module* définissent des *points d'interaction* par lesquels des messages peuvent être envoyés et reçus. *Deux points d'interaction* peuvent être connectés s'ils ont été définis pour les rôles en opposition associés à la même *définition de canal*. Une *définition de canal* contient deux *rôles*, un pour chacune des extrémités du *canal*. Pour chaque *rôle* sont définis les messages (appelés *interactions* en ESTELLE) pouvant être envoyés. Une telle *définition d'interaction* est constituée d'un nom et d'un ensemble de paramètres. A chaque *point d'interaction* est assignée une file d'attente dans laquelle les messages entrants sont stockés. Une *instance de module* peut aussi comporter une file d'attente commune partagée par plusieurs *points d'interaction*, voire par tous.

La structure d'une spécification ESTELLE est dynamique, c'est-à-dire que les *instances de module* peuvent être instanciées et libérées et les points d'interaction peuvent être connectés et déconnectés dynamiquement. Les instances de module d'une spécification ESTELLE font l'objet d'un ordre hiérarchique puissant. Chaque instance de module peut instancier ou libérer des *instances de modules* filles ou bien connecter ou déconnecter leurs *points d'interaction*. Une *instance de module* ne peut accéder aux instances sœurs (ou aux autres *instances de module* qui ne sont pas ses propres filles) que par l'échange d'*interactions* sur des *canaux*.

A l'origine, ESTELLE a été élaboré en vue de la spécification de services et de protocoles de communication. ESTELLE prend en charge l'encapsulation, mais il ne contient pas les éléments orientés objet d'héritage ou de sous-typage. Malgré cela, ESTELLE permet d'exprimer la plupart des concepts ODP. Les spécifications ESTELLE sont faciles à lire et comme ESTELLE est une technique constructive, il est très adapté à la simulation et à la réalisation.

Le paragraphe qui suit montre comment les concepts ODP de base peuvent être exprimés au moyen d'ESTELLE.

Dans ce texte, les concepts ESTELLE définis dans l'ISO sont en *italique*. Il convient de noter qu'ESTELLE utilise la notion d'*interaction* pour désigner les messages échangés entre *instances de module* en communication.

4.5.1 Concepts de modélisation de base

4.5.1.1 Objet

En ESTELLE, un objet est modélisé par une *instance de module.*

4.5.1.2 Environnement (d'un objet)

Il s'agit de la partie de la spécification qui ne fait pas partie de l'*instance de module*; en particulier l'*instance mère* et les autres instances qui sont raccordées aux *points d'interaction* de l'instance de module via des *canaux.*

4.5.1.3 Action

En ESTELLE, une action est représentée par l'exécution d'une *clause WHEN*, l'exécution d'une instruction d'action, une *transition* globale ou l'exécution d'une *procédure*. Citons des instructions d'action possibles:

- *sortie;*
- *initialisation;*
- *connexion;*
- *rattachement;*
- *libération;*
- *déconnexion;*
- *détachement;*
- *instruction d'affectation.*

Il existe plusieurs types d'**interactions**. L'exécution d'une instruction de *sortie* est une interaction tout comme l'exécution d'une *clause WHEN*. De même, la séquence d'actions constituée de la *sortie* d'une *interaction* par un *point d'interaction* et de sa consommation ultérieure par l'exécution d'une *clause WHEN* peut être considérée comme une interaction. La mise à jour d'une *variable exportée* constitue un autre type d'interaction. Toutes les autres actions sont **internes**.

NOTE – Etant donné que les *canaux* ESTELLE possèdent des files d'attente infinies, l'environnement est toujours prêt à participer aux *interactions*.

4.5.1.4 Interface

Il existe deux types d'interface en ESTELLE:

- le premier type est constitué par l'ensemble de toutes les *interactions* définies pour le *rôle* assigné (dans la *définition de canal* correspondante) en un *point d'interaction externe* d'objet;
- le second type est constitué par l'ensemble de toutes les *variables exportées* d'un objet.

Dans le premier cas, l'ensemble des interactions (constituant l'interface) contient l'ensemble des instructions de *sortie* ou des *clauses WHEN* que l'objet exécute au *point d'interaction*. Dans le second cas, l'ensemble des interactions est constitué de toutes les instructions qui permettent de lire ou d'écrire les *variables exportées*. Au niveau de ce type d'interface, les interactions ne sont possibles qu'entre une *instance de module* et son *instance mère*.

4.5.1.5 Activité

En général, une activité ne peut pas être indiquée explicitement, car elle peut couvrir plusieurs objets. Une activité est une certaine séquence d'actions, par exemple une *transition* ou une *procédure* ou une *fonction*, si les instructions simples sont considérées comme des actions.

4.5.1.6 Comportement (d'un objet)

Le comportement d'un objet est déterminé par l'ensemble de toutes les *transitions* de cet objet. Les contraintes sur les circonstances dans lesquelles les actions spécifiées peuvent avoir lieu sont définies dans les *clauses de transition* (*clause FROM, clause PROVIDED,* etc.). Un objet peut présenter un comportement non déterministe.

4.5.1.7 Etat (d'un objet)

L'état d'un objet se compose des aspects suivants:

- l'*état de contrôle* de l'*instance de module*;
- le contenu des files d'attente associées aux *points d'interaction*;
- les valeurs des *variables exportées* et internes et les paramètres formels de l'*instance de module*;
- l'état des *instances* filles existantes, la structure de leur connexion et les *variables exportées*.

L'ensemble de ces aspects détermine l'ensemble de toutes les séquences d'actions (*transitions*) auxquelles l'*instance de module* peut prendre part.

4.5.1.8 Communication

L'information est transportée entre objets par l'un des moyens suivants:

- par la *sortie* et la réception ultérieure d'une *interaction* (une séquence d'actions constituant une interaction);
- par la lecture et la mise à jour d'une *variable exportée*.

4.5.1.9 Position dans l'espace

Les actions se produisent dans des *instances de module* ou en des *points d'interaction* d'*instances de module*. La position dans l'espace d'une action correspond donc à la position dans l'espace de l'*instance de module* associée.

4.5.1.10 Position dans le temps

En ESTELLE, le temps n'est représenté que dans les *clauses DELAY* éventuellement associées aux *transitions*. La seule hypothèse qui soit faite sur le temps est que les intervalles de temps sont uniformes tout au long de l'exécution.

4.5.1.11 Point d'interaction

Un point d'interaction est représenté par un *point d'interaction* ou par l'ensemble de toutes les *variables exportées* d'un objet.

4.5.2 Concepts de spécification

L'expression des concepts de spécification donnés pose des problèmes en ESTELLE, car la prise en charge des concepts orientés objet est limitée.

4.5.2.1 Composition

- **d'objets:** une *instance de module* peut être composée d'un ensemble d'*instances de module* filles. Au niveau le plus élevé, une spécification peut être composée d'un ensemble d'*instances de système*.
- **de comportements:** lorsqu'une *instance de module* est une composition d'*instances* filles, le comportement de ces instances est:
  - composé en parallèle, entrelacé, si le *module* père est classé comme une *activité* ou comme une *activité-système*, ou
  - composé en parallèle, synchrone, si le *module* père est classé comme un *processus* ou comme un *processus-système*.

Au niveau le plus élevé, le comportement des *instances* filles est composé en parallèle, si la spécification est composée d'un ensemble d'instances de système.

4.5.2.2 Objet composite

Il s'agit d'une *instance de module* décrite par un ensemble d'*instances de module* filles.

4.5.2.3 Décomposition

- **d'objets:** il s'agit de la spécification d'un objet donné sous forme de composition.
- **de comportements:** il s'agit de la spécification d'un comportement donné sous forme de composition.

4.5.2.4 Compatibilité de comportements

On ne peut pas exprimer directement la compatibilité de comportement en ESTELLE. Toutefois, la base sémantique du langage en termes de systèmes de transition permet de définir la compatibilité de comportement et de la vérifier.

4.5.2.5 Affinement

On peut affiner un objet en le sous-structurant en *instances* filles coopérantes.

4.5.2.6 Trace

Les traces peuvent être obtenues à partir de l'interprétation dynamique (exécution/simulation) d'une spécification ESTELLE.

4.5.2.7 Type (d'un < *X* >)

On ne peut pas formuler explicitement de prédicats en ESTELLE.

4.5.2.8 Classe (de < *X* >)

Non prise en charge.

4.5.2.9 Sous-type/supertype

Non pris en charge.

4.5.2.10 Sous-classe/superclasse

Non prises en charge.

4.5.2.11 Squelette de < *X* >

Un squelette d'objet est représenté par une *définition de corps de module* et par la *définition d'en-tête de module* correspondante. Les squelettes d'action sont les suivants:

- *sortie*;
- *initialisation*;
- *libération*;
- *connexion*;
- *déconnexion*;
- *rattachement*;
- *détachement*;
- *instruction d'affectation*;
- *clause WHEN*.

Comme il existe deux types d'interfaces, un squelette d'interface est donné:

- par la *définition de canal* correspondante (pour les ensembles d'*interactions* en un *point d'interaction*). Dans ce cas, le comportement de l'interface associé peut être spécifié au moyen d'une *instance de module* fille qui est *instanciée* et rattachée au *point d'interaction* lorsque l'interface est créée. Si c'est le cas, le squelette d'interface contient la *définition du corps* et de l'*en-tête de ce module*.
- dans la *définition de l'en-tête du module* de l'objet (pour les *variables exportées*).

4.5.2.12 Signature d'interface

Une signature d'interface est représentée de l'une des deux manières suivantes:

- par les définitions des *interactions* qui figurent dans l'interface (pour un ensemble d'*interactions* en un *point d'interaction*).
- par les actions d'affectation pour les *variables exportées*.

4.5.2.13 Instanciation (d'un squelette de < *X* >)

Une instanciation d'objet est une *instance de module* de la *définition de module* correspondante. Une instanciation d'interface est un *point d'interaction* particulier (éventuellement avec une *instance de module* fille spécifique qui lui est rattachée, représentant le comportement de l'interface) ou l'ensemble des *variables exportées* d'une *instance de module* particulière.

4.5.2.14 Rôle

Un *rôle* est associé à chaque déclaration de *point d'interaction*. Les *points d'interaction* à raccorder doivent posséder des *rôles* opposés sur la même *définition de canal*. ESTELLE prend aussi en charge la spécification de différents *corps de module* pour le même *en-tête de module*. Grâce à cela, un choix est possible parmi différents comportements lorsqu'un objet est instancié. Les interfaces de l'objet sont les mêmes, quel que soit le *corps de module* retenu.

4.5.2.15 Création (d'un < *X* >)

Les objets sont créés au moyen de l'action d'*initialisation*. Les interfaces sont créées implicitement, au moment où l'objet est créé. Il n'y a pas de création dynamique d'interfaces. Toutefois, la création d'une interface sous forme de *point d'interaction* peut être modélisée par la sélection d'un *point d'interaction* (dans une matrice de *points d'interaction*) avec un *canal* approprié et un *rôle* associé. Si un *module* fils est spécifié pour représenter le comportement de l'interface, une *instance* du *module* est *instanciée* et rattachée au *point d'interaction* fourni.

4.5.2.16 Suppression (d'un < *X* >)

Un objet est supprimé explicitement au moyen de l'action de *libération*. Les interfaces sont supprimées implicitement en même temps que l'objet. Si la création dynamique d'interfaces est modélisée comme mentionné plus haut (voir 4.5.2.15), la suppression d'une interface correspond à la *libération* de l'*instance de module fille* rattachée (le cas échéant) et au marquage du *point d'interaction* comme ne représentant pas une interface.

4.5.2.17 Introduction (d'un objet)

On ne peut pas introduire d'objet. D'une manière générale, les mécanismes qui ne sont pas couverts par le modèle peuvent être réalisés au moyen de *fonctions* ou de *procédures externes primitives*.

4.5.2.18 Instance d'un type

Comme les sous-type et sous-classe ne sont pas pris en charge en ESTELLE, les instances d'un squelette sont donnés par les instanciations de ce squelette.

4.5.2.19 Type de squelette (d'un < *X* >)

Pour un objet, on peut énoncer le prédicat suivant: un objet est une instance de la *définition de module* correspondante. Une interface (un ensemble d'*interactions* en un *point d'interaction*) est une instance de la *définition de canal* correspondante. Une interface représentée par l'ensemble de toutes les *variables exportées* est une instance de la *définition d'en-tête de module* correspondante. Une action est une instance du squelette d'action correspondant.

NOTE – En ESTELLE, on peut affirmer qu'un objet, une interface ou une action est une instanciation d'un squelette donné. Le concept de sous-classe n'étant pas réellement pris en charge, seules les instanciations d'un squelette peuvent être identifiées, mais pas les instances.

4.5.2.20 Classe de squelette (d'un < *X* >)

La classe de squelette d'un objet est l'ensemble de toutes les *instances* du même module. La classe de squelette d'une interface est l'ensemble de tous les *points d'interaction* définis avec la même *définition de canal* et le même *rôle*.

4.5.2.21 Classe dérivée/classe de base

Non prises en charge.

4.5.2.22 Invariant

On ne peut pas formuler explicitement d'invariants en ESTELLE.

4.5.2.23 Précondition

Les préconditions de l'exécution d'une *transition* sont indiquées dans les *clauses de la transition*. Les préconditions d'une action à l'intérieur d'un *bloc de transition* sont données par les préconditions de la transition ainsi que par les actions du *bloc de transition* qui précèdent l'action concernée.

4.5.2.24 Postcondition

La postcondition d'une transition est définie au moyen de la *clause TO* de la transition ainsi que par les actions figurant dans le *bloc de transition*.

SÉRIES DES RECOMMANDATIONS UIT-T

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

Genève, 1998