UNION INTERNATIONALE DES TÉLÉCOMMUNICATIONS

UIT-T X.904

SECTEUR DE LA NORMALISATION
DES TÉLÉCOMMUNICATIONS
DE L'UIT

Amendement 1
(03/2000)

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

Amendement 1: Formalisation de traitement

Recommandation UIT-T X.904 – Amendement 1

(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 des couches	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.369
Réseaux à protocole Internet	X.370–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 ISO/CEI 10746-4
RECOMMANDATION UIT-T X.904

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

AMENDEMENT 1

Formalisation de traitement

Résumé

L'Amendement 1 de la Rec. UIT-T X.904 | ISO/CEI 10746-4 précise et étend la sémantique architecturale du traitement réparti ouvert (ODP, *open distributed processing*) en y ajoutant une formalisation de traitement dans le modèle de référence du traitement réparti ouvert (RM-ODP, *reference model for open distributed processing*). Le langage de traitement du modèle RM-ODP permet de décrire les systèmes ODP comme des ensembles d'objets en interaction. Dans le présent amendement, les concepts et les règles du langage de traitement ODP sont définis selon différentes techniques de description formelle (LOTOS, SDL, Z et Estelle).

Source

L'Amendement 1 de la Recommandation X.904 de l'UIT-T, élaboré par la Commission d'études 7 (1997-2000) de l'UIT-T, a été approuvé le 31 mars 2000. Un texte identique est publié comme Norme Internationale ISO/CEI 10746-4, Amendement 1.

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'étude à traiter par les Commissions d'études de l'UIT-T, lesquelles élaborent en retour des Recommandations sur ces thèmes.

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

NORME INTERNATIONALE

RECOMMANDATION UIT-T

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

AMENDEMENT 1

Formalisation de traitement

1) Avant-propos

Remplacer le 1er paragraphe de l'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.

par:

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 et dans l'article 7 (Language de traitement) de 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.

2) Article 0 – Introduction

Remplacer la 4e énumération sous "Le modèle RM-ODP se compose"

– 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.

par

– 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 traitement 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.

Remplacer le 4e paragraphe:

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.

par

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 traitement 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.

3) Article 1 – Domaine d'application

Ajouter le paragraphe suivant à la fin du Domaine d'application

L'Annexe A indique une manière selon laquelle le langage de point de vue traitement de la Rec. UIT-T X.903 | ISO/CEI 10746-3 peut-être représenté dans les langages formels LOTOS, SDL, Z et Estelle. La présente Recommandation | Norme internationale fait également appel aux concepts définis dans la Rec. UIT-T X.902 | ISO/CEI 10746-2.

4) Article 2 – Références normatives

Changer la date de publication pour la Recommandation UIT-T Z.100 de (1993) en (1999).

ISO/CEI 13568:

Ajouter la référence suivante:

Z Notation, ISO/CEI JTC 1 SC 22 GT 19, Advanced Working Draft 2.C, 13 juillet 1999.

5) Paragraphe 3.2 – Termes définis dans la Recommandation UIT-T Z.100

Remplacer la liste des termes par la suivante:

active, adding, all, alternative, and, any, as, atleast, axioms, block, call, channel, comment, connect, connection, constant, constants, create, dcl, decision, default, else, endalternative, endblock, endchannel, endconnection, enddecision, endgenerator, endnewtype, endoperator, endpackage, endprocedure, endprocess, endrefinement, endselect, endservice, endstate, endsubstructure, endsyntype, endsystem, env, error, export, exported, external, fi, finalized, for, fpar, from, gate, generator, if, import, imported, in, inherits, input, interface, join, literal, literals, map, mod, nameclass, newtype, nextstate, nodelay, noequality, none, not, now, offspring, operator, operators, or, ordering, out, output, package, parent, priority, procedure, process, provided, redefined, referenced, refinement, rem, remote, reset, return, returns, revealed, reverse, save, select, self, sender, service, set, signal, signallist, signalroute, signalset, spelling, start, state, stop, struct, substructure, synonym, syntype, system, task, then, this, timer, to, type, use, via, view, viewed, virtual, with, xor.

6) Paragraphe 3.3 – Termes définis dans "The Z Base Standard"

Changer le titre du paragraphe par:

3.3 – Termes définis dans "The Z Notation"

Remplacer la liste des termes par la suivante:

affinement de données, affinement d'opération, théorie des schémas, composition de schéma, description axiomatique, occultation, remplacement, schéma (opération, état, encadrement), séquence, type.

7) Annexe A

Ajouter une nouvelle Annexe A comme suit:

Annexe A

Formalisation de traitement

A.1 Formalisation du langage de point de vue traitement en LOTOS

A.1.1 Concepts

La formalisation du langage de traitement en LOTOS utilise les concepts définis dans la formalisation des règles de modélisation et de structuration de base indiquées aux articles 8 et 9 de la Rec. UIT-T X.902 | ISO/CEI 10746–2.

Structures élémentaires associées aux interfaces opération et signal

Pour formaliser le langage de traitement en LOTOS, il est nécessaire d'introduire certaines structures élémentaires. Celles-ci comportent des paramètres susceptibles d'être associés à certaines interfaces de traitement ainsi qu'un modèle d'information de base susceptible d'être utilisé dans un flux.

Pour formaliser des paramètres il est nécessaire d'adopter deux concepts, à savoir celui de noms d'objets et celui de types d'objets. Les noms sont simplement des étiquettes. Comme nous allons le voir, du point de vue traitement, ces étiquettes doivent faire l'objet de contrôles, visant par exemple à en vérifier l'égalité, au moment où les interfaces sont créées. En règle générale, nous pouvons représenter les noms comme suit:

```
type Name is Boolean
        sorts Name
        opns    newName: -> Name
                anotherName: Name -> Name
                _eq_,_ne_: Name, Name -> Bool
endtype (* Name *)
```

Par souci de concision, nous faisons l'impasse sur les équations, qui ne sont pas censées poser de problème particulier. Il est possible d'être ici plus normatif, par exemple en utilisant des chaînes de caractères de la bibliothèque LOTOS. La seule chose qui nous intéresse en ce qui concerne les noms est que nous puissions en déterminer l'égalité ou l'inégalité.

Comme indiqué dans la présente Recommandation | Norme internationale, un type selon la terminologie ODP ne peut pas être interprété directement dans la partie "algèbre de processus" du langage LOTOS. Il est toutefois possible de modéliser des types à l'aide de la partie "*Act One*" du LOTOS. Bien qu'elle ait été expressément conçue pour représenter des types, la partie *Act One* est malheureusement limitée dans ses modalités de vérification des types et des relations entre les types. Par exemple, il est impossible de vérifier le sous-typage ou l'équivalence, voire l'isomorphisme entre les types, du fait que l'égalité de types en *Act One* est fondée sur l'équivalence de noms de sortes. A l'appui de notre raisonnement, nous introduisons ici une notion élémentaire de types qui nous permet de tester l'égalité, l'inégalité et le sous-typage.

```
type AnyType is  Boolean
    sorts AnyType
    opns newType: -> AnyType
         anotherType: AnyType -> AnyType
         _eq_,_isSubtype_: AnyType, AnyType -> Bool
endtype (* AnyType *)
```

Un paramètre est une relation entre un nom et sa représentation de type sous-jacente. Un paramètre peut donc être représenté comme suit:

```
type Param is Name, AnyType
        sorts Param
        opns newParam: Name, AnyType -> Param
             _eq_,_ne_,_isSubtype_: Param, Param -> Bool
endtype (* Param *)
```

Comme précédemment, nous devons procéder à des contrôles de l'égalité ou de l'inégalité des paramètres ainsi que dans le cas où un paramètre est un sous-type d'un autre paramètre. Deux paramètres ont une relation de sous-typage lorsque leurs types ont une relation de sous-typage. Il est par ailleurs utile pour nous d'introduire des séquences de ces paramètres.

```
type PList is String actualizedby Param
        using sortnames PList for String Param for Element Bool for FBool
        opns _isSubtype_: PList, PList -> Bool
endtype (* PList *)
```

Nous utilisons ici le type *String* de la bibliothèque LOTOS, actualisé avec le type *Param* défini précédemment. En outre, nous incluons ici une opération *isSubtype* permettant de vérifier si une séquence de paramètres est un sous-type d'une autre. Une liste de paramètres est un sous-type d'une autre liste lorsque tous les paramètres qu'elle contient sont des sous-types de ceux qui figurent dans la première liste. En outre, les paramètres devraient occuper la même position dans leurs listes respectives. Il convient de noter que ces paramètres sont susceptibles de contenir des références aux interfaces utilisées pour limiter les interactions qui peuvent se produire. Alors qu'il est tout à fait possible de modéliser une interface dans l'algèbre de processus, il est impossible de modéliser une référence à cette interface dans l'algèbre de processus qui, pour ainsi dire, s'approprie la fonctionnalité de cette interface. Pour surmonter cette difficulté, nous modélisons les références d'interface en *Act One*. Etant donné qu'une référence d'interface reproduit, entre autres choses, la signature de l'interface, nous fournissons un modèle de signatures en *Act One* pour les opérations. Les opérations se composent d'un nom, d'une séquence d'entrées et éventuellement d'une séquence de sorties. Pour simplifier les choses, nous ne nous soucions pas ici de savoir si l'opération est en notation infixée, préfixée ou suffixée. Cela peut-être représenté par le fragment LOTOS suivant:

```
type Op is Name, PList
        sorts Op
        opns makeOp: Name, PList -> Op
             makeOp: Name, PList, PList -> Op
             getName: Op -> Name
             getInps: Op -> PList
             getOuts: Op -> PList
             _eq_: Op, Op -> Bool
        eqns forall op1,op2: Op, n: Name; pl1, pl2: PList
ofsort Name     getName(makeOp(n,pl1,pl2)) = n;
ofsort PList    getInps(makeOp(n,pl1)) = pl1;
                        getInps(makeOp(n,pl1,pl2)) = pl1;
                        getOuts(makeOp(n,pl1)) = <>;
                        getOuts(makeOp(n,pl1,pl2)) = pl2;
ofsort Bool     op1 eq op2 = ((getName(op1) eq getName(op2)) and
                                        (getInps(op1) isSubtype getInps(op2)) and
                                        (getOuts(op2) isSubtype getOuts(op1)));
endtype (* Op *)
```

Le fait de disposer d'une méthode permettant de déterminer si deux opérations sont identiques ramène le problème du sous-typage entre types abstraits de données à une comparaison d'ensembles, dont les éléments sont les opérations créées. Ainsi, un serveur est un sous-type d'un autre serveur s'il prend en charge toutes les opérations de cet autre serveur. A noter ici que nous modélisons deux formes d'opérations: celle dont on n'attend pas de résultats et celle dont on en attend. En outre, nous introduisons des ensembles de ces opérations.

```
type OpSet is Set actualizedby Op
        using sortnames OpSet for Set Op for Element Bool for FBool
endtype (* OpSet *)
```

Cela étant, une référence d'interface peut être représentée par le fragment LOTOS suivant:

```
type IRef is OpSet
        sorts IRef
        opns makeIRef     :  OpSet -> IRef
             NULL         : -> IRef
             getOps       : IRef -> OpSet
             _eq_         : IRef, IRef -> Bool
        eqns forall o: OpSet; ir1, ir2: IRef
             ofsort OpSet  getOps(makeIRef(o)) = o;
             ofsort Bool   ir1 eq ir2 = getOps(ir1) eq getOps(ir2) ;
endtype (* IRef *)
```

Nous constatons ici que l'égalité des références d'interface est fondée uniquement sur les opérations contenues dans la référence considérée. Elle pourrait parfaitement être étendue à d'autres aspects, tels que l'emplacement de l'interface ou les contraintes qui en restreignent l'utilisation. En outre, nous introduisons des ensembles de ces références d'interface.

```
type IRefSet is Set actualizedby IRef
        using sortnames IRefSet for Set IRef for Element Bool for FBool
endtype (* IRefSet *)
```

Structures élémentaires associées aux interfaces flux

Le langage de point de vue traitement de la Recommandation UIT-T X.903 | ISO/CEI 10746–3 examine en outre les interfaces relatives au flux continu de données, par exemple multimédias. Ces interfaces sont appelées *interfaces flux*. Les interfaces flux contiennent des ensembles finis de flux. Ces flux peuvent provenir de l'interface (flux producteur) ou se diriger vers l'interface (flux consommateur). Chaque flux est modélisé au moyen d'un modèle d'action. Chaque modèle d'action contient le nom du flux, le type du flux ainsi qu'une indication de causalité pour ce flux.

Le point de vue traitement s'éloigne du contenu du flux d'informations proprement dit. Nous retenons ici un principe générique du flux d'informations selon lequel ce flux est représenté par une séquence d'éléments de flux. Un élément de flux peut être considéré comme un élément particulier du flux d'informations. Nous constatons ici que les flux sont considérés du point de vue traitement comme des actions continues. Dans le modèle que nous présentons ici, les flux sont représentés comme des séquences d'événements temporels discrets. D'un côté cela nous permet de remédier aux problèmes de synchronisation des flux d'informations, même si cela se traduit par la perte du caractère continu des flux.

Chaque élément d'un flux d'information peut être considéré comme une unité composée de données (qui peuvent être comprimées) et que nous représentons par *Données*. Ce modèle peut indiquer comment les informations ont été comprimées, quelles informations ont été comprimées, etc. En tant que tel, ce modèle n'est pas examiné plus avant ici. De plus, les éléments de flux contiennent un champ d'indication horaire servant à modéliser l'heure à laquelle l'élément de flux considéré a été envoyé ou reçu. En outre, il arrive souvent, dans le cas de flux multimédias, que des éléments de flux de synchronisation soient nécessaires, par exemple pour synchroniser le son avec l'image. C'est pourquoi nous associons un nom donné (*Name*) à chaque élément de flux. Ce nom peut ensuite servir à sélectionner un élément de flux donné, selon les besoins. Il en découle que nous pouvons modéliser un élément de flux comme suit:

```
type FlowElement is Name, NaturalNumber, Data, Param
sorts FlowElement
opns makeFlowElement: Data, Nat, Name -> FlowElement
     nullFlowElement  : -> FlowElement
     getData    : FlowElement -> Data
     getTime    : FlowElement -> Nat
     getName    : FlowElement -> Name
     toParam    : FlowElement -> Param
     setTime    : Nat, FlowElement -> FlowElement
eqns forall d: Data, s,t: Nat, n: Name
          ofsort Data       getData(makeFlowElement(d,t,n)) = d;
          ofsort Nat        getTime(makeFlowElement(d,t,n)) = t;
          ofsort Name       getName(makeFlowElement(d,t,n)) = n;
          ofsort FlowElement        setTime(s,makeFlowElement(d,t,n)) = makeFlowElement(d,s,n);
endtype (* FlowElement *)
```

Il convient de noter ici que nous modélisons le temps sous la forme d'un nombre naturel. Toutefois, les valeurs de modélisation retenues pourraient très bien aussi être exprimées en temps réel (continu) ou en intervalles de temps. Pour simplifier les choses, nous nous bornons ici à représenter le temps discret sous la forme d'un nombre naturel. Par ailleurs, nous introduisons une opération permettant de convertir un élément de flux en un paramètre. Pour simplifier les choses, nous faisons l'impasse sur les équations associées. En outre, nous introduisons des séquences de ces éléments de flux:

```
type FlowElementSeq is FlowElement
          sorts FlowElementSeq
          opns makeFlowElementSeq: -> FlowElementSeq
               addFlowElement: FlowElement, FlowElementSeq -> FlowElementSeq
               remFlowElement: FlowElement, FlowElementSeq -> FlowElementSeq
               getFlowElement: Name, FlowElementSeq -> FlowElement
               timeDiff: FlowElement, FlowElement -> Nat
          eqns forall f1, f2: FlowElement, fs: FlowElementSeq, n1,n2: Name
          ofsort FlowElementSeq
          getTime(f1) le getTime(f2) =>
                    addFlowElement(f1,addFlowElement(f2,makeFlowElementSeq)) =
                                                  addFlowElement(f2,makeFlowElementSeq);
          ofsort FlowElement
                    getFlowElement(n1,makeFlowElementSeq) = nullFlowElement;
                    n1 ne n2 =>
                              getFlowElement(n1,addFlowElement(makeFlowElement(d,t,n2),fs)) =
                                        getFlowElement(n1,fs);
                    n1 eq n2 =>
                              getFlowElement(n1,addFlowElement(makeFlowElement(d,t,n2),fs)) =
                                        makeFlowElement(d,t,n2);
endtype (* FlowElementSeq *)
```

Par souci de concision, nous ne reproduisons pas toutes les équations. Les éléments de flux sont ajoutés à la séquence pour autant qu'ils présentent des valeurs de champ d'indication horaire croissantes. Une opération permettant de traverser une séquence d'éléments de flux pour trouver un élément de flux nommé est prévue. En outre, nous introduisons une opération permettant de calculer les différences horaires entre les champs d'indication horaire de deux éléments de flux. Le recours à cette opération permet de préciser, par exemple, que tous les éléments de flux d'une séquence sont séparés par des champs d'indication horaire égaux. Nous sommes alors en présence d'un flux isochrone. En outre, nous introduisons des ensembles de ces séquences d'éléments de flux:

```
type FlowElementSeqSet is Set actualizedby FlowElementSeq
        using sortnames FlowElementSeqSet for Set FlowElementSeq for Element Bool for FBool
endtype (* FlowElementSeqSet *)
```

A.1.1.1 Signal

Aucun élément propre au LOTOS ne permet de faire la distinction entre un signal, un flux et une opération. Il peut arriver, toutefois, qu'un style de LOTOS permette de faire la distinction entre des signaux, des flux et des opérations. Par exemple, tous les signaux pourraient avoir des formats analogues pour leurs offres d'événement. Un exemple d'un format possible côté serveur d'un signal est représenté dans le fragment LOTOS suivant:

```
<g> ?<sigName: Name> !<myRef> ?<inArgs: PList>;
```

Ici et dans le reste du paragraphe A.1, nous adoptons la notation selon laquelle <*X*> tient lieu de paramètre fictif pour un *X*, c'est-à-dire que *g*, *sigName*, *myRef* et *inArgs* tiennent lieu respectivement de paramètres fictifs pour la porte, le nom du signal, la référence d'interface associée au serveur offrant ce signal et les paramètres associés au signal.

Un exemple de format possible côté client d'un signal est représenté dans le fragment LOTOS suivant:

```
<g> !<sigName> !<SomeIRef> !<inArgs>;
```

Ici le côté client du signal contient une porte (*g*), une étiquette pour le nom du signal (*sigName*), une référence désignant l'objet auquel le signal est envoyé (*SomeIRef*) et les paramètres associés au signal (*inArgs*). Nous verrons au § A.1.1.11 comment ces offres d'événement peuvent être utilisées pour créer des signatures d'interface signal.

A.1.1.2 Opération

Occurrence d'une interrogation ou d'une annonce.

A.1.1.3 Annonce

Interaction consistant en une seule invocation. Pour les raisons indiquées au § A.1.1.1, on ne peut utiliser qu'une convention de modélisation informelle pour modéliser des annonces. Un exemple de cette règle pour le côté client d'une annonce pourrait être représenté comme suit:

```
<g> !<invName> !<SomeIRef> !<inArgs>;
```

Le côté serveur d'une annonce pourrait être représenté comme suit:

```
<g> ?<invName: Name> !<myRef>  ?<inArgs: PList>;
```

Les structures de données sont ici analogues à celles qui sont décrites au § A.1.1.1. Nous verrons au § A.1.1.12 comment ces offres d'événement peuvent être utilisées pour créer des parties de signature d'interface opération.

A.1.1.4 Interrogation

Invocation entre un client et un serveur, suivie d'une ou plusieurs terminaisons entre ce serveur et ce client. Toutefois, pour les raisons indiquées au § A.1.1.1, on ne peut utiliser qu'une convention de modélisation informelle pour modéliser les interrogations. Un exemple de cette règle pour le côté client d'une interrogation pourrait être représenté comme suit:

```
<g> !<invName> !<SomeIRef> !<inArgs> !<outArgs>;
( <g> ?<termName:Name> !<myRef> ?<outArgs: PList>;  (* ... other behaviour *)
  [ ]  (* ... other terminations *))
```

Ici *termName* représente les noms de terminaison et *outArgs* représente les paramètres de sortie. Le côté serveur d'une interrogation pourrait être représenté comme suit:

```
<g> ?<invName: Name> !<myRef>  ?<inArgs: PList> ?<outArgs: PList>;
( <g> !<termName> !<SomeIRef>  !<outArgs>;  (* ... other behaviour *)
  [ ] (* ... other terminations *))
```

Les autres structures de données considérées ici sont analogues à celles qui sont décrites au § A.1.1.1. Nous verrons au § A.1.1.12 comment ces offres d'événement peuvent être utilisées pour créer des parties de signatures d'interface opération.

A.1.1.5 Flux

Abstraction d'une séquence d'interaction, entre un objet producteur et un objet consommateur donnant lieu à l'acheminement d'informations. Pour les raisons indiquées au § A.1.1.1, les flux ne peuvent être représentés qu'en LOTOS à l'aide de conventions de modélisation informelle. Les flux sont souvent soumis à des exigences temporelles strictes. Une manière d'appliquer cette règle à la production de flux consisterait par exemple à recourir à un processus paramétré par une séquence de structures de données à envoyer, par exemple des éléments de flux qui peuvent être datés au moment où ils sont envoyés. Une manière simple de modéliser cette règle en LOTOS serait par exemple la suivante:

```
process ProduceAction[ g, ...](... toSend: FlowElementSeq, tnow: Nat, rate: Nat ...):noexit:=
        g !<flowName> !<SomeIRef> !<SetTime(tnow+rate,head(toSend))>;
        (*... other behaviour and recurse with FlowElement removed from toSend *)
endproc (* ProduceAction *)
```

Ici les éléments de flux sont envoyés ensemble avec l'indication de l'heure (locale) du moment considéré et de la cadence à laquelle les éléments de flux doivent être produits.

La consommation des éléments de flux est généralement soumise à des exigences différentes. La nécessité de surveiller continuellement les champs d'indication horaire des flux d'information entrants revêt une importance particulière. La consommation d'un flux d'information peut être représentée de manière simple comme suit:

```
process ConsumeAction[ g,...](myRef: IRef, recFlowElements: FlowElementSeq, tnow, rate: Nat...) :noexit:=
        g ?<flowName: Name> !myRef ?<inFlowElement: FlowElement>;
        (* check temporal requirements of inFlowElement are satisfied then *)
        (* display FlowElement and recurse with time incremented *)
        (* or recurse with FlowElement added to received FlowElements and time incremented *)
endproc (* ConsumeAction *)
```

A.1.1.6 Interface signal

Comme il n'existe aucun moyen direct, en LOTOS, d'établir une distinction formelle entre un signal et tout autre événement LOTOS, on ne peut déterminer qu'une interface donnée est une interface signal que de manière informelle en modélisant les événements LOTOS utilisés pour représenter les signaux différemment de tout autre événement. Un exemple de la manière dont une signature d'interface signal pourrait être modélisée en LOTOS est donné au § A.1.1.11.

A.1.1.7 Interface opération

Comme il n'existe aucun moyen direct, en LOTOS, d'établir une distinction formelle entre une opération et tout autre événement LOTOS, on ne peut déterminer qu'une interface donnée est une interface opération que de manière informelle en modélisant les événements LOTOS utilisés pour représenter les opérations différemment de tout autre événement. Un exemple de la manière dont une signature d'interface opération pourrait être modélisée en LOTOS est donné au § A.1.1.12.

A.1.1.8 Interface flux

Comme il n'existe aucun moyen direct, en LOTOS, d'établir une distinction formelle entre un flux et tout autre événement LOTOS, on ne peut déterminer qu'une interface donnée est une interface de flux que de manière informelle en modélisant les événements LOTOS utilisés pour représenter les flux différemment de tout autre événement. Un exemple de la manière dont une signature d'interface flux pourrait être modélisée en LOTOS est donné au § A.1.1.13.

A.1.1.9 Modèle d'objet de traitement

En LOTOS, un modèle d'objet de traitement est représenté par une définition de processus à laquelle est associé un ensemble de modèles d'interface de traitement que l'objet peut instancier et par une spécification de comportement, c'est-à-dire une expression de comportement qui n'est pas composée d'événements modélisés sous forme de signatures de signal, de signatures de flux ou de signatures d'opération. Par ailleurs, une forme quelconque de contrat d'environnement devrait être modélisée dans le cadre de la définition de processus. Toutefois, le LOTOS n'est pas doté de toutes les fonctions nécessaires à la modélisation intégrale de contrats d'environnement. Il est possible de modéliser certaines fonctions d'un contrat d'environnement à l'aide d'un type de donnée *Act One*. Ce type doit être indiqué sous la forme d'un paramètre formel dans la liste des paramètres de valeur de la définition de processus.

A.1.1.10 Modèle d'interface de traitement

Modèle d'interface signal, modèle d'interface flux ou modèle d'interface opération.

A.1.1.11 Signature d'interface signal

En LOTOS, une signature d'interface signal est représentée par une définition de processus, de telle sorte que toutes les offres d'événement qui doivent être synchronisées avec l'environnement pour se produire soient modélisées sous forme de signatures de signal. L'occurrence de ces offres d'événement se traduit par l'établissement d'une communication unidirectionnelle entre un objet initiateur et un objet répondeur. Sur le plan structurel, une signature de signal est analogue à une invocation d'annonce (ou de terminaison associée à une interrogation), c'est-à-dire qu'elle se compose d'un nom (pour le signal), d'une séquence de paramètres associés au signal et d'une indication de causalité. Comme tous les événements en LOTOS sont atomiques, il n'existe aucune distinction intrinsèque entre les événements modélisés sous forme d'annonces et les événements modélisés sous forme de signaux.

Les signatures d'interface signal diffèrent toutefois des signatures d'interface opération en ceci qu'elles ne requièrent pas qu'une causalité soit conférée à toute l'interface. Les signatures d'interface signal peuvent au contraire contenir des signaux avec des causalités d'initiateur ou de répondeur. Il s'ensuit que nous pouvons modéliser une signature d'interface signal en LOTOS comme suit:

```
process SignalIntSig[ g... ](myRef: IRef, known: IRefs...):noexit:=
  g !<sigName> !<SomeIRef> !<pl>;                    ...(* other behaviour *)
    [ ]... (* other initiating actions *)
  [ ]
  g ?<sigName: Name> !myRef ?<inArgs: PList>;
            ([ not(makeOp(sigName,inArgs) IsIn getOps(myRef))] -> ...(* unsuccessful behaviour *)
            [ ]
            [ makeOp(sigName,inArgs) IsIn getOps(myRef) ] ->     ...(* successful behaviour *) )
  [ ]... (* other responding actions *)
endproc (* SignalIntSig *)
```

Nous constatons ici qu'une interface signal se compose d'un ensemble d'offres d'événement. Ces offres d'événement peuvent modéliser des signaux sortants, pour celles d'entre elles qui comportent un point d'exclamation (**!**) précédant le nom du signal et la liste de paramètres, ou des signaux entrants, pour celles d'entre elles qui comportent un point d'interrogation (**?**) précédant le nom du signal et la liste de paramètres. Dans le cas de signaux entrants, il est possible de vérifier que le signal entrant est bien un des signaux qui sont attendus, c'est-à-dire qu'il fait partie de l'ensemble de signaux autorisés associés à cette référence d'interface.

NOTE – Ce fragment de spécification requiert que le processus soit instancié avec au moins une porte qui corresponde au point d'interaction auquel l'interface se situe. Le processus devrait aussi être instancié avec un ensemble de références d'interface et sa propre référence d'interface. Nous constatons ici qu'il est impossible d'écrire des prédicats sur les signaux envoyés. Il faudrait pour cela un niveau d'autorité que nous n'avons pas, garantissant par exemple que *SomeIRef* soit une des références d'interfaces figurant dans l'ensemble de références d'interface connues associé au processus. Il est toutefois possible de soumettre les signaux entrants à des contrôles, le but étant de vérifier que le signal entrant est bien un des signaux associés à cette référence d'interface. Nous notons en outre que nous avons utilisé ici l'opérateur choix pour modéliser la composition des différents signaux. Il est tout à fait possible d'utiliser ici plusieurs autres opérateurs de composition, par exemple l'opérateur d'entrelacement. L'utilisation de la composition avec entrelacement permet la réception de plusieurs signaux entrants avant qu'un signal de réponse ne soit envoyé. Les interfaces étant généralement matérialisées sous une forme quelconque, c'est-à-dire qu'elles offrent des opérations qui peuvent être invoquées plusieurs fois, les commentaires représentant d'autres comportements sont susceptibles de contenir des instanciations de processus récursif. L'utilisation de l'opérateur de choix nous offre une forme de blocage des signaux, c'est-à-dire qu'en cas d'arrivée d'un signal nous devrons d'abord y répondre avant de pouvoir accepter un nouveau signal. Des arguments analogues s'appliquent à tous les autres processus représentant des signatures d'interface de traitement.

A.1.1.12 Signature d'interface opération

En LOTOS, une signature d'interface opération est représentée par une définition de processus, de telle sorte que toutes les offres d'événement qui doivent être synchronisées avec l'environnement pour se produire soient modélisées sous la forme d'une partie de signatures d'opération, en d'autres termes qu'elles représentent toutes des parties d'annonce ou d'interrogation. Nous pouvons modéliser une signature d'interface opération pour un client selon la définition de processus suivante:

```
process OpIntSigClient[ g... ](myRef: IRef, known: IRefs, ...):noexit:=
  g !<invName> !<SomeIRef> !<inArgs>;                ...(* other behaviour *)
    [ ]... (* other announcements *)
  [ ]
  g !<invName> !<SomeIRef> !<inArgs> !<outArgs>;          ...(* other behaviour *)
            (g ?<termName: Name> !myRef ?<outArgs: PList>;
              [ not(makeOp(termName,outArgs) IsIn getOps(myRef))] -> ...(* return error message *)
              [ ]
              [ makeOp(termName,outArgs) IsIn getOps(myRef)] ->     ...(* other behaviour *)
            [ ] ... (* other terminations *))
   [ ] ... (* other interrogations *)
endproc (* OpIntSigClient *)
```

Nous constatons ici qu'une signature d'interface client se compose d'un ensemble d'offres d'événement. Ces offres d'événement peuvent modéliser soit des invocations (annonce ou interrogation) sortantes, pour celles de ces offres qui comportent un point d'exclamation (**!**) précédant le nom de l'invocation et la liste de paramètres, soit des terminaisons entrantes, pour les offres d'événement qui comportent un point d'interrogation (**?**) précédant le nom de la terminaison et la liste de paramètres. Dans le cas de terminaisons entrantes, il est possible de vérifier que la terminaison entrante est bien une des terminaisons prévues, c'est-à-dire qu'elle fait partie de la série de terminaisons autorisées et associées à cette référence d'interface.

La Note du § A.1.1.11 s'applique également aux signatures d'interface opération, sous réserve d'en modifier dûment le texte (remplacement de l'expression "signal entrant" par "invocation", par exemple).

Les signatures d'interfaces opération pour les serveurs peuvent être représentées en LOTOS comme suit:

```
process OpIntSigServer[ g... ](myRef: IRef, known: IRefs, ...):noexit:=
  g ?<invName: Name> !myRef ?<inArgs: PList>;
   ([ not(makeOp(invName,inArgs) IsIn getOps(myRef))] ->        ...(* ignore/other behaviour *)
    [ ]
    [ makeOp(invName,inArgs) IsIn getOps(myRef) ] ->            ...(* other behaviour *)
    [ ]... (* other announcements *))
 [ ]
  g ?<invName: Name> !myRef ?<inArgs:PList> ?<outArgs:PList>;    ...(* other behaviour *)
   ([ not(makeOp(invName,inArgs,outArgs) IsIn getOps(myRef))] -> ...(* return error message *)
    [ ]
    [ makeOp(invName,inArgs,outArgs) IsIn getOps(myRef) ] ->    ...(* other behaviour *)
     g !<termName> !<SomeIref> !resList ;                  ...(* other behaviour *)
     [ ] ... (* other terminations *))
  [ ] ... (* other interrogations *))
endproc (* OpIntSigServer *)
```

Comme les signatures d'interface client, une signature d'interface serveur a une série de références d'interface connues et sa propre référence. Cette dernière référence d'interface sert à vérifier que les invocations d'annonce ou d'interrogation que le serveur reçoit sont bien celles qui étaient attendues, c'est-à-dire qu'elles faisaient partie de l'ensemble d'opérations associé à cette référence d'interface. En cas d'impossibilité de recevoir ces invocations, pour cause par exemple d'inexactitude des paramètres ou d'indisponibilité de l'opération demandée, des comportements de traitement d'erreurs sont mis en œuvre. Dans le cas d'annonces, cette situation pourrait donner lieu à un appel récursif sans que cela modifie la liste des paramètres formels. Il est également possible de recourir ici à un mécanisme de garde qui empêche dès le départ l'événement de se produire. Nous ne le faisons pas car cela pourrait aboutir à des impasses indésirables dans la spécification. Dans le cas d'interrogations, cela se traduirait par le renvoi d'une forme quelconque de message d'erreur.

Comme dans le cas des interfaces opération client, il est possible d'exiger que les messages reçus soient ceux qui étaient attendus. Il est toutefois impossible d'assortir les messages envoyés d'instructions. D'aucuns diront que cette restriction n'est pas nécessairement une mauvaise chose du fait que, pour autant que chaque processus traite de la même manière les messages reçus, les messages envoyés ne devraient pas donner lieu à des impasses imputables au fait, par exemple, que leur format n'est pas compris.

A.1.1.13 Signature d'interface flux

En LOTOS, une signature d'interface flux est représentée par une définition de processus, de telle sorte que toutes les offres d'événement qui doivent être synchronisées avec l'environnement pour se produire soient modélisées sous forme de flux producteurs ou consommateurs. Une telle signature peut être représentée en LOTOS comme suit:

```
process StreamIntSig[ g... ](myRef: IRef, known: IRefSet, fss: FlowElementSeqSet...):noexit:=
          ConsumeAction[ g...](myRef, known, recFlowElements...) [ ]... (* other consume actions *)
           [ ]
           ProduceAction[ g...](myRef, known, FlowElementstoSend, ...) [ ]... (* other produce actions *)
endproc (* StreamIntSig *)
```

Comme dans le cas des interfaces signal, la notion de causalité est appliquée aux modèles d'action individuels dans la signature d'interface flux. Une telle signature contient des ensembles de flux consommateurs ou producteurs d'actions. Chaque signature de flux est représentée par un processus. Ces processus contiennent la référence de l'interface flux à laquelle ils sont associés, un ensemble de références d'interface représentant les références d'interface connues de cette interface ainsi qu'une séquence d'éléments de flux à envoyer (dans le cas de flux producteurs) ou à recevoir (dans le cas de flux consommateurs). Par souci de concision, nous n'indiquons pas ici comment les ensembles de séquences d'éléments de flux qui sont transmis à une signature d'interface flux sont attribués aux flux producteurs dans cette interface. Lorsqu'ils sont instanciés, tous les flux consommateurs sont, naturellement, vides.

La Note au § A.1.1.11 s'applique aussi aux signatures d'interface flux, sous réserve d'en modifier dûment le texte (remplacement de "signal entrant" par "flux consommateur", par exemple).

A.1.1.14 Objet liaison

Objet qui assure une liaison entre un ensemble d'autres objets de traitement. Le fragment LOTOS suivant donne un exemple de la manière dont cette liaison peut être modélisée:

```
process ServerInterface[ g ...](myRef: IRef, known: IRefSet, ...):noexit:=
    g ?bind: Name !myRef ?pl: PList;
            ([ getIRef(pl) IsIn known ] ->    ....(* already bound to server *)
              ServerInterface[ g ...](myRef,known...)
            []
            [ not(getIRef(pl) IsIn known) and not(getOps(getIRef(pl)) IsSubsetOf getOps(myRef)) ]->
                    ...(* operations requested by client not supported by server *)
            ServerInterface[ g ...](myRef,known...)
            [ not(getIRef(pl) IsIn known) and (getOps(getIRef(pl)) IsSubsetOf getOps(myRef)) ] ->
                    ...(* successful behaviour *)
            ServerInterface[ g ...](myRef,Insert(getIRef(pl),known)...)
    []
    ...(* other behaviours restricted to clients in known *)
endproc (* ServerInterface *)
```

Ici, si le client est déjà lié au serveur, nous refusons la demande de liaison et un appel récursif est passé. Il convient de noter que tel n'est pas nécessairement le cas, c'est-à-dire que le même client pourrait être lié au même serveur plusieurs fois simultanément. A chacune de ces liaisons pourraient être associées des propriétés différentes, par exemple différents ensembles d'opérations demandées avec des contraintes différentes. Cela exigerait que l'objet serveur renvoie des références d'interface différentes pour chaque liaison correctement établie. Par exemple, plutôt que d'insérer la référence d'interface client dans l'ensemble *connu* pour des demandes de liaison qui aboutissent, le serveur pourrait produire une référence d'interface qui serait envoyée au client et ajoutée à l'ensemble *connu*.

Si le client n'est pas déjà lié au serveur, c'est-à-dire s'il ne figure pas dans l'ensemble *connu* de références d'interface, on procède à une vérification des opérations associées à la demande du client. Si les opérations demandées ne sont pas disponibles dans le serveur, un comportement erroné est mis en œuvre et un appel récursif est passé. Pour simplifier les choses, nous nous gardons d'examiner ici les questions liées à la vérification du type des paramètres des opérations client et serveur. Pour des raisons analogues, nous ne traitons pas non plus des contrats d'environnement. Par souci de concision, nous n'indiquons pas les opérations *Act One* permettant d'accéder aux références d'interface contenues dans les listes de paramètres (*getIRef*). Nous nous bornons à rappeler que les opérations que le client demande devraient faire partie de l'ensemble d'opérations que le serveur assure.

Enfin, si le client demande des opérations légales, c'est-à-dire qui fassent partie de l'ensemble d'opérations admises par la référence d'interface serveur, cela donne lieu à un comportement fructueux, comme l'envoi d'une réponse favorable au client, ensuite de quoi la référence d'interface client est ajoutée à l'ensemble *connu* d'interfaces. L'appartenance à cet ensemble permet d'avoir accès à l'autre comportement (non spécifié ici) disponible au niveau de cette interface. Cet autre comportement devrait mettre en œuvre l'ensemble d'opérations et de contraintes indiqué par la référence d'interface *myRef*.

A.1.2 Règles de structuration

A.1.2.1 Règles de dénomination

Les règles de dénomination contenues dans le langage de traitement de la Rec. UIT-T X.903 | ISO/CEI 10746-3 ne peuvent être appliquées en LOTOS qu'à condition de suivre des pratiques de modélisation strictes: par exemple, lors de la création de références d'interface opération, *myRef*, par exemple, on veillera à ce que les noms des opérations d'invocation et de terminaison soient uniques, ainsi que les noms des paramètres associés à ces opérations. L'application effective de ces règles pourra alors être assurée à l'aide de filtres permettant de détecter la légalité des structures de données reçues (par transfert de valeurs) à cette interface.

A.1.2.2 Règles d'interaction

A.1.2.2.1 Règles d'interaction signal

Il est toujours de règle en LOTOS qu'un objet offrant une interface signal (ou une interface flux ou une interface opération) puisse uniquement émettre (et répondre à) des signaux (des flux ou des opérations) qui soient des instances de la signature signal (flux ou opération) associée de ce type d'interface signal (flux ou opération). C'est là une caractéristique prédéfinie des règles de synchronisation du LOTOS, c'est-à-dire que seules les offres d'événement dont les notations d'action correspondent (ou se chevauchent) peuvent être synchronisées. Comme indiqué dans la présente Recommandation | Norme internationale, il n'existe aucune notion réelle de causalité en LOTOS et les événements se produisent instantanément, ensemble ou pas du tout. Il n'arrive donc jamais qu'une invocation soit envoyée, puis reçue. L'envoi et la réception d'une invocation sont représentés par l'occurrence d'un événement LOTOS.

A.1.2.2.2 Règles d'interaction pour les flux

Voir § A.1.2.2.1.

A.1.2.2.3 Règles d'interaction pour les opérations

Voir § A.1.2.2.1.

A.1.2.2.4 Règles de paramétrage

En LOTOS, il est possible d'utiliser des sortes *Act One* comme identificateurs d'interface de traitement. Ces identificateurs peuvent être transférés (sous forme de valeurs) dans les interactions d'une spécification donnée. A la suite de ces interactions, ces identificateurs (sortes) peuvent être utilisés dans des offres d'événement ultérieures des objets expéditeur et destinataire, ce qui permet que des interactions se produisent entre l'expéditeur et le destinataire de l'identificateur. Les identificateurs peuvent donc être considérés comme des identificateurs d'interface de traitement.

Il est possible de modéliser en *Act One* différentes représentations d'un identificateur d'interface de traitement donné. La réécriture *Act One* permet d'obtenir une certaine forme d'égalité à cette fin.

En LOTOS, il est possible de faire en sorte que les identificateurs d'interface de traitement identifient la même interface de traitement.

A.1.2.2.5 Flux, opérations et signaux

Si les flux et les opérations doivent impérativement être représentés en termes de signaux, cela exige que des conventions de modélisation appropriées en LOTOS soient adoptées. Les noms des signaux et des opérations ou flux associés pourront, par exemple, être soumis à des vérifications (filtres).

A.1.2.3 Règles de liaison

A.1.2.3.1 Liaison implicite pour les interfaces opération serveur

Un exemple du côté serveur d'une liaison implicite est donné au § A.1.1.14. Un exemple du côté client d'une liaison implicite, qui souhaite invoquer l'opération *SomeOp* offerte par le serveur ayant pour référence *SomelRef* et ce jusqu'à la fin des opérations, pourrait être représenté en LOTOS comme suit:

```
let myRef: IRef = makeIRef(insert(someOp,{})
in ClientInterface[g,...](myRef,Insert(SomeIRef,{},... )
        process ClientInterface[ g, ...](myRef: IRef, known: IRefSet, ...):noexit:=
            g !bind !SomeIRef !pl;                  (*  pl contains myRef, SomeIRef references server *)
            ( g ?reply: Name !myRef ?pl: PList;     (* receive successful bind response *)
              g !someOp !SomeIRef !pl2; stop)       (* invoke someOp (contained in myRef) and terminate *)
    endprocess (* ClientInterface *)
```

A.1.2.3.2 Règles de liaison primitive

Une liaison primitive peut être établie par l'adoption de conventions de modélisation appropriées en LOTOS. Un exemple d'une telle liaison est donné aux § A.1.1.14 et A.1.2.3.1. A noter que le côté client de la liaison primitive ne devrait pas avoir à créer dynamiquement l'interface associée (et par conséquent la référence d'interface) comme indiqué au § A.1.2.3.1. Au contraire, cette interface devrait déjà exister.

A.1.2.3.3 Règles de liaison composite

Une liaison composite peut être représentée en LOTOS par l'adoption des conventions de modélisation appropriées. Ces conventions exigent de spécifier un processus (représentant un objet de liaison) qui accepte (par transfert de valeurs) des ensembles de références d'interface représentant les objets qui souhaitent être liés ensemble. Une fois que toutes les références ont été reçues, l'objet de liaison envoie à l'expéditeur de chacune des demandes de liaison initiales une demande visant à créer des instances de toutes les interfaces référencées. Les interfaces ainsi créées sont ensuite liées les unes aux autres (à l'aide de la liaison primitive comme indiqué aux paragraphes A.1.1.14 à A.1.2.3.1). Une fois la liaison effectuée, des processus sont instanciés (créés) dans l'objet de liaison qui peuvent ensuite servir pour contrôler les interactions des objets en présence dans la liaison composite.

A.1.2.4 Règles de typage

A.1.2.4.1 Règles de sous-typage pour les interfaces signal

Les règles de sous-typage pour les interfaces signal peuvent être mises en œuvre en LOTOS moyennant l'observation de certaines conventions de modélisation. Des exemples de ces conventions sont donnés aux § A.1.1.11 et A.1.1.14.

A.1.2.4.2 Règles de sous-typage pour les interfaces flux

Les règles de sous-typage pour les interfaces flux peuvent être mises en œuvre en LOTOS moyennant l'observation de certaines conventions de modélisation. Des exemples de ces conventions sont donnés aux § A.1.1.13 et A.1.1.14.

A.1.2.4.3 Règles de sous-typage pour les interfaces opération

Les règles de sous-typage pour les interfaces opération peuvent être mises en œuvre en LOTOS moyennant l'observation de certaines conventions de modélisation. Des exemples de ces conventions sont donnés aux § A.1.1.12 et A.1.1.14.

A.1.2.5 Règles de modèle

A.1.2.5.1 Règles de modèle d'objets de traitement

Un objet de traitement peut:

- émettre des signaux ou y répondre en faisant modéliser les offres d'événement de l'expression de comportement qui lui est associée sous forme de signatures de signal avec la causalité appropriée;
- produire ou consommer des flux en modélisant des signatures de flux dans l'expression de comportement qui lui est associée;
- invoquer ou mettre fin à des opérations en faisant modéliser des offres d'événement sous forme de signatures d'interrogation et d'annonce dans l'expression de comportement qui lui est associée;
- instancier des modèles d'interface ou d'objet en introduisant de tels modèles dans l'expression de comportement qui lui est associée;
- lier des interfaces en adoptant une expression de comportement qui rendra possible une liaison entre interfaces;
- accéder à son propre état et le modifier en produisant des événements constituant une partie de l'expression de comportement qui lui est associée;
- s'arrêter (s'autodétruire) en prévoyant une terminaison appropriée du LOTOS dans le cadre de l'expression de comportement qui lui est associée, par exemple arrêt, sortie ou [>;
- produire dynamiquement, ramifier et joindre des activités en combinant différents opérateurs de composition LOTOS dans l'expression de comportement qui lui est associée, par exemple choix, entrelacement et composition parallèle;
- se lier à une fonction de courtage en offrant des offres d'événement pouvant se synchroniser avec une fonction de courtage dans le cadre de l'expression de comportement qui lui est associée. Cela suppose que la spécification LOTOS qui contient l'objet contienne également une spécification de courtage. Il convient de souligner que l'événement LOTOS qui peut représenter une action de courtage peut ne pas être différent de n'importe quel autre événement LOTOS. En d'autres termes, la distinction entre une action de courtage et toute autre action réside simplement dans le fait que l'événement se produit entre le courtier et l'objet. Par conséquent, une action de courtage ne pourra être différenciée de toute autre action qu'au moyen de ses paramètres de notation d'action en LOTOS. Il convient donc d'examiner attentivement les sortes *Act One* utilisées comme identificateurs pour pouvoir faire la distinction entre les différentes actions.

A.1.2.5.2 Instanciation de modèle d'interface de traitement

S'il est vrai que l'instanciation d'un modèle d'interface de traitement crée une nouvelle interface de traitement en LOTOS, cela n'a pas également pour effet d'établir des identificateurs de traitement pour les interfaces ainsi créées. On peut toutefois obtenir cet effet en adoptant certaines conventions de modélisation. Par exemple, on pourra convenir de créer, au moment où des interfaces (processus LOTOS) associées sont créées, des références d'interface d'ingénierie et des structures de données *Act One*.

A.1.2.5.3 Instanciation de modèles d'objet de traitement

En LOTOS, l'instanciation d'un modèle d'objet de traitement est réalisée par l'instanciation de la définition de processus associée, représentant le modèle d'objet. Les références des interfaces créées dans le processus d'instanciation d'objet doivent être conçues comme indiqué au § A.1.2.5.2.

A.1.2.6 Règles de défaillance

En LOTOS, on peut procéder dans une certaine mesure à la modélisation de défaillances en indiquant pour un système donné tous les comportements possibles, c'est-à-dire aussi bien ceux qui aboutissent que ceux qui échouent. Cela

suppose, toutefois, de connaître préalablement tous les comportements possibles, ce qui, selon le type de défaillance, n'est pas toujours possible. En tant que telle, cette méthode de modélisation des défaillances est limitée en ce que les défaillances ici sont prévisibles, alors qu'en général tel ne sera pas le cas.

Les défaillances d'infrastructure qui peuvent se produire pendant une interaction (liaison, sécurité, communication et ressource) peuvent toutes, dans une certaine mesure, être modélisées en LOTOS moyennant l'indication de tous les comportements possibles du système.

A.1.2.7 Règles de portabilité

Le langage LOTOS accepte toutes les règles de portabilité de la Rec. UIT-T X.903 | ISO/CEI 10746-3. Toutefois, la vérification de la relation de sous-typage entre signatures et l'établissement de liaisons avec les interfaces de courtage ne seront possibles que si l'on adopte des conventions de modélisation qui permettent de représenter la signature des interfaces en *Act One* et d'en faire un outil de base pour la vérification des types et pour les opérations de liaison ultérieures.

A noter qu'aucune action donnée en LOTOS ne répond véritablement à la notion d'action de branchement ou de jonction. Ce n'est qu'à l'examen du comportement de spécification que des actions de ramification et de jonction peuvent être identifiées.

A.1.2.8 Conformité et points de référence

Une spécification LOTOS consiste en un système de comportements possibles. De ce fait, il n'est pas possible d'identifier directement des points de référence qui peuvent devenir des points de conformité de programmation, de perception, d'échange ou d'interfonctionnement. Une spécification intègre tous ses points de référence et il appartient au réalisateur de la spécification de les identifier, de les étiqueter et de les tester. Ainsi, un processus de test agissant sur une spécification LOTOS peut être limité à une certaine partie de la spécification, c'est-à-dire à un objet donné ou à une interface donnée. L'identification de cet objet ou de cette interface en tant que point de conformité fait toutefois partie du processus de test et non du processus de spécification.

Il existe de nombreux types différents de conformité possible en LOTOS. Tous rattachent le comportement de la spécification à une forme quelconque de comportement attendu. Une spécification donnée est donc déclarée conforme si elle affiche le comportement correct, c'est-à-dire le comportement attendu dans le processus de test.

A.2 Formalisation du langage de point de vue traitement en SDL

Un système ODP (applications, fonctions ODP) est décrit en SDL, du point de vue traitement, comme une configuration d'objets de traitement. Ces objets de traitement sont des instances de types bloc et de types processus. Ces types (SDL) doivent être dérivés des types processus et des types bloc définis dans le présent amendement. Les objets de traitement entrent en interaction à l'aide d'une infrastructure offrant les fonctions ODP. Le modèle de traitement de l'infrastructure est modélisé par un bloc SDL. Toutes les fonctions ODP apparaissent ainsi, du point de vue traitement, comme des procédures SDL distantes.

Le présent paragraphe indique comment l'utilisation de la technique de description formelle SDL permet d'exprimer les concepts et les règles du langage de traitement. Pour les concepts non intégralement traités, l'utilisation de descriptions textuelles informelles est proposée. L'utilisation d'*italiques* permet de faire la distinction entre les termes ODP et les termes SDL.

En SDL, du point de vue traitement, un système ODP est décrit comme une configuration d'objets de traitement. Ces objets sont des instances de type bloc et de type processus.

A.2.1 Concepts

Les concepts du langage de traitement sont exprimés en SDL-92 conformément à la Recommandation UIT-T Z.100 et à la Recommandation UIT-T Z.105 et conformément aux définitions, règles et principes génériques de la présente Recommandation | Norme internationale.

A.2.1.1 Signal

Un signal peut être représenté par l'occurrence d'une action de sortie de signal SDL (*OUTPUT*<*sdl-signal*>) et par la réception de ce signal <*sdl-signal*> (ou d'un signal <*sdl-signal*> connexe contenant l'information du signal <*sdl-signal*> initial) par le port d'entrée (*inputport*) du processus (*PROCESS*) de destination.

NOTE – Un signal est une action atomique. Bien qu'elle soit modélisée en SDL par une série d'actions (SDL), l'atomicité est garantie car la transmission par un canal et la réception par l'intermédiaire du port d'entrée des récepteurs sont implicites. Une sortie peut être instantanée dans le cas où l'initiateur et le répondeur sont connectés par des canaux sans retard et par des routes de signaux.

A.2.1.2 Opération

Une opération est l'occurrence d'une interrogation ou d'une annonce.

A.2.1.3 Annonce

Une annonce est une séquence d'actions, modélisée par l'occurrence des actions suivantes:

- sortie (*OUTPUT*) d'un signal <*sdl-signal*> par un processus (*PROCESS*) d'interface d'un objet client;
- entrée (*INPUT*) de ce signal <*sdl-signal*> ou d'un signal <*sdl-signal*> connexe contenant l'information du signal <*sdl-signal*> initial, par un processus (*PROCESS*) d'interface de l'objet serveur (**Invocation**).

NOTE – Le début de la fonction que doit exécuter le serveur est modélisé par la transition déclenchée par l'entrée (*INPUT*).

La structure d'une annonce est indiquée dans le Tableau A.1.

Tableau 1 – Annonce (côté client et côté serveur)

```
PROCESS TYPE Client
INHERITS OperationInterface;
...
OUTPUT servicel(p1,p2,p3) TO Serverl

...
ENPROCESS TYPE Client
```

```
PROCESS TYPE Server
INHERITS OperationInterface;
ADDING GATE InterfaceSignature
        ADDING IN WITH servicel;

...
STATE bound;
        INPUT service1(a1,a2,a3);
        TASK 'perform required function';
        NEXTSTATE-;

...
ENDPROCESS TYPE Server
```

A.2.1.4 Interrogation

Une interrogation est une interaction entre un objet client et un objet serveur comprenant les actions suivantes:

- sortie (*OUTPUT*) d'un signal <*sdl-signal*> par un processus (*PROCESS*) d'interface d'un objet client;
- entrée (*INPUT*) de ce signal <*sdl-signal*> (ou d'un signal <*sdl-signal*> connexe contenant l'information du signal <*sdl-signal*> initial) par un processus *(PROCESS*) d'interface de l'objet serveur (**Invocation**) suivi de:
- exécution éventuelle de la fonction demandée;
- sortie (*OUTPUT*) d'un signal <*sdl-signal*> par un processus (*PROCESS*) d'interface de l'objet serveur; et
- entrée (*INPUT*) de ce signal <*sdl-signal*> (ou d'un signal <*sdl-signal*> connexe contenant l'information du signal <*sdl-signal*> initial) par un processus (*PROCESS*) d'interface de l'objet client (**Termination**).

La structure d'une interrogation est indiquée dans le Tableau A.2.

Tableau A.2 – Interrogation utilisant des signaux SDL

```
PROCESS TYPE Client
INHERITS OperationInterface;
...
OUTPUT servicel(p1,p2,p3) TO Serverl;
NEXTSTATE waitService1;
STATE waitService1
        INPUT service1Term1;
                ...
        INPUT service1Term2;

                ...
        SAVE*;
...
ENPROCESS TYPE Client
```

```
PROCESS TYPE Server
INHERITS OperationInterface;
        ADDING GATE InterfaceSignature
        ADDING IN WITH servicel;
...
STATE bound;
        INPUT service1(a1,a2,a3);
        TASK 'perform required function';
        OUTPUT service1term1 TO SENDER;
                NEXTSTATE-;

...
ENDPROCESS TYPE Server
```

Des interrogations synchrones peuvent être modélisées par des procédures distantes (*REMOTE PROCEDURES*) renvoyant des valeurs. Le client appelle une procédure distante (*REMOTE PROCEDURE*) du processus (*PROCESS*) d'interface de l'objet serveur. Le modèle de remplacement SDL des procédures distantes (*REMOTE PROCEDURES*) pourvoit aux besoins de continuité de l'interrogation.

La structure d'une interrogation synchrone est indiquée dans le Tableau A.3.

Tableau A.3 – Interrogation utilisant des procédures distantes SDL

```
PROCESS TYPE Client
INHERITS OperationInterface;
IMPORTED PROCEDURE service1;
...
CALL servicel(par1,par2,par3) TO Serverl;
...
ENPROCESS TYPE Client
```

```
PROCESS TYPE Server
INHERITS OperationInterface;
ADDING
EXPORTED PROCEDURE service1 REFERENCED;
...
STATE bound;
        INPUT PROCEDURE service1(p1,p2,p3);
                NEXTSTATE-;

...
ENDPROCESS TYPE Server
```

A.2.1.5 Flux

Un flux est une abstraction d'un ensemble d'interactions. Il est modélisé en SDL côté producteur par un signal continu (*continuous signal*), comme indiqué dans le Tableau A.4.

Tableau A.4 – Flux (modélisé par un signal)

```
/* Producer*/

...
PROVIDED Available(Data);
        OUTPUT Frame(GetFrame(Data)) TO BoundTo;
        NEXTSTATE-;
...
```

```
/* Consumer*/

...
STATE Receive;
        PRIORITY INPUT Frame(Data);
                ...;
        NEXTSTATE-;
        INPUT NONE;

                ...;
        NEXTSTATE-;
```

Une nouvelle abstraction peut être fondée sur le concept des valeurs *IMPORTED* et *EXPORTED*.

A.2.1.6 Interface signal

Instance de processus (*PROCESS*), qui communique uniquement avec des signaux (*SIGNALS*) (SDL) par l'intermédiaire de canaux (*CHANNELS*) sans retard. Le processus (*PROCESS*) est une instanciation d'un sous-type d'un modèle d'interface signal (SignalInterfaceTemplate).

A.2.1.7 Interface opération

Interface dans laquelle toutes les interactions sont des opérations. Le processus (*PROCESS*) est une instanciation d'un sous-type d'un modèle d'interface opération (OperationalInterfaceTemplate).

A.2.1.8 Interface flux

Instance de processus (*PROCESS*) qui communique uniquement avec des signaux (*SIGNALS*) (SDL) par l'intermédiaire de canaux (*CHANNELS*) ou au moyen des variables *EXPORTED* ou *IMPORTED*. Le processus (*PROCESS*) est une instanciation d'un sous-type d'un modèle d'interface (flux StreamInterfaceTemplate).

A.2.1.9 Modèle d'objet de traitement

Modèle d'objet pour un objet de traitement. En SDL, un tel modèle est représenté sous la forme d'une définition de bloc (*BLOCK*) fondée sur le type, où le type de bloc (*BLOCK TYPE*) est une spécialisation du modèle d'objet de traitement (ComputationalObjectTemplate) type de bloc (*BLOCK TYPE*). Le concept de contrat d'environnement n'étant pas admis en SDL, il convient d'utiliser à la place un texte informel.

Toutes les portes d'entrée/de sortie d'un modèle d'objet de traitement (ComputationalObjectTemplate) doivent être connectées à des instanciations de sous-types des processus de modèle d'interface (InterfaceTemplate).

La structure d'un modèle d'objet de traitement (*BLOCK TYPE*) est indiquée dans le Tableau A.5.

Tableau A.5 – Modèle d'objet de traitement (ComputationalObjectTemplate) de type bloc (BLOCK TYPE)

```
BLOCK TYPE ComputationalObjectTemplate;
VIRTUAL PROCESS TYPE SignalInterfaceTemplate REFERENCED;
VIRTUAL PROCESS TYPE StreamInterfaceTemplate REFERENCED;
VIRTUAL PROCESS TYPE OperationInterfaceTemplate REFERENCED;
VIRTUAL PROCESS TYPE BehaviourTemplate REFERENCED;
PROCESS LocalBehaviour(1,) : BehaviourTemplate;
/*
        GATE Definitions
        SIGNALROUTE Definitions
        PROCESS Definitions
        have to be added in specializations of this TYPE
/*
ENDBLOCK TYPE;
```

A.2.1.10 Modèle d'interface de traitement

Modèle d'interface signal, modèle d'interface flux ou modèle d'interface opération. Le concept de contrat d'environnement n'étant pas admis en SDL, il convient d'utiliser à la place un texte informel.

Un modèle d'interface signal (SignalInterfaceTemplate) est représenté par une définition de processus (*PROCESS*) fondée sur le type, où le type de processus (*PROCESS TYPE*) est au moins une spécialisation du modèle d'interface signal de type processus (*PROCESS TYPE*), comme indiqué dans le Tableau A.6. Le type de processus (*PROCESS TYPE*) ne doit avoir qu'une porte (*GATE*) connectée à l'extérieur du bloc (*BLOCK*) environnant. Cette porte (*GATE*) représente la signature de l'interface signal. Toute communication vers l'extérieur du bloc (*BLOCK*) doit être fondée sur l'échange de signaux (*SIGNAL*) à travers cette porte (*GATE*). Il ne doit exister aucune sortie (*OUTPUT*) vers l'extérieur du bloc (*BLOCK*) environnant dans l'état (*STATE*) non lié. Le comportement est spécifié par le graphique de processus du Tableau A.6.

Tableau A.6 – Modèle d'interface signal (SignalInterfaceTemplate) de type processus (PROCESS TYPE)

```
PROCESS TYPE SignalInterfaceTemplate;
GATE InterfaceSignature IN FROM ATLEAST SignalInterfaceTemplate
        OUT TO ATLEAST SignalInterfaceTemplate;
DCL BoundTo PId;
                START VIRTUAL; NEXTSTATE unbound;
                STATE unbound;  INPUT VIRTUAL Bind(BoundTo); NEXTSTATE bound;
                                INPUT VIRTUAL *;   NEXTSTATE -;
                STATE bound;    INPUT VIRTUAL UnBind;   NEXTSTATE unbound;
                                INPUT VIRTUAL *;   NEXTSTATE -;
                STATE*;         INPUT VIRTUAL Delete;  STOP;
ENDPROCESS TYPE;
```

Un modèle d'interface flux est représenté par une définition de processus (*PROCESS*) fondée sur le type, où le type de processus (*PROCESS TYPE*) est au moins une spécialisation du modèle d'interface flux (StreamInterfaceTemplate) de type processus (*PROCESS TYPE*), comme indiqué dans le Tableau A.7. Le type de processus (*PROCESS TYPE*) ne doit avoir qu'une porte (*GATE*) connectée à l'extérieur du bloc (*BLOCK*) environnant. Cette porte (*GATE*) représente la signature de l'interface flux. Toute communication vers l'extérieur du bloc (*BLOCK*) doit être fondée sur l'échange de signaux (*SIGNAL*) à travers cette porte (*GATE*) par des signaux continus (*continuous signals*) (flux) dans l'état (*STATE*) lié.

Le comportement est spécifié par le graphique de processus du Tableau A.7.

Tableau A.7 – Modèle d'interface flux (StreamInterfaceTemplate) de type processus (PROCESS TYPE)

```
PROCESS TYPE StreamInterfaceTemplate;
                    INHERITS SignalInterfaceTemplate
                    GATE InterfaceSignature ADDING IN FROM ATLEAST BinderStreamInterfaceTemplate
                              OUT TO ATLEAST BinderStreamInterfaceTemplate;
ENDPROCESS TYPE;
```

Un modèle d'interface opération est représenté par une définition de processus (*PROCESS*) fondée sur le type, où le type de processus (*PROCESS TYPE*) est au moins une spécification du modèle d'interface opération (OperationInterfaceTemplate) de type processus (*PROCESS TYPE*), comme indiqué dans le Tableau A.8. Le type processus (*PROCESS TYPE*) ne doit avoir qu'une porte (*GATE*) connectée à l'extérieur du bloc (*BLOCK*) environnant. Cette porte (*GATE*) représente la signature de l'interface opération. Toute communication vers l'extérieur du bloc (*BLOCK*) doit être fondée sur l'échange de signaux (*SIGNAL*) à travers cette porte (*GATE*) ou par des procédures distantes (*REMOTE PROCEDURES*). Le comportement est spécifié par le corps de processus (*PROCESS*) dans le Tableau A.8.

Tableau A.8 – Modèle d'interface opération (OperationInterfaceTemplate) de type processus (PROCESS TYPE)

```
PROCESS TYPE OperationInterfaceTemplate;
                    INHERITS SignalInterfaceTemplate
                    GATE InterfaceSignature ADDING IN FROM ATLEAST OperationInterfaceTemplate
                              OUT TO ATLEAST OperationInterfaceTemplate;
ENDPROCESS TYPE;
```

A.2.1.11 Signature d'interface signal

La signature d'une interface signal est indiquée par une définition de porte (*GATE*) du processus (*PROCESS*) correspondant. Cette signature contient l'ensemble de noms de tous les signaux (*SIGNALS*) envoyés/reçus par le processus (*PROCESS*) d'interface et donne une indication de leur causalité (*IN WITH* ou *OUT WITH* respectivement). Une signature de signal est donnée par une définition *SIGNAL* comprenant;

- un nom pour le signal;
- le nombre et les types des paramètres du signal.

Le SDL ne permet pas de dénommer les paramètres; toutefois, ceux-ci sont identifiés par leur position.

En outre, l'ensemble de signaux (*SIGNALSET*) du processus (*PROCESS*) contient l'ensemble de noms de tous les signaux (*SIGNALS*) que le processus (*PROCESS*) peut recevoir.

A.2.1.12 Signature d'interface opération

Une signature d'interface pour une interface opération comprend la signature de chaque opération de l'interface. Elle est indiquée par les définitions de porte (*GATE*) conformément au § A.2.1.10 ou par les définitions de procédure distante et importée ou exportée (*REMOTE* and *IMPORTED* or *EXPORTED PROCEDURE*) [dans le cas des procédures distantes (*REMOTE PROCEDURES*)].

Dans le cas de la procédure distante (*REMOTE PROCEDURE*), la signature d'annonce est représentée par la signature de procédure (*PROCEDURE*). Autrement, elle est indiquée par une définition de signal (*SIGNAL*).

Dans le cas de procédures distantes (*REMOTE PROCEDURES*), la signature de terminaison est représentée par la signature du type de données de la valeur retour de la procédure (*PROCEDURE*). Autrement, elle est indiquée par une définition de signal (*SIGNAL*).

A.2.1.13 Signature d'interface flux

La signature d'interface flux est donnée par une définition de porte (*GATE*), contenant l'ensemble de noms de tous les signaux (*SIGNALS*) envoyés/reçus par le processus (*PROCESS*) d'interface, c'est-à-dire les flux et une indication de causalité comme indiqué au § A.2.1.10.

En outre, l'ensemble de signaux (*SIGNALSET*) du processus (*PROCESS*) contient l'ensemble de noms de tous les signaux (*SIGNALS*) que le processus (*PROCESS*) peut recevoir.

Dans le cas de procédures distantes, celles-ci doivent être spécifiées dans le processus (*PROCESS*) d'interface en tant que procédures exportées (*EXPORTED*) ou importées (*IMPORTED*). La clause *REMOTE* limite la visibilité des procédures exportées (*EXPORTED PROCEDURES*).

NOTE – Les signatures d'interface complémentaires sont indiquées par les définitions de porte (*GATE*) avec des listes de signaux identiques mais des clauses de sens complémentaires (*IN WITH* et *OUT WITH* respectivement).

A.2.1.14 Objet de liaison

Instance d'un canal (*CHANNEL*). La structure interne d'un objet de liaison est spécifiée par une sous-structure de canal (*CHANNEL SUBSTRUCTURE*).

Les objets de liaison plus complexes devraient être représentés par une configuration de deux canaux (*CHANNELS*) ou plus et une ou plusieurs instances de bloc (*BLOCK*).

NOTE – Certains canaux (*CHANNELS*) en SDL sont indiqués implicitement, par exemple les canaux (*CHANNELS*) assurant l'acheminement de la communication au moyen de procédures exportées (*EXPORTED PROCEDURES*) et de valeurs exportées (*EXPORTED*). L'utilisation d'une configuration constituée d'un bloc (*BLOCK*) et de deux canaux (*CHANNELS*) ou plus permet d'établir des liaisons entre plus de deux interfaces et avec une ou plusieurs interfaces de commande.

A.2.2 Règles de structuration

En SDL, une spécification de traitement décrit la décomposition fonctionnelle d'un système ODP dans des termes transparents à la distribution, à savoir:

- une configuration de blocs (*BLOCK*S) et de canaux (*CHANNELS*)*;*
- des actions internes modélisées par des processus (*PROCESSES*) locaux qui ne communiquent pas vers l'extérieur du bloc (*BLOCK*)*;*
- des interactions modélisées par des processus (*PROCESSES*) d'interface.

En SDL, une spécification de traitement est limitée par les règles du langage de traitement et par la sémantique du SDL.

L'ensemble initial d'objets de traitement est indiqué par l'ensemble des blocs (*BLOCK*S) et de processus (*PROCESSES*) contenus dans ces objets. Un nombre initial peut être spécifié pour chaque type différent de processus (*PROCESS*). Les modifications apportées à la configuration sont exprimées dans la spécification de comportement.

A.2.2.1 Règles de dénomination

La sémantique statique du SDL garantit les règles suivantes:

- les noms de signaux sont distincts dans n'importe quelle signature d'interface signal, car un signal (*SIGNAL*), ne peut figurer qu'une seule fois dans une définition de porte (*GATE*);
- les noms d'opérations sont distincts dans n'importe quelle signature d'interface opération, car un processus (*PROCESS*) ne doit pas exporter ou importer deux procédures (*PROCEDURES*) différentes sous le même nom;
- les paramètres sont identifiés de façon univoque par leur position. Il n'existe pas de dénomination pour les paramètres de signal (*SIGNAL*);
- les identificateurs d'interface de traitement sont mappés avec les *PIds* (identificateurs de processus) et sont donc uniques dans toutes les spécifications.

A.2.2.2 Règles d'interaction

Pour satisfaire à la contrainte selon laquelle des interactions au niveau d'une interface non liée causent une défaillance de l'infrastructure, toutes les actions de sortie (*OUTPUT*) doivent être qualifiées par les clauses *VIA* ou *TO*. Tous les signaux (*SIGNALS*) envoyés vers l'extérieur d'un bloc (*BLOCK*) d'objets de traitement doivent être envoyés par un processus (*PROCESS*) d'interface et acheminés par une porte (*GATE*) de ce processus.

Tous les appels de procédure distante (*Remote Procedure Calls*) doivent être qualifiés par TO. Toutes les défaillances doivent être explicitement spécifiées, du fait que SDL n'offre aucun mécanisme de traitement des défaillances. Le comportement est indéterminé après qu'une erreur s'est produite.

A.2.2.2.1 Règles d'interaction pour les signaux

Les règles d'interaction pour les signaux sont garanties par la sémantique du SDL. Un processus (*PROCESS*) d'interface signal ne peut émettre/recevoir que les signaux (*SIGNALS*) qui sont spécifiés pour ce processus.

A.2.2.2.2 Règles d'interaction pour les flux

Les interfaces flux sont fondées sur l'échange de signaux (*SIGNAL*). Par conséquent, les règles d'interaction pour les flux sont garanties par la sémantique du SDL. Un processus (*PROCESS*) d'interface flux ne peut produire/consommer que les flux qui sont spécifiés pour ce processus.

A.2.2.2.3 Règles d'interaction pour les opérations

Dans le cas de procédures distantes (*Remote PROCEDURES*), les règles d'interaction pour les opérations sont garanties par la sémantique du SDL. Dans le cas de signaux (*SIGNALS*), le spécificateur doit appliquer les règles suivantes:

- pour tous les signaux (*SIGNALS*) modélisant les invocations d'opérations, il faut prévoir des transitions dans tous les états du processus (*PROCESS*) pour traiter ces signaux (*SIGNALS*) (pas de mise au rebut implicite!);
- la transition ou séquence de transitions déclenchée par l'invocation doit prendre fin exactement sur un signal <*sdl-signal*> de sortie (*OUTPUT*), où <*sdl-signal*> est un signal (*SIGNAL*) modélisant une des terminaisons.

L'ordre d'occurrence des signaux de remise d'invocations ou de terminaisons simultanées ne suit pas nécessairement l'ordre d'occurrence des signaux de dépôt d'invocation ou de terminaison correspondants. En SDL, les signaux (*SIGNALS*) simultanés sont classés dans un ordre arbitraire (non déterministe). Il n'y a aucun moyen de décrire directement la durée d'une opération. Cette durée est arbitraire.

A.2.2.2.4 Règles de paramétrage

Les identificateurs d'interface de traitement sont représentés par l'abréviation *PIds*. Il peut s'agir aussi bien de paramètres d'argument que de paramètres de résultat, dans des interactions pour les signaux ou pour les opérations. Les identificateurs peuvent être transmis sous la forme d'un paramètre dans d'autres interactions.

Le destinataire d'un identificateur d'interface de traitement peut utiliser l'identificateur pour s'engager dans des interactions avec l'objet prenant en charge l'interface, pour autant qu'une liaison puisse être établie entre les interfaces.

Les identificateurs de traitement ne prêtent pas à équivoque dans le cadre de la spécification de système (*SYSTEM*). On peut utiliser des synonymes (*SYNONYMS*) pour donner à une interface plusieurs identificateurs.

Il est toujours possible de déterminer si deux identificateurs d'interface de traitement identifient la même interface de traitement. Toutefois, il n'existe aucun moyen de qualifier un *PId* par un type de signature d'interface ou de détecter le type de l'interface auquel le *PId* se réfère.

A.2.2.2.5 Flux, opérations et signaux

Le modèle de remplacement d'opérations et de flux par des signaux est garanti en SDL. Dans le cas de procédures distantes (*REMOTE PROCEDURES*) et de procédures exportées/importées (*EXPORTED/IMPORTED*) un modèle de remplacement analogue est prévu par la norme SDL.

A.2.2.3 Règles de liaison

Une interaction entre objets de traitement n'est possible que lorsque leurs interfaces sont liées au même objet de liaison. Cette condition est garantie par la sémantique SDL du fait que chaque communication entre blocs (*BLOCKS*) en SDL exige que les blocs (*BLOCK*S) soient connectés par l'intermédiaire d'un canal (*CHANNEL*). Chaque sortie (*OUTPUT*) doit utiliser la clause *TO* <*PId*> ou *VIA* <*channel*> ou *VIA* <*gate*>.

A.2.2.3.1 Liaison implicite pour les interfaces opération serveur

En SDL, une liaison implicite est possible pour des blocs (*BLOCKS*) d'objet de traitement qui sont directement connectés par l'intermédiaire d'un canal (*CHANNEL*). Une liaison implicite est toujours établie au moyen d'opérations modélisées selon des procédures distantes (*REMOTE PROCEDURES*). Le modèle de remplacement SDL garantit les règles de liaison implicite pour des interactions entre opérations serveur. Toutefois, l'interface opération client doit être créée explicitement. Le champ d'application d'une procédure exportée (*EXPORTED PROCEDURE*) est limité par une clause de distance (*REMOTE*).

A.2.2.3.2 Règles de liaison primitive

L'établissement d'une liaison primitive exige que les processus (*PROCESSES*) d'interface des deux objets de traitement soient directement connectés par des canaux (*CHANNELS*) comme indiqué dans le Tableau A.9.

Tableau A.9 – Action de liaison

```
/*Initiater*/

...
STATE unbound;
...
TASK
Destination:=CALL Bind(SELF,TDesc) TO Dest;
DECISON Destination=Dest;
FALSE: /*Binding Failure*/
        NEXTSTATE -;
TRUE: TASK BoundTo:= add(BoundTo,Dest)
        NEXTSTATE Bound;
ENDDECISION;
STATE Bound;
...
```

```
/*Destination*/
VIRTUAL EXPORTED PROCEDURE Bind
NEWTYPE PIDSet ATLEAST
        OPERATORS noequality;
                add: PIDSet,PId->PIDSet;
                remove: PIDSet,PId->PIDSet;
ENDNEWTYPE;
DCL BoundTo PIDSet,ThisType TypeDescrType, Failure Boolean>>
FPAR Source PId, Typ TypeDescrType;
RETURNS Dest PId;
START;
        /*type checking*

                NULL: TASK Failure:=true; RETURN NULL;
                ELSE: TASK BoundTo:=add(BoundTo,Source),
                        Failure:= false; RETURN SELF;
        ENDDECISION;
RETURN NULL
ENDPROCEDURE;
STATE unbound;
        INPUT Bind;
                DECISION failure;
                        false: NEXTSTATE Bound;
                ENDDECISION;
                NEXTSTATE -;
```

L'action de liaison est modélisée par une procédure exportée (*EXPORTED PROCEDURE*) conformément au Tableau A.9. Une procédure (*PROCEDURE*) complémentaire est à prévoir pour supprimer la liaison.

A.2.2.3.3 Liaison composite

Afin d'utiliser une liaison composite, il faut spécifier un objet de liaison [type de bloc (*BLOCK TYPE*)]. Ce type de bloc (*BLOCK TYPE*) doit contenir au moins deux processus (*PROCESSES*) d'interface et un processus (*PROCESS*) de comportement local. Cet objet de liaison doit être connecté aux objets en présence dans la liaison par l'intermédiaire de canaux (*CHANNELS*). Les clauses *ATLEAST* peuvent être utilisées pour réunir les préconditions de la liaison.

L'action de liaison composite comprend les étapes suivantes:

- instanciation de l'objet de liaison;
- instanciation des modèles d'interface de l'objet de liaison, qui sont associés à un rôle formel du modèle d'objet de liaison (cette étape peut faire partie de l'instanciation de l'objet);
- liaison primitive des interfaces en présence dans la liaison, avec les interfaces correspondantes de l'objet de liaison;
- instanciation des interfaces de commande, si besoin est.

Les interfaces de commande peuvent être spécifiées et instanciées conformément aux § A.2.1.10 et A.2.2.5.2.

A.2.2.4 Règles de typage

Le SDL n'offre aucun moyen de décrire formellement le concept de type ODP. Les règles de typage de la Rec. UIT-T X.903 | ISO/CEI 10746-3 doivent être utilisées à titre d'indication de style pour le processus de spécification. Les relations de sous-typage requises pour la liaison des interfaces doivent être exprimées sous forme de comportement de l'action de liaison [lieur de processus (*PROCESS Binder*)].

Les règles de typage de modèle peuvent être exprimées au moyen des clauses *ATLEAST*.

NOTE – Les types de données ASN.1 ou *Act One* peuvent être utilisés pour modéliser le concept de typage. Toutefois, il est impossible de vérifier formellement la relation entre l'objet et son type.

A.2.2.5 Règles de modèle

A.2.2.5.1 Règles de modèle d'objet de traitement

Un objet de traitement peut:

- émettre des signaux ou y répondre (*INPUT/OUTPUT*);
- produire ou consommer des flux;
- émettre des invocations d'opération;

- répondre à des invocations d'opération;
- émettre des terminaisons d'opération;
- répondre à des terminaisons d'opération;
- instancier des modèles d'interface (*CREATE <process>*);
- instancier des modèles d'objet (*OUTPUT* CreateObject(<objectname>) *TO* LocalBehaviour);
- lier des interfaces;
- accéder à son propre état et le modifier (*TASK*-actions, *NEXTSTATE*);
- détruire une ou plusieurs de ses interfaces (*OUTPUT* Delete *TO* Interface);
- s'autodétruire (*STOP*);
- produire dynamiquement, ramifier et joindre des activités (*PROCESS* Creation, *SERVICE* decomposition);
- obtenir un identificateur d'interface de traitement pour une instance de la fonction de courtage.

A.2.2.5.2 Instanciation d'interface de traitement

L'instanciation de modèle d'interface de traitement:

- crée un nouveau processus (*PROCESS*) d'interface;
- produit un identificateur d'interface de traitement pour l'interface (*PId*).

L'instanciation est modélisée par le type de processus d'interface *CREATE <interface-process-type>*. Les variables *SELF* du créateur et *OFFSPRING* (descendant) du processus créé contiennent l'identificateur de la nouvelle interface.

A.2.2.5.3 Instanciation de modèle d'objet de traitement

L'instanciation de modèle d'objet de traitement:

- crée un nouveau comportement local de processus (*PROCESS* LocalBehaviour) pour l'objet;
- produit un ensemble (non vide) d'identificateurs pour les processus d'interface initiaux du nouvel objet.

L'instanciation est modélisée par *CALL* CreateObject *TO <object-type>*. Cela crée un nouveau comportement local de *PROCESS* LocalBehaviour. L'instanciation des processus d'interface du nouvel objet peut être incluse dans la procédure de création d'objet CreateObject ou peut faire partie de la phase initiale de transition du comportement local (LocalBehaviour). La procédure de création d'objet (*PROCEDURE* CreateObject) peut être affinée par héritage.

Tableau A.10 – Procédure de création d'objet (CreateObject PROCEDURE)

```
EXPORTED PROCEDURE CreateObject ATLEAST CreateObject
RETURNS ObjectID PId;
        START VIRTUAL;  CREATE THIS;
        RETURN OFFSPRING;
ENDPROCEDURE CreateObject;
```

NOTE – L'approche spécifiée ici est fondée sur l'hypothèse qu'il existera toujours au moins un objet de ce type. Si cela ne peut être garanti, il faut ajouter au bloc de modèle d'objet de traitement (ComputationalObjectTemplate BLOCK) un processus (gestionnaire) spécial ayant pour but de créer de nouvelles instances.

A.2.2.6 Règles de défaillance

Toutes les défaillances de traitement possibles peuvent être explicitement spécifiées. Le SDL n'offre aucun moyen de traiter les défaillances pendant l'exécution d'une spécification. Après l'occurrence d'une erreur (SDL) le comportement ultérieur d'un système est indéterminé.

A.2.2.7 Règles de portabilité

Le SDL satisfait à toutes les exigences des règles de portabilité, sauf en ce qui concerne:

- les garanties d'ordonnancement et de remise pour les annonces [la remise de signaux (*SIGNALS*) est toujours un succès ou un échec];
- la vérification de la relation de sous-typage entre signatures d'interface.

Le SDL offre un modèle de traitement fondé sur les événements, les actions autorisées étant représentées directement dans le cadre du langage (*INPUT, OUTPUT, CREATE, CALL*) et par l'intermédiaire de structures syntaxiques.

A.2.2.8 Conformité et points de référence

Les portes (*GATES*) des processus (*PROCESSES*) d'interface assurent la communication vers l'extérieur de l'objet de traitement et représentent les points de référence.

Par ailleurs, on peut utiliser des diagrammes de séquence de message (MSC, *message sequence charts*) pour spécifier le comportement requis en un point de référence. Il existe aussi des méthodes d'essai permettant de vérifier la conformité entre une spécification SDL et une spécification MSC-UIT.

Une liaison entre le langage de spécification d'interface normalisé CORBA-IDL et le SDL a été mise au point. On peut l'utiliser pour tester la conformité d'un objet aux points de conformité de programmation.

A.3 Formalisation du langage de point de vue traitement en Z

Structures élémentaires associées aux interfaces opération et signal

Pour formaliser les concepts associés au point de vue traitement en Z, il est nécessaire d'introduire des étiquettes [Name] désignant les différents éléments, par exemple les noms d'opérations et de leurs types ODP. Les types ODP qui existent dans le système sont désignés par un identificateur de type [Type].

Les paramètres qui sont associés à des interfaces avec des objets de traitement sont composés d'un nom et d'un type. Il devrait toujours être possible de déterminer le type d'un paramètre dans un contexte donné (par exemple comme indiqué au § 7.2.1 de la Rec. UIT-T X.903 | ISO/CEI 10746-3). Ainsi, Param est introduit sous forme de fonction partielle dans le sens noms vers types ODP dans un tel contexte.

$$Param : Name \pfun Type$$

Cette fonction comporte, dans son domaine, tous les noms de paramètre qui existent dans un contexte donné. Par ailleurs, il est utile d'introduire des séquences de ces paramètres pour pouvoir examiner les ensembles de paramètres associés à des signaux, à des invocations ou à des terminaisons.

$$PList == \mathbf{seq}\ Param$$

Structures élémentaires associées aux interfaces flux

Comme dans le cas de la formalisation LOTOS du point de vue traitement, nous examinons une notion générique de flux constitués d'éléments de flux. Chaque élément d'un flux d'information peut être considéré comme une unité composée de données (lesquelles peuvent être comprimées) que nous représentons par la structure [Data]. Ce modèle peut indiquer la manière dont les informations ont été comprimées, la nature des informations comprimées, etc. En tant que tel, ce modèle n'est pas examiné plus avant ici. De plus, les éléments de flux contiennent un champ d'indication horaire (ts) servant à modéliser l'heure à laquelle l'élément de flux considéré a été envoyé ou reçu, ce qui permet d'introduire le type [Time]. En outre, il arrive souvent, dans le cas de flux multimédias, que des éléments de flux de synchronisation soient nécessaires pour synchroniser le son avec l'image, par exemple. C'est pourquoi nous associons un nom donné (Name) à chaque élément de flux. Ce nom peut ensuite servir à sélectionner un élément de flux donné, selon les besoins. Il en découle que nous pouvons modéliser un élément de flux comme suit:

```
┌─FlowElement──────────────
│label : Name
│data : Data
│ts : Time
└──────────────────────────
```

A.3.1 Concepts

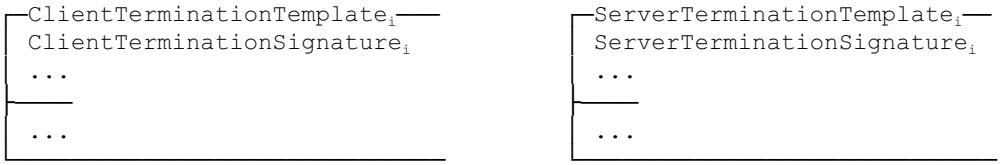
A.3.1.1 Signal

Un signal est une interaction atomique entre un initiateur et un répondeur. Le langage Z n'assurant pas entièrement la fonction d'encapsulage orienté objet, la modélisation d'interactions entre objets impose des restrictions quant aux styles de spécification à suivre. Par exemple, on fera en sorte que les signaux envoyés par les initiateurs aux répondeurs aient des noms de variables (et des types compatibles) appropriés à leurs étiquettes respectives de sortie et d'entrée associées.

On satisfera aux considérations de dénomination en renommant comme il se doit le texte des schémas représentant les signatures de signaux, comme indiqué au § A.3.1.11. Le fragment de langage Z suivant donne un exemple de la manière dont on peut procéder pour ce faire.

```
InitiatingSignalSignature ≙ SignalSignature[pl!/inArgs]
RespondingSignalSignature ≙ SignalSignature[pl?/inArgs]
```

Un modèle de signal initiateur et répondeur peut donc être exprimé comme suit:



Les points sont utilisés ici pour indiquer qu'il y aura vraisemblablement plus d'informations dans la partie déclarative des schémas, concernant par exemple l'état des objets associés aux signaux initiateurs et répondeurs, ainsi que des prédicats pour indiquer les effets des schémas opérationnels apparaissant c'est-à-dire le comportement.

Il convient de noter que l'on peut recourir à l'occultation et à la projection des schémas pour occulter des déclarations qui ne devraient pas être visibles pendant l'interaction, c'est-à-dire qu'elles peuvent être retirées des déclarations et se faire attribuer des quantificateurs existentiels dans la partie prédicative du schéma. Le modèle de signal pour l'interaction proprement dite pourra être construit par concaténation des modèles respectifs des signaux initiateur et répondeur.

```
SignalTemplate ≙ InitiatingSignalTemplate >> RespondingSignalTemplate
```

Le signal proprement dit ne pourra effectivement être émis que si les prédicats associés aux schémas composés se vérifient.

NOTE – Les informations qui sont acheminées comme prescrit au § 8.8 de la Rec. UIT-T X.902 | ISO/CEI 10746-2 sont données par les paramètres de sortie extraits du signal initiateur, c'est-à-dire `pl!`.

A.3.1.2 Opération

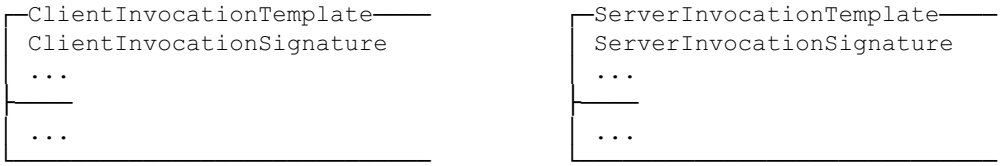
Occurrence d'un schéma d'opération modélisant une interrogation ou une annonce. Voir aussi la NOTE du § A.3.1.4.

A.3.1.3 Annonce

Occurrence d'un schéma d'opération modélisant une invocation entre un client et un serveur. Comme indiqué au § A.3.1.1, le langage Z n'assurant pas entièrement la fonction d'encapsulage orienté objet, on doit adopter des conventions de modélisation pour représenter des systèmes d'objets en interaction. On peut assurer l'application des conventions de dénomination en renommant comme il se doit le modèle d'invocation indiqué au § A.3.1.12. Cette opération peut être représentée comme suit:

```
ClientInvocationSignature ≙ InvocationSignature[pl!/inArgs]
ServerInvocationSignature ≙ InvocationSignature[pl?/inArgs]
```

L'on peut donc exprimer comme suit un modèle d'invocation de client et de serveur:



L'annonce proprement dite pourra alors être modélisée par concaténation des schémas respectifs d'invocation client et d'invocation serveur.

```
InvocationTemplate ≙ ClientInvocationTemplate >> ServerInvocationTemplate
```

L'annonce proprement dite ne pourra effectivement être émise que si les prédicats associés aux schémas composés se vérifient.

NOTE – Le texte du § A.3.1.1 est également applicable aux annonces afin d'expliquer les points dans les schémas qui représentent le comportement non spécifié et l'usage des méthodes d'occultation et de projection de schéma pour fournir une forme d'encapsulage.

A.3.1.4 Interrogation

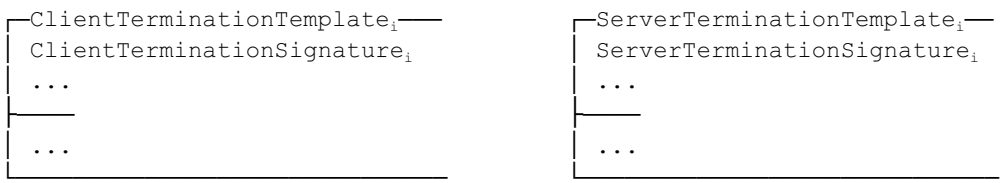
Occurrence d'un schéma d'opération modélisant une invocation entre un client et un serveur, suivie de l'occurrence d'un schéma d'opération associé modélisant une terminaison entre le serveur et le client en question. Comme indiqué au § A.3.1.1, le langage Z n'assurant pas entièrement la fonction d'encapsulage orienté objet, on doit adopter des

conventions de modélisation pour représenter des systèmes d'objets en interaction. On peut assurer l'application des conventions de dénomination en renommant comme il se doit le modèle d'invocation indiqué au § A.3.1.12. Les invocations associées à des interrogations sont modélisées de la même façon que les invocations associées à des annonces, comme indiqué au § A.3.1.3.

Comme indiqué au § A.3.1.1, on peut recourir à l'occultation et à la projection de schémas pour modéliser une forme d'encapsulage. On peut modéliser les terminaisons, ainsi que les conventions de dénomination qu'elles doivent respecter, en renommant les modèles de terminaison (indiqués au § A.3.1.12). Le côté client et le côté serveur d'une terminaison peuvent être représentés comme suit:

$$ServerTerminationSignature_i \triangleq TerminationSignature[pl_i!/outArgs]$$
$$ClientTerminationSignature_i \triangleq TerminationSignature[pl_i?/outArgs]$$

Ici, l'indice inférieur *i* indique qu'il peut y avoir plusieurs de ces (signatures de terminaison) associées à une seule invocation. Un modèle de terminaison de client et de serveur peut donc être formulé comme suit:



L'indice inférieur *i* sert à indiquer qu'il y aura probablement plusieurs modèles de terminaison, dont chacun possédera une signature associée. Ces signatures pourront être différentes.

Les terminaisons proprement dites pourront donc être représentées par concaténation des schémas respectifs des terminaisons de serveur et de client.

$$TerminationTemplate_1 \triangleq ServerTerminationTemplate_1 >> ClientTerminationTemplate_1$$
$$TerminationTemplate_2 \triangleq ServerTerminationTemplate_2 >> ClientTerminationTemplate_2$$
...

Un modèle d'interrogation sous forme d'invocation suivie d'une ou de plusieurs des terminaisons possibles peut donc être représenté comme suit:

$$InterrogationTemplate \triangleq InvocationTemplate >> (TerminationTemplate_1 \vee TerminationTemplate_2 \vee \ldots)$$

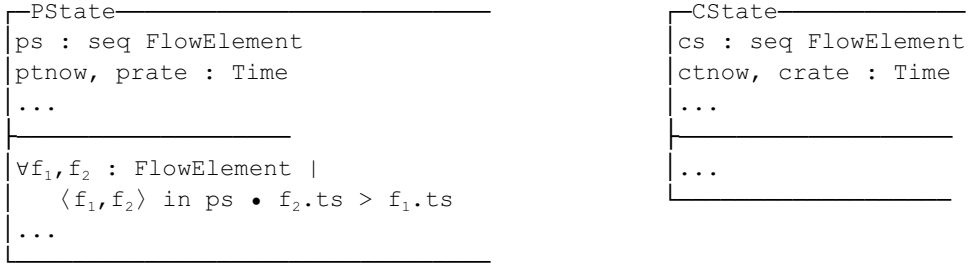
L'interrogation proprement dite ne pourra effectivement être émise que si les prédicats associés aux schémas composés se vérifient.

NOTE 1 – Le texte du § A.3.1.1 est également applicable aux interrogations afin d'expliquer les points dans les schémas qui représentent le comportement non spécifié et l'usage des méthodes d'occultation et de projection de schéma pour fournir une forme d'encapsulage.

NOTE 2 – Ce modèle d'interrogation représente un schéma d'opération unique, c'est-à-dire qu'il est inexact que l'invocation se produise d'abord et qu'elle soit suivie d'une des terminaisons. L'interrogation représente une seule action atomique qui se produit dans son intégralité ou pas du tout – en fonction des prédicats qui lui sont associés. Il convient d'utiliser le commentaire informel associé à la spécification du langage Z pour expliquer l'effet recherché.

A.3.1.5 Flux

La modélisation des flux en langage Z dépend grandement du niveau d'abstraction retenu pour l'examen de la séquence d'interactions représentant le flux. En règle générale, les flux d'informations sont assortis de considérations de synchronisation strictes. Le modèle que nous examinons ici est fondé sur un producteur de flux ayant une séquence d'éléments de données (éléments de flux) à envoyer à un consommateur de flux. Ces éléments de données sont assortis d'un champ d'indication horaire lorsqu'ils sont envoyés par le producteur, lequel champ permet de déterminer leur validité à leur réception par le consommateur. Des exemples de l'état d'un producteur (*Pstate*) et de l'état d'un consommateur (*Cstate*) peuvent être représentés comme suit:



Nous utilisons ici le modèle d'élément de flux indiqué au § A.3.1. Nous constatons que l'état du producteur (comme celui du consommateur) comporte au moins une séquence d'éléments de flux (*ps/cs*), l'heure locale au moment considéré (*ptnow/ctnow*) et le rythme auquel les éléments de flux doivent être envoyés (*prate*) ou acceptés (*crate*). Nous constatons en outre que tous les éléments de flux de la séquence d'éléments de flux associés à un producteur ont des valeurs de champ d'indication horaire croissantes. Ce modèle de l'état du producteur nous permet de modéliser l'envoi d'un élément de flux comme suit:

```
┌─PSendFlowElement──────────────────────────────────────────────
│ΔPState
│f!: FlowElement
│...
├──────────────────
│ps ≠ < > ∧ prate′ = prate ∧
│ps′ = tail ps ∧ ptnow′ = ptnow + 1 / prate ∧
│f! == (μ FlowElement | data = (head ps).data ∧
│             ts = ptnow′ ∧ label = (head ps).label) ...
└──────────────────────────────────────────────────────────────
```

Plusieurs observations s'imposent ici. L'envoi d'un élément de flux supprime celle-ci de la séquence des éléments de flux à envoyer. Le rythme d'envoi des éléments associés au flux, au moment considéré, est inchangé. L'heure effective à laquelle l'élément de flux a été envoyé dépend du rythme d'envoi des éléments de flux et de l'heure locale au moment considéré.

L'élément de flux effectivement envoyé est le premier de la séquence des éléments de flux à envoyer. Il est daté de la valeur horaire calculée précédemment. Nous constatons ici que l'utilisation de la description définie oblige à apporter une preuve que l'élément de flux envoyé est unique. La modélisation de tous les éléments de flux de la séquence avec des valeurs de champ d'indication horaire croissantes (c'est-à-dire inégales) satisfait à cette obligation. Comme aucun élément de flux de la séquence n'a le même champ d'indication horaire, l'élément de flux envoyé avec l'heure du moment considéré est unique. En outre, nous exigeons comme précondition que la séquence des éléments de flux ne soit pas vide.

Un consommateur peut recevoir un élément de flux avec succès pour autant que les contraintes imposées pour son acceptation soient satisfaites.

```
┌─CGetFlowElementOk─────────────────────────────────────────────
│ΔCState
│f?: FlowElement
│...
├──────────────────
│cs′ = cs^<f?> ∧ crate′ = crate ∧ ctnow′ = ctnow + 1 / crate ∧ ...
└──────────────────────────────────────────────────────────────
```

Par souci de concision, nous n'examinons pas en détail ici les contraintes imposées pour l'acceptation des éléments de flux. Celles-ci peuvent obliger à tenir compte des variations dans les heures auxquelles l'élément de flux est acceptable, par exemple la gigue. Le modèle effectif d'un flux peut maintenant être représenté comme suit:

```
Flow ≙ PSendFlowElement >> CGetFlowElementOk
```

Nous notons ici que ce modèle de flux exige que l'élément de flux envoyé et reçu ait le même nom de base et que les autres variables locales des états du producteur et du consommateur aient des étiquettes différentes. Par souci de concision, nous n'examinons pas les cas d'erreurs associés à l'envoi ou à la réception des éléments d'un flux d'information.

A.3.1.6 Interface Signal

Interface dans laquelle tous les *schémas d'opération* sont modélisés sous forme de signaux. Un exemple du format d'un *schéma d'opération* représentant une signature de signal est donné au § A.3.1.1.

NOTE – La spécification de comportement et le contrat d'environnement associés à une interface donnée devraient être représentés par d'autres structures de données en langage Z, par exemple des schémas représentant l'état des objets en présence dans les interactions au niveau de l'interface. L'instanciation d'un modèle d'interface donné devrait satisfaire à tous les prédicats associés au modèle d'interface.

A.3.1.7 Interface opération

Interface dans laquelle tous les *schémas d'opération* sont modélisés sous forme d'opérations. Un exemple du format des *schémas d'opération* représentant des parties d'une signature d'opération est donné aux § A.3.1.3 et A.3.1.4. Voir aussi la NOTE du § A.3.1.6.

A.3.1.8 Interface flux

Interface dans laquelle tous les *schémas d'opération* sont modélisés sous forme de flux. Un exemple du format des *schémas d'opération* représentant une signature de flux est donné au § A.3.1.5. Voir aussi la NOTE du § A.3.1.6.

A.3.1.9 Modèle d'objet de traitement

Modèle d'objet (voir § 4.4.2.11) comprenant un ensemble de modèles d'interface que l'objet peut instancier, une spécification de comportement et un contrat d'environnement. Il convient de noter que le langage Z est essentiellement une notation uniforme et qu'il n'assure pas par conséquent la prise en charge de la modélisation d'objets directement sous la forme d'une fonction linguistique. Il convient au contraire d'utiliser le commentaire en langage naturel qui devrait être associé à chaque spécification en langage Z pour désigner le texte en langage Z, par exemple les *schémas d'opération*, comprenant l'interface ou les interfaces des objets et la relation entre ceux-ci.

A.3.1.10 Modèle d'interface de traitement

Modèle d'interface pour une interface signal, une interface flux ou une interface opération. Voir aussi la NOTE du § A.3.1.6.

A.3.1.11 Signature d'interface signal

Signature d'interface pour une interface signal. Une signature d'interface signal comprend un ensemble fini de modèles d'action, un pour chaque type de signal de l'interface. Chaque modèle d'action comprend le nom du signal, le nombre, le nom et les types de ses paramètres et une indication de causalité (initiateur ou répondeur) afférente à l'objet qui instancie le modèle. Une signature de signal peut être représentée comme suit:

```
┌─SignalSignature──────────────
│inArgs: PList
└──────────────────────────────
```

Ici, le nom du schéma (`SignalSignature`) est utilisé pour représenter le nom du signal et *inArgs* est utilisé pour représenter le nombre, le nom et le type des paramètres associés à ce signal. L'utilisation de ce schéma pour créer des instances de signatures de signaux initiateurs ou répondeurs, c'est-à-dire avec indication de causalité, est indiquée au § A.3.1.1. Voir aussi la NOTE du § A.3.1.6.

A.3.1.12 Signature d'interface opération

Signature d'interface pour une interface opération. Une signature d'interface opération comprend un ensemble fini d'annonces et d'interrogations, selon le cas, une pour chaque type d'opération de l'interface, ainsi qu'une indication de causalité (client ou serveur) pour l'interface tout entière, afférente à l'objet qui instancie le modèle. Les annonces consistent uniquement en des invocations. Les interrogations consistent en une invocation suivie d'une des terminaisons possibles. Une signature d'invocation peut être représentée comme suit:

```
┌─InvocationSignature──────────
│inArgs: PList
└──────────────────────────────
```

Ici, le nom du schéma est utilisé pour représenter le nom de l'invocation et *inArgs* est utilisé pour représenter le nombre, le nom et le type des paramètres associés à cette invocation. L'utilisation de ce schéma pour créer des instances de signatures d'invocations client ou serveur, c'est-à-dire avec l'indication de causalité associée, est indiquée aux § A.3.1.3 et A.3.1.4. Le prédicat associé à ce schéma est utilisé pour satisfaire aux règles de dénominations, des paramètres, c'est-à-dire que les noms de paramètres sont uniques dans le contexte d'un modèle d'invocation. Voir § A.3.2.1.

Une signature de terminaison peut être représentée comme suit:

```
┌─TerminationSignature─────────
│outArgs: PList
└──────────────────────────────
```

Ici le nom du schéma est utilisé pour représenter le nom de la terminaison et *outArgs* est utilisé pour représenter le nombre, le nom et le type des paramètres associés à cette terminaison. L'utilisation de ce schéma pour créer des signatures de terminaisons client ou serveur, c'est-à-dire avec indication de causalité associée, est indiqué aux § A.3.1.3 et A.3.1.4. Il est probable que des prédicats seront associés à ce schéma, par exemple des règles de dénomination etc., comme indiqué au § A.3.2.1. Ces prédicats sont analogues à ceux qui ont été indiqués précédemment pour les modèles d'invocation (avec les modifications de quantification qui s'imposent, par exemple remplacer *inArgs* par *outArgs*). Voir aussi la NOTE du § A.3.1.6.

A.3.1.13 Signature d'interface flux

Signature d'interface pour une interface flux. Une signature d'interface flux comprend un ensemble fini de modèles d'action, un pour chaque type de flux de l'interface. Chaque modèle d'action d'un flux contient le nom du flux, le type d'information du flux et une indication de causalité pour le flux (producteur ou consommateur) afférent à l'objet qui instancie le modèle. Un exemple d'une signature de flux particulière est donné au § A.3.1.5. Les signatures de flux peuvent être identifiées comme faisant partie d'une interface flux donnée au moyen du texte informel associé à chaque spécification en langage Z. Voir aussi la NOTE du § A.3.1.6.

A.3.1.14 Objet de liaison

Objet assurant une liaison avec un ensemble d'autres objets de traitement. Le langage Z n'assurant pas la prise en charge de la modélisation d'objets et de leurs interfaces associées en tant que fonction linguistique, la modélisation d'objets de liaison est généralement limitée. Toutefois, le recours à la théorie des schémas et à des descriptions textuelles informelles permet de modéliser totalement des scénarios d'interaction complexes dans lesquels on peut considérer qu'il existe une forme de liaison. Pour des objets de liaison entre objets client et serveur par exemple, on peut parvenir à ce résultat en modélisant d'autres schémas d'opération (représentant des parties de l'interface avec l'objet de liaison) composés d'invocations client qui seront remises ultérieurement à des serveurs. Cette opération peut être représentée comme suit:

```
InvocationViaBind ≙ (ClientInvocation >> BindInvocation) >> ServerInvocation
```

Le schéma *BindInvocation* devrait avoir des types de données et des étiquettes compatibles pour les variables qui représentent les informations qui sont transmises entre le client et le serveur. Un exemple du schéma *BindInvocation* pour l'invocation client/serveur du § A.3.1.3 se présente comme suit:

```
┌─BindInvocation──────
│inArgs!,inArgs?: PList
│...
├─────────────────────
│...
└─────────────────────
```

Ici, le schéma devrait avoir le même nom de base pour les structures de données transmises par le client (*inArgs?*) que pour celles qui sont transmises vers le serveur (*inArgs!*). Il convient de souligner que la notion de défaillance partielle du schéma *InvocationViaBind* n'existe pas. En d'autres termes, il ne peut arriver que l'invocation client (*ClientInvocation*) et son acceptation par l'objet lieur (*BindInvocation*) aboutissent et que la remise de l'invocation au serveur par l'objet de liaison (*ServerInvocation*) échoue. Ce cas peut être représenté comme suit:

```
InvocationViaBindFail ≙ ClientInvocation >> BindInvocationFail
```

Ici le schéma *BindInvocationFail* peut être modélisé comme suit:

```
┌─BindInvocationFail──────
│inArgs?: PList
│...
├─────────────────────
│...
└─────────────────────
```

En d'autres termes, l'objet de liaison accepte les données transmises par le client (*inArgs?*) mais, pour une raison non spécifiée, ne les remet pas au serveur. Les différentes possibilités d'opérations fructueuses ou infructueuses qu'offre un objet de liaison peuvent être représentées au moyen de la théorie des schémas. En règle générale, on recourt à la réunion logique pour représenter les choix possibles, c'est-à-dire les cas de défaillance.

NOTE – Le comportement associé au schéma *BindInvocation* peut imposer des contraintes aux données qu'il reçoit et qu'il enverra ultérieurement, c'est-à-dire qu'il est possible d'écrire des prédicats sur les valeurs des variables qu'il accepte sous forme d'entrées et qu'il donne sous forme de sorties.

A.3.2 Règles de structuration du point de vue traitement

A.3.2.1 Règles de dénomination

Les règles de dénomination du langage de point de vue traitement peuvent être mises en œuvre de deux manières: par l'écriture de prédicats, comme indiqué au § A.3.1.12 pour les règles de dénomination associées à des paramètres, ou par la détection globale des noms de schémas. Il n'est donc pas possible de déclarer deux schémas d'opération sous les mêmes noms, c'est-à-dire que toutes les actions sont identifiées de manière univoque dans une spécification en langage Z sémantiquement correcte.

A.3.2.2 Règles d'interaction

En règle générale, dans le langage Z, les schémas d'opération (*operation schemas*) ne sont jamais groupés ensemble pour former une nouvelle construction en langage Z, par exemple un schéma représentant l'interface avec un objet. Dans la plupart des cas, cela donnerait un schéma qui n'aurait pas la même structuration modulaire et dont les prédicats, représentant le comportement des différents schémas, risqueraient d'être contradictoires. Il s'ensuit que les règles d'interaction de la Rec. UIT-T X.903 | ISO/CEI 10746-3 ne sont généralement pas admises dans le langage Z.

A.3.2.2.1 Règles d'interaction pour les signaux

La notion de causalité n'existant pas dans le langage Z, il est dénué de sens de dire que les interfaces qui émettent des signaux ont une causalité initiateur ou que celles qui y répondent ont une causalité répondeur. L'étiquette de causalité qui peut être appliquée à une interface donnée l'est de manière informelle. Il peut toutefois arriver que des notions de causalité puissent être traitées dans le commentaire informel associé à chaque spécification en langage Z, pour accompagner des combinaisons de schémas appropriées, par exemple à l'aide du symbole >>.

A.3.2.2.2 Règles d'interaction pour les flux

Voir les § A.3.2.2 et A.3.2.2.1.

A.3.2.2.3 Règles d'interaction pour les opérations

Voir § A.3.2.2 et A.3.2.2.1. Par ailleurs, il convient de noter qu'en règle générale le langage Z ne modélise pas le séquencement ou l'ordonnancement d'actions. Une telle modélisation intervient généralement lorsqu'on apporte des améliorations à une spécification. Par conséquent, le fait qu'un client envoie une invocation qu'un serveur reçoit n'implique pas qu'il existe une construction intrinsèque au langage Z qui oblige ce serveur à envoyer une terminaison appropriée à un stade ultérieur. Au contraire, l'envoi et la réception de l'invocation entre le client et le serveur puis l'envoi et la réception de la terminaison entre le serveur et le client sont généralement modélisés sous la forme d'un seul schéma, comme indiqué dans l'exemple donné au § A.3.1.4. Les actions d'émission et de réception d'une invocation et d'émission et de réception d'une terminaison peuvent aussi être modélisées sous forme de schémas distincts accompagnés d'un texte informel destiné à expliquer la relation entre ces actions.

A.3.2.2.4 Règles de paramétrage

En règle générale, dans le langage Z, les *schémas d'opération* ne sont jamais groupés ensemble pour former une interface avec un objet qui peut être étiqueté puis utilisé pour des interactions avec l'interface auquel cet objet fait référence. De ce fait, le langage Z ne permet pas directement la modélisation de références d'interface de traitement sous forme de paramètres.

A.3.2.2.5 Flux, opérations et signaux

En langage Z, il n'existe aucune distinction intrinsèque entre un flux, une opération ou un signal. Tous ces éléments sont représentés par des schémas d'opération qui peuvent se combiner les uns aux autres de nombreuses manières, par exemple au moyen de la théorie des schémas, selon le comportement du système en cours de spécification. De ce fait, on peut procéder à la modélisation de flux ou d'opérations à l'aide de signaux en faisant en sorte que les schémas représentant les signaux soient munis d'étiquettes et de types de données appropriés associés au schéma correspondant représentant respectivement le flux ou l'opération.

A.3.2.3 Règles de liaison

En règle générale, le langage Z ne permet pas de grouper ensemble des *schémas d'opération* pour former une interface avec un objet qui peut être étiqueté puis utilisé pour des interactions avec une interface auquel cet objet fait référence. C'est pourquoi les règles de liaison de la Rec. UIT-T X.903 | ISO/CEI 10746-3 ne sont généralement pas admises par une spécification en langage Z. En revanche, il est plus fréquent que le langage Z permette une forme de liaison fondée sur des *schémas d'opération* individuels (représentant des signaux, des flux, des invocations ou des terminaisons) se combinant les uns aux autres. Un exemple de cette forme de liaison est donné au § A.3.1.14. Elle permet de faire respecter certaines règles de liaison, moyennant par exemple l'écriture de prédicats destinés à vérifier les types de paramètres transmis. Cependant, le langage Z n'offre aucune fonction limitant de manière générale les possibilités de combinaison des *schémas d'opération*, qui permettrait par exemple de ne combiner entre eux que les *schémas d'opération* ayant un certain nom et des déclarations analogues. Combiner entre eux des schémas incompatibles, (par exemple à l'aide du symbole $\fatsemi$) et des schémas dont les déclarations de variables ont des noms de base analogues mais des types différents, aboutit à une spécification sémantiquement incorrecte.

A.3.2.3.1 Règles de liaison implicite pour les interfaces opération serveur

Voir les § A.3.2.2, A.3.2.3 et A.3.2.2.4.

A.3.2.3.2 Règles de liaison primitive

Voir les § A.3.2.2, A.3.2.3 et A.3.2.2.4.

A.3.2.3.3 Règles de liaison composite

Voir les § A.3.2.2, A.3.2.3 et A.3.2.2.4.

A.3.2.4 Règles de typage

Les règles de typage indiquées pour les interfaces de traitement dans la Rec. UIT-T X.903 | ISO/CEI 10746-3 ne sont généralement pas admises dans le langage Z, pour les raisons indiquées aux § A.3.2.2, 6.2.3 et A.3.2.2.4.

A.3.2.4.1 Règles de sous-typage de signature pour les interfaces signal

Voir § A.3.2.4.

A.3.2.4.2 Règles de sous-typage de signature pour les interfaces flux

Voir § A.3.2.4.

A.3.2.4.3 Règles de sous-typage de signature pour les interfaces opération

Voir § A.3.2.4.

A.3.2.5 Règles de modèle

A.3.2.5.1 Règles de modèle d'objet de traitement

Le langage Z autorise toutes les actions associées aux règles de modèle d'objet de traitement – pour autant qu'il se présente sous une forme textuelle appropriée indiquant, par exemple, les schémas d'opération applicables à la modélisation de signaux initiateurs ou répondeurs – sous réserve des exceptions suivantes:

- la liaison d'interfaces n'est pas entièrement autorisée pour les raisons indiquées aux § A.3.2.3 et A.3.2.2.4;
- la production dynamique, la ramification et la jonction d'activités ne comptent pas parmi les fonctions du langage Z. Toutefois, il est possible d'en modéliser une représentation abstraite;
- l'obtention d'une référence désignant une fonction de courtage n'est pas entièrement autorisée, pour les raisons indiquées aux § A.3.2.3 et A.3.2.2.4;
- le langage Z n'autorise pas la vérification de la relation de sous-typage entre signatures d'interface de traitement, pour les raisons indiquées aux § A.3.2.3 et A.3.2.2.4.

A.3.2.5.2 Instanciation d'interface de traitement

L'instanciation d'un modèle d'interface de traitement peut être réalisée par l'établissement d'une liaison valable pour les variables du texte en langage Z identifiées comme représentant le modèle d'interface. En règle générale, le langage Z ne permet pas de produire un identificateur qui puisse être utilisé pour entrer en interaction avec l'interface créée, et ce pour les raisons indiquées au § A.3.2.3.

A.3.2.5.3 Instanciation de modèle d'objet de traitement

L'instanciation d'un modèle d'objet de traitement peut être réalisée par l'établissement d'une liaison valable pour les variables du texte en langage Z identifiées comme représentant le modèle d'objet.

A.3.2.6 Règles de défaillance

Les types de défaillance observables par un objet sont déterminés par sa spécification de comportement et par son contrat d'environnement. Toute action de traitement décrite au § A.3.2.5 peut échouer.

En langage Z, il existe plusieurs méthodes de modélisation des défaillances. La méthode la plus simple est sans doute de recourir à la réunion logique de schémas, qui consiste à spécifier un schéma satisfaisant ainsi que d'éventuels schémas erronés, lesquels sont ensuite combinés au schéma satisfaisant par réunion logique. La visibilité de ces schémas erronés dépend de la possession d'entrées (**?**) et de sorties (**!**). Ainsi, les défaillances de signaux étant visibles et identiques pour toutes les parties en présence dans l'interaction, une défaillance de signal peut être spécifiée sous la forme d'un schéma avec des sorties qui sont envoyées à toutes les parties en présence dans l'interaction.

En fonction des besoins, les défaillances de schéma pour les flux ou les opérations ainsi que les défaillances associées à l'instanciation d'interface et d'objet peuvent être spécifiées de trois manières: afin qu'elles soient visibles par toutes les parties en même temps; afin qu'elles soient visibles uniquement par une des parties à un moment donné; et afin qu'elles renvoient des paramètres différents aux différentes parties. Toutes ces options peuvent être mises en œuvre grâce à la théorie des schématiques en langage Z.

A.3.2.7 Règles de portabilité

Les actions requises pour la prise en charge d'une norme de portabilité de base ou étendue sont autorisées en langage Z – pour autant que le texte fourni en langage Z représente l'action appropriée – sous réserve des exceptions indiquées au § A.3.2.5.1.

A.3.2.8 Conformité et points de référence

Il existe un point de référence au niveau de chaque interface de chaque objet de traitement. Les réalisateurs se déclarant conformes doivent énumérer les points de référence d'ingénierie correspondant au point de référence de traitement et indiquer quelles transparences et structures d'ingénierie s'y appliquent. De ce fait, les points de référence de traitement deviennent des points de conformité.

Ainsi, pour une spécification donnée en langage Z du point de vue traitement, on peut vérifier la conformité en faisant en sorte que les schémas d'opération associés aux interfaces de la spécification qui ont été identifiées par le réalisateur comme étant des points de conformité soient conformes à la réalisation. A cet effet, on doit ici faire en sorte que les invariants de la spécification ne soient pas violés et que toutes les préconditions et postconditions auxquelles répond la spécification soient également conformes à la réalisation.

A.4 Formalisation du langage de point de vue traitement en ESTELLE

Dans le présent paragraphe, nous indiquons comment la technique de description formelle Estelle peut être utilisée pour interpréter les concepts et les règles du langage de traitement de la Rec. UIT-T X.903 | ISO/CEI 10746-3. Dans le texte qui suit, les *italiques* désignent les concepts Estelle tels qu'ils sont définis dans le document normalisé. L'interprétation est expliquée dans des exemples. Tous les concepts ODP ne pouvant être formellement interprétés en Estelle, on utilise parfois un style désignant implicitement le concept approprié.

A.4.1 Concepts

A.4.1.1 Signal

On entend par "signal" la présentation d'une *interaction* en Estelle par l'intermédiaire d'un *point d'interaction* d'une *instance de module* et sa consommation ultérieure par le destinataire.

A.4.1.2 Opération

Occurrence d'une annonce ou d'une interrogation.

A.4.1.3 Annonce

On entend par "annonce" la présentation d'une *interaction* en Estelle par l'intermédiaire d'un *point d'interaction* d'une *instance de module* qui représente le client, et son utilisation ultérieure par une *instance de module* qui représente le serveur.

A.4.1.4 Interrogation

On entend par "interrogation" la présentation d'*interactions* en Estelle. Une *interaction* – l'**invocation** – est présentée par une *instance de module*, représentant le client. L'autre *interaction* – la **terminaison** – est présentée par une *instance de module* représentant le serveur. La gestion des relations entre invocations et terminaisons doit être effectuée dans les *instances de module*.

A.4.1.5 Flux

Ce concept a deux acceptions en Estelle. Un flux est modélisé au moyen d'une séquence d'*interactions* envoyée par un objet producteur à un objet consommateur. Pour ce faire, on peut rattacher un *module secondaire* au *point d'interaction*, ce qui produit la séquence d'*interactions*. Une *interaction* continue entre les deux modules et les données acheminées peut aussi servir à représenter un flux d'information.

A.4.1.6 Interface signal

On entend par "interface signal" un *point d'interaction* en Estelle. Le *point d'interaction* a un ensemble défini d'*interactions* qui ne peuvent être que des signaux. La définition du *point d'interaction* est fondée sur une *définition de canal* appropriée.

A.4.1.7 Interface opération

On entend par "interface opération" un *point d'interaction* en Estelle. Le *point d'interaction* a un ensemble défini d'*interactions* qui ne peuvent être que des invocations et leurs terminaisons et annonces respectives. La définition du *point d'interaction* est fondée sur la *définition de canal* appropriée.

A.4.1.8 Interface flux

On entend par "interface flux" un *point d'interaction* en Estelle. Le *point d'interaction* a un ensemble défini d'*interactions* qui ne peuvent être que des flux. La définition des *points d'interaction* est fondée sur la *définition de canal* appropriée. Si le flux est modélisé au moyen d'une séquence d'*interactions*, une *instance de module secondaire* est rattachée au *point d'interaction* qui réalise la séquence d'*interactions.*

A.4.1.9 Modèle d'objet de traitement

Un modèle d'objet de traitement peut être modélisé à l'aide d'une *définition d'en-tête de module* qui contient les *points d'interaction* et d'une *définition de corps de module* qui contient le comportement interne et le comportement des modèles d'interface. Il n'existe aucun moyen de spécifier explicitement les contraintes d'environnement en Estelle.

A.4.1.10 Modèle d'interface de traitement

Un modèle d'interface de traitement est modélisé à l'aide d'une *définition de canal* qui contient les signatures d'interface, d'une *définition d'en-tête de module* qui contient les *points d'interaction* appropriés et d'une *définition de corps de module* qui contient la spécification du comportement de l'interface. Les contraintes d'environnement ne peuvent pas être explicitement modélisées en Estelle. Un modèle d'interface de traitement en Estelle est composé d'un *point d'interaction*, d'une *instance de module primaire* qui est rattachée à une *instance de module secondaire* ayant le même *point d'interaction* que le *canal* correspondant. Cette approche permet la réception transparente de diverses *interactions*, car l'*instance de module secondaire* est toujours prête pour la réception.

A.4.1.11 Signature d'interface signal

Une signature d'interface signal est interprétée comme faisant partie de la *définition de canal*. Cette définition contient les rôles associés au *canal*, le nom des *interactions* et les informations qu'elles acheminent, qui sont spécifiées à l'aide d'un ou de plusieurs noms de paramètres et types associés.

A.4.1.12 Signature d'interface opération

Une signature d'interface opération est interprétée comme faisant partie de la *définition de canal.* Cette définition contient les rôles associés au *canal*, le nom de l'*interaction* et les informations qu'elle achemine, qui sont spécifiées à l'aide des noms de paramètres et des types associés. Une signature d'annonce ne contient qu'une seule *interaction* représentant l'invocation. Une signature d'interrogation contient à la fois une *interaction* représentant l'invocation et un ensemble d'*interactions* représentant les terminaisons possibles.

A.4.1.13 Signature d'interface flux

Une signature d'interface flux est interprétée comme faisant partie de la *définition de canal*. Cette définition contient les rôles associés au *canal*, le nom des *interactions* et les informations qu'elles acheminent, qui sont spécifiées à l'aide d'un nom de paramètre et d'un type associé.

A.4.1.14 Objet de liaison

Un objet de liaison est modélisé par une *instance de module* ainsi que par les canaux (*CHANNELS*) par lesquels il est connecté aux objets de traitement.

A.4.2 Règles de structuration

En Estelle, une spécification de traitement décrit la décomposition fonctionnelle d'un système ODP sous forme d'une configuration d'objets de traitement et de liaison, représentés les uns et les autres sous formes d'*instances de module*. Les actions internes d'une *instance de module* sont modélisées sous la forme d'une machine à états finis étendue. Cette machine est réalisée en Estelle à l'aide de transitions, les *interactions* entre objets sont échangées par l'intermédiaire de canaux (*CHANNELS*). Les *interactions* entre *modules* sont asynchrones. Les extrémités d'un *canal* sont des interfaces de

traitement représentées en Estelle sous forme de *points d'interaction*. Les *instances de module* sont toutes des *instances de module secondaire* directes d'une *instance de module* attribuée en fonction de l'*activité-système*. Il s'ensuit que l'exécution de tous les *modules secondaires* n'est pas déterministe. Le *module primaire* ne peut être structuré qu'en *modules* attribués en fonction de l'*activité*. L'*instance de module primaire* permet l'instanciation, la destruction, la connexion et la déconnexion d'*instances de module* représentant des objets de traitement. Les contrats d'environnement des objets ne sont pas modélisés explicitement en Estelle.

A.4.2.1 Règles de dénomination

La sémantique de l'Estelle garantit toutes les règles de dénomination. Les noms, en Estelle, ont le contexte associé suivant: *interaction*, *rôle* et *canal* font partie d'une *définition de canal*. Il n'existe aucune différence syntaxique entre les diverses *interactions*. Les différences sémantiques sont causées par l'utilisation différente des *interactions*. *Canal* et *identificateur de rôle* sont utilisés pour la définition de *points d'interaction* dans la *définition d'en-tête de module*. Un nom d'*interaction* est défini dans le contexte de l'*instance de module*, par combinaison de celle-ci au nom du *point d'interaction* correspondant. Le nom de paramètre d'une *interaction* est un identificateur dans le contexte de la *définition d'interaction*. Le nom d'un *point d'interaction* est défini dans le contexte de toutes les *instances de module* dont la *définition d'en-tête de module* a défini ce point d'interaction.

Le nom d'une *instance de module* est valable dans la *définition de corps de module* dans laquelle il a été défini.

A.4.2.2 Règles d'interaction

En Estelle, les *interactions* ne peuvent être envoyées et reçues qu'aux *points d'interaction*. L'envoi d'une *interaction* par l'intermédiaire d'un *point d'interaction* qui n'est pas connecté entraîne la perte de l'*interaction*. L'*instance de module* émettrice n'est pas informée de son erreur. Si nous devons signaler une défaillance d'infrastructure dans le cas où une *interaction* se produit au niveau d'une interface qui n'est pas liée, toutes les interfaces d'un objet qui ne sont pas liées peuvent être connectées à des *points d'interaction* internes de l'environnement. Ainsi, les *interactions* au niveau de l'une quelconque de ces interfaces peuvent être détectées et signalées comme des défaillances d'infrastructure.

A.4.2.2.1 Règles d'interaction pour les signaux

Lorsqu'une *instance de module* représente un objet qui offre une interface signal d'un type d'interface signal donné, cette interface est représentée par un *point d'interaction* de *canal* et de *définition de rôle* approprié, avec une *instance de module secondaire* qui lui est rattachée et qui modélise le comportement au niveau de l'interface. Cette *instance de module secondaire* ne peut qu'émettre des *interactions* dont le paramètre causalité est mis sur "initiateur". Elle doit contenir une transition avec les *clauses WHEN* appropriées pour la réception de l'une quelconques des *interactions* définies dans le modèle d'interface signal.

A.4.2.2.2 Règles d'interaction pour les flux

Une interface flux est modélisée par un *point d'interaction* auquel est rattachée une *instance de module secondaire* qui modélise le comportement au niveau de l'interface. Un flux peut être représenté par une *interaction* ou par une séquence d'*interactions*, selon les divers types de flux. Si l'objet a un rôle consommateur pour cette interface, l'*instance de module secondaire* doit contenir une transition avec les *clauses WHEN* appropriées à la réception de l'*interaction* pour les flux. Si l'objet a un rôle producteur, l'*instance de module secondaire* doit contenir des déclarations de sortie relatives à l'*interaction* pour les flux.

A.4.2.2.3 Règles d'interaction pour les opérations

Lorsqu'une *instance de module* représente un objet qui offre une interface opération d'un type d'interface opération donné, cette interface est représentée par un *point d'interaction* dont le *canal* et le *rôle* sont définis de manière appropriée et auquel est rattachée une instance de module secondaire qui modélise le comportement au niveau de l'interface. Si l'objet a le rôle client pour cette interface, l'*instance de module secondaire* doit contenir une transition avec la *clause WHEN* appropriée pour la réception de terminaisons et les instructions de sortie appropriées pour la remise des invocations. Si l'objet a le rôle serveur pour cette interface, l'*instance de module secondaire* doit contenir une transition avec la *clause WHEN* appropriée pour la réception d'invocations et les instructions de sortie appropriées pour la remise des terminaisons. En Estelle, il n'existe aucun moyen d'attribuer une durée à une transition ou à une séquence de transitions.

A.4.2.2.4 Règles de paramétrage

Un type de donnée spécifique (Estelle) a été défini pour un identificateur d'interface de traitement. Chaque interface de traitement peut être identifiée par une variable de ce type. Ces variables peuvent être transmises sous forme d'arguments dans des interactions représentant des opérations. Le destinataire d'un tel identificateur d'interface de traitement peut l'utiliser ultérieurement pour établir une liaison avec l'interface référencée.

A.4.2.2.5 Flux, opérations et signaux

Les informations entre modules Estelle sont toujours échangées à l'aide *d'interactions*. Selon cette approche, le type d'information n'est pas visible. Un signal est interprété uniquement à l'aide du modèle d'interaction. L'interprétation des flux et des opérations étant fondée sur la notion d'interaction, on peut utiliser des signaux pour expliquer leurs fonctions spéciales. En Estelle, seuls des *points d'interaction* complémentaires du même canal peuvent être connectés. En partant de l'hypothèse que toutes les *interactions* en un *point d'interaction* sont du même type, une liaison composite entre différents types d'interfaces est impossible en Estelle.

A.4.2.3 Règles de liaison

Afin d'autoriser une *interaction* entre interfaces d'objets de traitement, les *points d'interaction* modélisant ces interfaces sont tous deux connectés aux *points d'interaction* du même objet de liaison. L'objet de liaison est instancié par une action de liaison, qui est invoquée par l'environnement des *instances de module.* Chaque objet de traitement possède un *point d'interaction* externe désigné, par l'intermédiaire duquel il est connecté à (un *point d'interaction* interne de) son environnement, c'est-à-dire à l'*instance de module* environnante. Au niveau de ce *point d'interaction*, l'action de liaison est invoquée avec les paramètres appropriés.

Il convient de noter que ce *point d'interaction* peut être considéré comme un type spécial d'interface au sens où on l'entend en traitement ODP; toutefois, l'action de liaison proprement dite n'est pas une interaction ODP, car elle ne se produit pas entre objets de traitement. Une interface de commande de l'objet de liaison est connectée à une interface de commande correspondante de l'objet client. L'environnement, c'est-à-dire l'*instance de module* environnante, peut vérifier si la relation de sous-typage entre les interfaces à lier est appropriée. Les contrats d'environnement n'étant pas modélisés explicitement (jusqu'à présent), il est impossible de vérifier s'ils sont respectés de manière satisfaisante. Une liaison sans participation explicite de l'environnement ne peut pas être interprétée au moyen de concepts Estelle.

A.4.2.3.1 Règles de liaison implicite pour les interfaces d'opération serveur

Une liaison implicite est requise si l'objet client et l'objet serveur ne sont pas encore liés. Il n'existe aucun moyen, en Estelle, de détecter la non-existence de cette liaison. Le client doit gérer un réseau avec des *points d'interaction* et des serveurs liés.

A.4.2.3.2 Règles de liaison primitive

Dans le cas d'une liaison primitive, aucun objet de liaison n'intervient. Les *points d'interaction* des *instances de module* assurant la modélisation des objets de traitement sont connectés immédiatement. Cette connexion peut être établie durant l'instanciation ou peut être établie par l'environnement après l'invocation de la fonction appropriée. Dans le second cas, une notification sera envoyée si une défaillance se produit.

A.4.2.3.3 Règles de liaison composite

Les actions de liaison composite permettent de lier un ensemble d'interfaces à l'aide d'un objet de liaison prenant en charge la liaison. Un modèle d'objet de liaison peut être interprété, en Estelle, de la même manière qu'un modèle d'objet de traitement. Les préconditions pour une liaison composite peuvent être exprimées par les expressions Estelle suivantes:

- les paramètres d'interface correspondants doivent être du même type; cette condition peut être remplie en Estelle à l'aide d'une *définition de canal* valable;
- les paramètres d'interface correspondants doivent avoir des causalités complémentaires. Cette condition peut être remplie en Estelle à l'aide d'une *définition de canal* valable et du *rôle* approprié;
- le paramètre d'interface correspondant doit être un sous-type du type de signature. Bien que le sous-typage ne soit pas pris en charge en Estelle, cette condition peut être remplie à l'aide d'une *définition de canal* valable.

Une action de liaison composite peut être interprétée à l'aide des constructions Estelle suivantes:

- une instance de module pour un objet de liaison est initialisée par le *module primaire*. L'*en-tête de module*, le *corps de module* et une *variable de module* ont été définis précédemment;
- durant l'instanciation, les *points d'interaction* de l'*instance de module* sont également instanciés;
- l'utilisation de l'*instruction de connexion* permettra d'établir une liaison primitive entre l'objet de liaison et les autres objets de traitement;
- des interfaces de commande ont été instanciées. Leur identificateur peut être renvoyé;
- les fonctions nécessaires au niveau de l'interface de commande peuvent être assurées par l'*instance de module*.

A.4.2.4 Règles de typage

En Estelle, les *points d'interaction* ne peuvent être connectés que s'ils sont définis avec la même *définition de canal.* Il n'existe pas de relation de sous-typage entre *points d'interaction*. Afin de lier des interfaces qui se trouvent dans une relation de sous-typage à l'aide d'un objet de liaison, cet objet de liaison doit avoir des types d'interface différents pour son interface qui joue le rôle client (consommateur) et son interface qui joue le rôle serveur (producteur). La définition récursive d'interfaces n'étant pas autorisée en Estelle, seules les règles de sous-typage simplifiées peuvent être appliquées. Les types de données Estelle sont fondées sur les types de données ISO PASCAL. Par conséquent, les types de sous-intervalle constituent la seule forme de sous-typage autorisée. Les types de sous-intervalle peuvent être définis pour des types de données ordinales, à savoir le nombre entier de type de données et les types énumérés définis par l'utilisateur.

A.4.2.5 Règles de modèle

A.4.2.5.1 Règles de modèle d'objet de traitement

Un objet de traitement peut:

- émettre un signal en exécutant les *instructions de sortie* correspondantes pour l'*interaction* représentant le signal ou répondre à des signaux en exécutant les *clauses WHEN* correspondantes en un *point d'interaction* représentant une interface signal. Les *instructions de sortie* et les *clauses WHEN* sont mises en œuvre dans des *modules secondaires* rattachés au *point d'interaction* du *module primaire*;
- produire des flux en exécutant les *instructions de sortie* correspondantes pour l'*interaction* représentant le flux ou consommer des flux en exécutant les *clauses WHEN* correspondantes en un *point d'interaction* représentant une interface flux. Les *instructions de sortie* et les *clauses WHEN* sont mises en œuvre dans des *modules secondaires* rattachés au *point d'interaction* du *module primaire*. Une boucle peut exister à l'intérieur du *module secondaire* pour lancer la séquence d'*interaction*s modélisant un flux;
- invoquer des opérations en exécutant les *instructions de sortie* correspondantes pour l'*interaction* représentant l'invocation d'opération ou répondre à des invocations d'opération en exécutant la *clause WHEN* correspondante au niveau d'une interface opération. Les *instructions de sortie* et les *clauses WHEN* sont mises en œuvre dans des *modules secondaires* rattachés au *point d'interaction* du *module primaire*;
- terminer des opérations en exécutant l'*instruction de sortie* correspondante pour une *interaction* représentant une terminaison d'opération;
- instancier des modèles d'interfaces en attribuant un *point d'interaction* inutilisé à la nouvelle interface, en instanciant l'*instance de module secondaire* définissant le comportement au niveau de l'interface et en la rattachant au *point d'interaction*;
- instancier directement des modèles d'objets en instanciant une nouvelle instance de la *définition de module* correspondante (seuls des objets internes à l'objet instanciateur peuvent être créés de cette manière), ou indirectement par un échange d'*interactions* avec l'environnement;
- lier des interfaces en invoquant une action de liaison avec l'environnement;
- accéder à son propre état et le modifier en exécutant une transition d'état ou en modifiant des variables;
- s'autodétruire indirectement en réclamant lui-même d'être arrêté par l'environnement;
- produire dynamiquement une activité en instanciant une *instance de module secondaire*;
- se lier à un objet de fonction de courtage en invoquant une action de liaison avec l'environnement.

Il convient de noter que la ramification et la jonction d'activités ne sont pas autorisées en Estelle. Il convient aussi de noter que les interfaces ne peuvent pas être supprimées et que seules les connexions entre points d'interaction peuvent être déconnectées.

A.4.2.5.2 Instanciation de modèle d'interface de traitement

Un *point d'interaction* inutilisé est attribué à la nouvelle interface, une *instance de module secondaire* est créée par le *module primaire* et rattachée au *point d'interaction* et l'identificateur d'interface interne est produit pour l'interface. La création dynamique de *points d'interaction* est impossible en Estelle. Il s'ensuit que chaque *instance de module* est créée avec (pour chaque type d'interface) un ensemble de *points d'interaction* représentant le nombre maximum d'interfaces de ce type existant simultanément. A un instant donné, un *point d'interaction* est inutilisé ou représente une interface (à laquelle il est attribué). Le *module primaire* doit tenir à jour les informations relatives à l'état de chaque *point d'interaction* dans une structure de données appropriée. Les interfaces initiales de l'objet sont instanciées dans la *transition d'initialisation* du *module*. Chaque *module* produit un identificateur d'interface interne pour chacune de ses interfaces. L'identificateur de traitement externe complet est créé par l'environnement de l'objet à l'aide de l'identificateur d'interface interne et d'un *identificateur de module*.

A.4.2.5.3 Instanciation de modèle d'objet de traitement

L'instanciation d'un objet de traitement ne peut être entreprise que par l'environnement qui est modélisé sous la forme d'un module avec pour attribut *activité-système*. Il convient de distinguer deux phases. Tout d'abord, l'*instance de module* est créée. Pour ce faire, on peut utiliser deux formes génériques de l'*instruction d'initialisation*. Dans une de ces formes, des paramètres peuvent être transmis à l'*instance de module* en cours de création. Comme les *variables de module* utilisées dans l'*instruction d'initialisation* doivent être déclarées avant d'être utilisées, la création dynamique d'*instances de module* est impossible en Estelle. Dans une seconde étape, les *points d'interaction* appropriés doivent être connectés à leurs partenaires respectifs.

A.4.2.6 Règles de défaillance

Les signaux, tels que la soumission d'une invocation d'opération, étant modélisés en Estelle par des messages envoyés par l'intermédiaire d'un *point d'interaction*, une défaillance d'infrastructure ne se produit pas automatiquement si l'interface n'est pas liée. Un message envoyé par l'intermédiaire d'un *point d'interaction* qui n'est pas connecté est simplement perdu. Dans une application de l'interprétation Estelle, toutefois, la perte d'un message due à des interfaces non liées peut être détectée et signalée à l'*instance de module* émettrice en tant que défaillance d'infrastructure. La création dynamique de *points d'interaction* ou d'*instances de module* est impossible en Estelle. Il est toujours nécessaire de définir un ensemble avec des variables du type approprié. L'instanciation de modèles échouera si toutes les instances possibles existent déjà.

A.4.2.7 Règles de portabilité

Des modèles d'action peuvent être donnés pour toutes les actions qui peuvent être interprétées en Estelle. Ce langage satisfait à tous les critères des règles de portabilité sous réserve des exceptions suivantes: la ramification et la jonction d'actions ne sont pas autorisées en Estelle, ni les garanties d'ordonnancement et de remise pour les annonces.

A.4.2.8 Conformité et points de référence

Les points de référence sont des *points d'interaction* externes de *modules* Estelle. Des méthodes permettant d'extraire par dérivation des jeux d'essai à partir des spécifications Estelle sont disponibles. Néanmoins, la question de la dérivation de jeux d'essai pour des spécifications multimodule est encore à l'étude. Il n'existe aucune liaison linguistique entre un langage de spécification d'interface normalisé et Estelle. Il existe toutefois une méthode d'essai globale fondée sur les interactions observables dans les protocoles de communication.

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	Réseaux câblés et transmission des signaux radiophoniques, télévisuels et autres signaux multimédias
Série K	Protection contre les perturbations
Série L	Construction, installation et protection des câbles et autres éléments des installations extérieures
Série M	RGT et maintenance des réseaux: systèmes de transmission, circuits téléphoniques, télégraphie, télécopie 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 et protocole Internet
Série Z	Langages et aspects généraux logiciels des systèmes de télécommunication

Genève, 2002