UNION INTERNATIONALE DES TÉLÉCOMMUNICATIONS

UIT-T Z.500

(05/97)

SECTEUR DE LA NORMALISATION
DES TÉLÉCOMMUNICATIONS
DE L'UIT

SÉRIE Z: LANGAGES DE PROGRAMMATION

Méthodes de validation et d'essai

Cadre général des méthodes formelles appliquées aux tests de conformité

Recommandation UIT-T Z.500

(Antérieurement Recommandation du CCITT)

RECOMMANDATIONS UIT-T DE LA SÉRIE Z

LANGAGES DE PROGRAMMATION

TECHNIQUES DE DESCRIPTION FORMELLE	Z.100–Z.199
Langage de description et de spécification (SDL)	Z.100–Z.109
Application des techniques de description formelle	Z.110–Z.119
Diagrammes des séquences de messages	Z.120–Z.129
LANGAGES DE PROGRAMMATION	Z.200–Z.299
CHILL: le langage de haut niveau de l'UIT-T	Z.200–Z.209
LANGAGE HOMME-MACHINE	Z.300–Z.499
Principes généraux	Z.300–Z.309
Syntaxe de base et procédures de dialogue	Z.310–Z.319
LHM étendu pour terminaux à écrans de visualisation	Z.320–Z.329
Spécification de l'interface homme-machine	Z.330–Z.399
QUALITÉ DES LOGICIELS DE TÉLÉCOMMUNICATION	Z.400–Z.499
MÉTHODES DE VALIDATION ET D'ESSAI	**Z.500–Z.599**

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

RECOMMANDATION UIT-T Z.500

CADRE GÉNÉRAL DES MÉTHODES FORMELLES APPLIQUÉES AUX TESTS DE CONFORMITÉ

Source

La Recommandation UIT-T Z.500, élaborée par la Commission d'études 10 (1997-2000) de l'UIT-T, a été approuvée le 6 mai 1997 selon la procédure définie dans la Résolution n° 1 de la CMNT.

AVANT-PROPOS

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

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

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

Dans certains secteurs de la technologie 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

Introduction

De nombreuses spécifications de protocole et de service sont aujourd'hui décrites en termes de notations formelles appelées Techniques de Description Formelle (FDT, *formal description techniques*). Le SDL, LOTOS, Estelle et ASN.1 sont des exemples de FDT normalisées. Il existe également une notation formelle pour des spécifications de suites de tests: la notation combinée arborescente et tabulaire (TTCN, *tree and tabular combined notation*). L'utilisation des FDT présente les avantages suivants:

– elles décrivent les formats et comportements de manière univoque,

– elles fournissent une base de validation rigoureuse incluant les essais de conformité.

La conformité à une norme de protocole ou de service de communication est considérée comme une exigence préalable de l'interopérabilité correcte des systèmes ouverts. Le test de conformité, c'est-à-dire l'évaluation au moyen d'essais de la conformité d'un produit à sa spécification, est un aspect important du développement du produit car il augmente la confiance en une interopérabilité correcte.

La présente Recommandation "Cadre général des méthodes formelles des tests de conformité" (FMCT, *framework on formal methods in conformance testing*), définit la signification de la conformité lorsque des méthodes formelles sont utilisées pour la spécification d'un protocole ou d'un service de communication. Elle est également destinée à servir de guide pour la génération de tests informatisés.

La présente Recommandation définit un cadre pour l'utilisation des méthodes formelles dans les essais de conformité. Elle est destinée aux personnes chargées des implémentations, des tests et des spécifications, impliquées dans des tests de conformité afin de les aider à définir la conformité ainsi que les procédures d'essai d'une implémentation donnée, eu égard à une spécification fournie comme description formelle.

Recommandation Z.500

CADRE GÉNÉRAL DES MÉTHODES FORMELLES APPLIQUÉES AUX TESTS DE CONFORMITÉ

(Genève, 1997)

1 Domaine d'application

La présente Recommandation s'applique lorsqu'il existe une spécification formelle de protocole ou de service de communication à partir de laquelle une suite de tests de conformité doit être développée. Elle peut servir de guide à la procédure manuelle ainsi qu'au développement d'outils pour la génération de tests élémentaires assistée par ordinateur.

La présente Recommandation définit un cadre général et ne prescrit aucune méthode particulière de génération de tests élémentaires ni ne prescrit de relation de conformité spécifique entre une spécification formelle et une implémentation donnée. Elle complète la norme mixte UIT-T/ISO "Cadre général et méthodologie des tests de conformité" (CTMF) [ISO/CEI 9646], qui s'applique à une large gamme de produits et de spécifications, y compris des spécifications écrites en langage naturel. Le cadre FMCT interprète les concepts de test de conformité dans un contexte formel.

2 Références normatives

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

2.1 Tests de conformité

NOTE – Dans le texte de la présente Recommandation, il sera fait référence à l'ensemble des normes ci-dessous par le terme CTMF.

– Recommandation UIT-T X.290 (1995) (équivalant à l'ISO/CEI 9646-1:1994), *Cadre général et méthodologie des tests de conformité OSI pour les Recommandations sur les protocoles pour les applications de l'UIT-T – Spécifications des suites de tests abstraites*.

– Recommandation UIT-T X.291 (1995) (équivalant à l'ISO/CEI 9646-2:1994), *Cadre général et méthodologie des tests de conformité d'interconnexion des systèmes ouverts pour les Recommandations sur les protocoles pour les applications de l'UIT-T – Spécification de suite de tests abstraite*.

– Recommandation X.292 du CCITT (1992) (équivalant à l'ISO/CEI 9646-3:1992), *Cadre général et méthodologie des tests de conformité OSI pour les Recommandations sur les protocoles pour les applications du CCITT – Notation combinée arborescente et tabulaire*.

– Recommandation UIT-T X.293 (1995) (équivalant à l'ISO/CEI 9646-4:1994), *Cadre général et méthodologie des tests de conformité d'interconnexion des systèmes ouverts pour les Recommandations sur les protocoles pour les applications de l'UIT-T – Réalisation des Tests*.

– Recommandation UIT-T X.294 (1995) (équivalant à l'ISO/CEI 9646-5:1994), *Cadre général et méthodologie des tests de conformité OSI pour les Recommandations sur les protocoles pour les applications de l'UIT-T – Prescriptions des laboratoires de test et des clients en matière de processus d'évaluation de conformité*.

– Recommandation UIT-T X.295 (1995) (équivalant à l'ISO/CEI 9646-6:1994), *Cadre général et méthodologie des tests de conformité d'interconnexion des systèmes ouverts pour les Recommandations sur les protocoles pour les applications de l'UIT-T – Spécification des tests de profil de protocole*.

– Recommandation UIT-T X.296 (1995) (équivalant à l'ISO/CEI 9646-7:1995), *Cadre général et méthodologie des tests de conformité OSI pour les Recommandations sur les protocoles pour les applications de l'UIT-T – Déclarations de conformité d'instance*.

2.2 Techniques de description formelle

– Recommandation Z.100 du CCITT (1993), *Langage de description et de spécification du CCITT*.

– Recommandation UIT-T Z.120 (1996), *Diagramme de séquence des messages*.

– ISO/CEI 8807:1989, *Systèmes de traitement de l'information – Interconnexion de systèmes ouverts – LOTOS – Technique de description formelle basée sur l'organisation temporelle de comportement observationnel* .

– ISO/CEI 9074:1989, *Systèmes de traitement de l'information – Interconnexion de systèmes ouverts – Estelle – Technique de description formelle basée sur un modèle de transition d'état étendu* .

3 Définitions

3.1 Termes d'autres normes apparentées

NOTE – Bien que les définitions suivantes soient données dans l'UIT-T X.290 et l'ISO/CEI 9646-1, elles sont reprises ici car leur signification est importante pour les interprétations formelles de la présente Recommandation.

3.1.1 méthode de test abstraite: définition de la façon dont une instance sous test doit être testée, formulée à un niveau d'abstraction approprié pour rendre cette description indépendante de toute réalisation particulière d'un moyen de test, mais suffisamment détaillée pour permettre la spécification des tests au moyen de cette méthode.

3.1.2 suite de tests de conformité: ensemble complet de tests élémentaires, pouvant être combinés en groupes de tests imbriqués, qui est choisi pour l'exécution de tests de conformité dynamique d'un ou plusieurs protocoles.

3.1.3 test de conformité: test visant à déterminer jusqu'à quel point une instance sous test est conforme.

3.1.4 implémentation conforme: instance sous test qui satisfait les conditions de conformité statique et dynamique, en accord avec les capacités déclarées dans la ou les déclarations ICS.

3.1.5 condition de conformité dynamique: toutes les conditions (et options) qui déterminent le comportement observable autorisé par la ou les spécifications applicables dans les instances de communication.

3.1.6 verdict d'échec: verdict de test rendu lorsque les résultats de test observés attestent la non-conformité à au moins une des conditions de conformité qui faisaient l'objet du test élémentaire, ou contiennent au moins un événement de test non valide par rapport aux spécifications concernées.

3.1.7 déclaration de conformité d'une implémentation (ICS, ***implementation conformance statement*)**: déclaration faite par le fournisseur d'une implémentation ou d'un système qui déclare se conformer à une spécification donnée, qui précise les capacités mises en œuvre. La déclaration ICS peut prendre différentes formes: ICS de protocole, ICS de profil, ICS de protocole spécifique et ICS d'objet informationnel.

3.1.8 instance sous test (IUT, ***implementation under test*)**: ensemble des procédures d'un ou de plusieurs protocoles mises en œuvre dans une relation utilisateur/fournisseur adjacente qui constitue la partie du système ouvert réel qui doit être soumise à des tests.

3.1.9 informations supplémentaires sur l'implémentation concernant le test (IXIT, ***implementation extra information for testing*)**: déclaration faite par le fournisseur ou le réalisateur d'une instance sous test, qui contient toutes les informations (en plus de celles fournies dans la déclaration ICS) concernant l'instance sous test et son environnement de test, ou qui y renvoie, et qui doit permettre au laboratoire de test d'exécuter une suite de tests appropriée sur cette instance. Une déclaration IXIT peut prendre différentes formes: IXIT de protocole, IXIT de profil, IXIT de profil spécifique et IXIT d'objet informationnel, déclaration d'instance de protocole TMP.

3.1.10 moyens de test (MOT, ***means of testing*)**: ensemble des équipements et procédures qui peuvent s'acquitter de la dérivation, de la sélection, de la paramétrisation et de l'exécution de tests élémentaires, conformément à une suite ATS normalisée de référence; cet ensemble peut produire un journal de conformité.

3.1.11 **suite de tests abstraite paramétrée**: suite de tests abstraite sélectionnée dont tous les paramètres pertinents ont été fixés à des valeurs conformes aux déclarations ICS et IXIT appropriées.

3.1.12 **succès (verdict de)**: verdict de test rendu lorsque les résultats de test observés attestent la conformité à la ou aux conditions de conformité sur lesquelles était axé l'objet du test élémentaire et lorsque aucun événement de test non valide n'a été détecté.

3.1.13 **point de contrôle et d'observation (PCO**, ***point of control and observation*****)**: point dans un environnement de test, où il faut contrôler et observer l'apparition d'événements de test, comme cela est défini dans la méthode de test abstraite.

3.1.14 **suite de tests abstraite de référence normalisée**: suite ATS normalisée pour laquelle sont réalisés les moyens de tests.

3.1.15 **spécifications de conformité statique**: une des conditions qui spécifie les limites imposées à la combinaison de capacités mises en œuvre autorisée dans un système ouvert réel déclaré conforme aux spécifications concernées.

3.1.16 **objectif d'un test**: description simple d'un objectif de test strictement défini, centré sur une seule condition de conformité ou un ensemble de conditions de conformité connexes, telle qu'elle est spécifiée dans la spécification appropriée.

3.1.17 **verdict d'un test**: jugement "succès", "échec" ou "non concluant" spécifié dans le test élémentaire abstrait, sur la conformité d'une instance sous test lorsqu'un test élémentaire a été exécuté.

3.2 Termes définis dans la présente Recommandation

3.2.1 **compatibilité**: voir 6.4.3

3.2.2 **(suite de tests) complète**: voir 8.2

3.2.3 **conformité**: voir 6.4

3.2.4 **test de conformité**: voir 8.2

3.2.5 **conformité dynamique**: voir 6.4.2 et 6.4.3

3.2.6 **(suite de tests) exhaustive**: voir 8.2

3.2.7 **modèle de défaut**: voir 8.4.1

3.2.8 **couverture des défauts**: voir 8.5

3.2.9 **spécification formelle**: voir 6.2

3.2.10 **implémentation**: voir 6.3

3.2.11 **point d'accès d'une implémentation**: voir 7.2

3.2.12 **(modèle formel de) point d'accès d'une implémentation**: voir 7.3

3.2.13 **option d'implémentation**: voir 6.2

3.2.14 **relation d'implémentation**: voir 6.4.2

3.2.15 **instance sous test**: voir 3.1

3.2.16 **(modèle formel de) instance sous test**: voir 7.3

3.2.17 **spécification instanciée**: voir 6.2

3.2.18 **point d'interaction**: voir 7.3

3.2.19 **spécification plus souple**: voir 8.4.2

3.2.20 **modèle d'implémentation**: voir 6.3

3.2.21 **mutant**: voir 8.4.1

3.2.22 **observation**: voir 7.4.1

3.2.23 **spécification paramétrée**: voir 6.2

3.2.24 **point de contrôle et d'observation**: voir 3.1

3.2.25 **(modèle formel de) point de contrôle et d'observation**: voir 7.3

3.2.26 **relation de satisfaction**: voir 6.4.3

3.2.27 **(suite de tests) saine**: voir 8.2

3.2.28 **spécification**: voir 6.2

3.2.29 **conformité statique**: voir 6.4.1

3.2.30 **architecture de test**: voir 7.2

3.2.31 **(modèle formel de) test élémentaire**: voir 7.3

3.2.32 **exécution de test élémentaire**: voir 7.4.1

3.2.33 **contexte de test**: voir 7.2

3.2.34 **(modèle formel de) contexte de test**: voir 7.3

3.2.35 **génération de test**: voir 8.3

3.2.36 **objectif d'un test**: voir 3.1 et 7.4.2

3.2.37 **(modèle formel de) objectif d'un test**: voir 7.4.2

3.2.38 **(coût de) suite de tests**: voir 8.6

3.2.39 **(modèle formel de) suite de tests**: voir 7.3

3.2.40 **(réduction de) suite de tests**: voir 8.4

3.2.41 **testeur**: voir 7.2

3.2.42 **(modèle formel de) testeur**: voir 7.3

3.2.43 **rapport d'implémentation plus faible**: voir 8.4.2

4 Abréviations

La présente Recommandation utilise les abréviations suivantes:

CTMF cadre général et méthodologie des tests de conformité (*conformance testing methodology and framework*)

FDT technique de description formelle (*formal description technique*)

FMCT méthodes formelles de tests de conformité (*formal methods in conformance testing*)

IAP point d'accès d'implémentation (*implementation access point*)

ICS déclaration de conformité d'implémentation (*implementation conformance statement*)

IUT instance sous test (*implementation under test*)

IXIT informations supplémentaires sur l'implémentation destinées au test (*implementation extra information for testing*)

LOTOS langage de spécification d'ordonnancement temporel (*language of temporal ordering specifications*)

LTS système de transition étiqueté (*labelled transition system*)

MSC diagramme de séquences de messages (*message sequence chart*)

PCO point de contrôle et d'observation (*point of control and observation*)

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

TTCN notation combinée arborescente et tabulaire (*tree and tabular combined notation*)

5 Concepts mathématiques et conventions de notation

5.1 Ensembles

Tout sous-ensemble d'un ensemble est représenté par des lettres majuscules (par exemple *A*, *B*, *C*) ou par une série de lettres majuscules (par exemple *SPECS*, *IMPLS*, *TESTS*). Les éléments d'un ensemble sont représentés par des lettres minuscules (par exemple *a*, *b*, *c*).

Les opérations suivantes sur les ensembles sont utilisées dans la présente Recommandation:

$\{a,b,c,...\}$	l'ensemble contenant les éléments *a*, *b*, *c*, ...; l'ordre d'apparition de ces éléments dans un ensemble n'a pas d'importance.
$\emptyset$	l'ensemble vide, c'est-à-dire l'ensemble sans élément.
$a \in A$	*a* est un élément de l'ensemble *A*.
$\{a \in A \mid P(a)\}$	l'ensemble contenant tous les éléments de *A* qui satisfont au prédicat *P*. Quelquefois, $\{a \mid P(a)\}$ est utilisé lorsque l'ensemble *A* peut être déduit du contexte.
$A \subseteq B$	*A* est un sous-ensemble de *B*, cela veut dire que tous les éléments de *A* sont également éléments de *B*.
$A = B$	l'ensemble *A* est égal à l'ensemble *B*, cela veut dire que *A* est un sous-ensemble de *B* et *B* est un sous-ensemble de *A*.
$A \subset B$	*A* est un sous-ensemble vrai de *B*, cela veut dire que *A* est un sous-ensemble de *B* et *A* n'est pas égal à *B*.
$A \cap B$	l'intersection de *A* et de *B*, c'est-à-dire l'ensemble contenant tous les éléments qui sont à la fois dans *A* et dans *B*.
$\bigcap_{i \in I} A_i$	intersection généralisée, c'est-à-dire l'intersection de tous les ensembles A_i: $A_1 \cap A_2 \ldots \cap A_n$, où *n* est un nombre naturel.
$A \cup B$	la réunion de *A* et de *B*, c'est-à-dire l'ensemble contenant tous les éléments qui sont soit dans *A*, soit dans *B* (ou dans les deux à la fois).
$\bigcup_{i \in I} A_i$	réunion généralisée, c'est-à-dire la réunion de tous les ensembles A_i: $A_1 \cup A_2 \ldots \cup A_n$, où *n* est un entier naturel.
$A \times B$	le produit Cartésien de *A* et de *B* représentant l'ensemble de tous les couples ordonnés (*a*, *b*) tel que $a \in A$ et $b \in B$.
$A_1 \times A_2 \times ... \times A_n$	le produit Cartésien généralisé de A_1, A_2, ..., A_n représentant l'ensemble de tous les couples ordonnés (a_1, a_2, …, a_n) tel que $a_1 \in A_1$, $a_2 \in A_2$, ..., $a_n \in A_n$.
$A - B$	l'ensemble différence de *A* et de *B*, c'est-à-dire l'ensemble contenant tous les éléments de *A* qui ne sont pas dans *B*.
Ensemble de puissance(A)	l'ensemble de puissance de *A*, c'est-à-dire l'ensemble contenant tous les sous-ensembles de *A*.
$R_{\geq 0}$	l'ensemble des nombres réels positifs, y compris zéro.

Le symbole '/' superposé à un opérateur (ensemble) est utilisé pour représenter la négation de l'opérateur, exemple $a \notin A$ (*a* n'appartient pas à l'ensemble *A*), $A \not\subset B$ (*A* n'est pas un sous-ensemble vrai de *B*), etc.

5.2 Logique

Les notations logiques suivantes sont utilisées dans la présente Recommandation:

$\neg p$	non *p*, c'est-à-dire la négation de *p*
$p \wedge q$	*p* et *q*, la conjonction de *p* et *q*
$p \vee q$	*p* ou *q*, la disjonction de *p* et *q*
$p \Rightarrow q$	*p* implique *q*, également lu: non *p* ou *q*

$p \Leftrightarrow q$	p est équivalent à q, c'est-à-dire $(p \Rightarrow q) \wedge (q \Rightarrow p)$
$\forall\, a \in A$	pour tous les éléments a de l'ensemble A
$\exists\, a \in A$	il existe un élément a dans l'ensemble A

5.3 Relations

Les relations sont représentées par une abréviation en lettres minuscules soulignées (par exemple rel).

Soient A et B des ensembles; ainsi une relation binaire rel entre A et B est un sous-ensemble de leur produit Cartésien: rel $\subseteq A \times B$. L'élément $a \in A$ est en relation avec $b \in B$ si $(a,b) \in$ rel (ou réciproquement a rel b). De la même manière, une relation n-aire est un sous-ensemble de $A_1 \times A_2 \times ... \times A_n$.

Le *domaine* dom(rel) de la relation rel $\subseteq A \times B$ est défini comme étant l'ensemble contenant tous les éléments dans A qui sont en relation à un élément quelconque $b \in B$, c'est-à-dire:

$$\text{dom(rel)} = \{a \in A \mid \exists\, b \in B : (a,b) \in \text{rel}\}$$

Une relation (binaire) rel sur A est un sous-ensemble de $A \times A$.

5.4 Fonctions

Les noms de fonction sont représentés par une abréviation en lettres minuscules soulignées (par exemple fonc).

Une *fonction partielle* func est une relation entre deux ensembles A et B dont la propriété est telle que pour chaque $a \in A$ il existe au moins un élément $b \in B$ tel que $\langle a,b \rangle \in$ func, c'est-à-dire:

$$\forall a \in A : \forall b_1,b_2 \in B : (\langle a,b_1 \rangle \in \text{func} \wedge \langle a,b_2 \rangle \in \text{func}) \Rightarrow b_1 = b_2$$

Lorsqu'une fonction est introduite, sa signature est représentée ainsi:

$$\text{func} : A \rightarrow B$$

Une *fonction* (*totale*) func : $A \rightarrow B$ est une fonction partielle qui satisfait à dom(func) = A.

6 La signification de la conformité

6.1 Introduction

La conformité implique la définition d'une implémentation, à savoir s'il s'agit d'une implémentation valide d'une spécification donnée par rapport à une certaine notion d'exactitude. Afin de formaliser le concept de conformité, les implémentations seront modélisées par des objets formels appelés modèles. La conformité peut être caractérisée par des relations entre modèles d'implémentations et spécifications ou par la satisfaction à des conditions spécifiées par des modèles d'implémentations ou par les deux à la fois. Le présent paragraphe définit la conformité qui comprend des définitions de spécification, d'implémentations, de modèles d'implémentations, de relations d'implémentation et de conditions de conformité.

6.2 Spécifications

Une *spécification (formelle)* prescrit le comportement d'un système en utilisant une FDT. Si la FDT permet l'utilisation des paramètres, une spécification ayant des paramètres formels est appelée *spécification paramétrée*. Si les paramètres formels sont instanciés avec des valeurs réelles ou s'il n'y a pas de paramètres formels, la spécification est une spécification instanciée.

L'ensemble des spécifications instanciées est représenté par *SPECS*. Une spécification paramétrée s est considérée comme une fonction de son paramètre d'espace D_s (l'ensemble de toutes les valeurs réelles possibles pour les paramètres formels) vers *SPECS*:

$$s : D_s \rightarrow SPECS$$

Les spécifications contiennent souvent des descriptions de caractéristiques dont la prise en charge est optionnelle dans un produit mettant en œuvre la norme. La liberté qui comprend la prise en charge ou non de certaines caractéristiques est appelée options d'implémentation. Les spécifications contiennent, en général, de telles options. Les options d'implémentation sont représentées par les paramètres formels d'une spécification, ce qui signifie que la spécification est paramétrée sur ses options d'implémentation.

Une spécification qui contient des *options d'implémentation* définit un ensemble de spécifications instanciées; une spécification instanciée pour chaque choix parmi les options. Une spécification doit clairement indiquer quelles sont les associations de prise en charge d'options qui sont autorisées.

Il est nécessaire d'inclure dans le formulaire de déclaration ICS de protocole/profil, des informations sur les options d'implémentation. La déclaration ICS décrit un choix spécifique d'options d'implémentation dans le formulaire ICS. Une spécification paramétrée sur son formulaire de déclaration ICS est instanciée par une déclaration ICS. La déclaration ICS doit contenir suffisamment d'informations pour fournir les valeurs réelles d'options d'implémentation.

Le terme *spécification* est utilisé dans la présente Recommandation pour représenter une *spécification instanciée*.

NOTE – Le CTMF exige que la déclaration ICS de protocole/profil soit publiée en annexe à la norme de protocole/profil.

Exemple

Supposons une spécification qui permet de mettre en œuvre différents niveaux de prise en charge des caractéristiques. Les niveaux sont numérotés de 0 à 3. La spécification paramétrée et normalisée peut alors être représentée par: *s*(*niveau:* 0...3). La déclaration ICS de protocole peut contenir les informations demandant la mise en œuvre de *niveau* = 2. Ceci signifie que la spécification instanciée de l'implémentation sous test est, en fait, *s*(2).

6.3 Implémentations

Une implémentation est constituée d'une association de matériels et de logiciels. Les implémentations disposent de connecteurs physiques ou d'interfaces de programmation pour communiquer avec leur environnement ou les utilisateurs finaux. L'ensemble des implémentations est représenté par *IMPS*.

Une spécification est un objet formel tandis qu'une implémentation est un objet physique. Afin de formaliser le concept de conformité, ces différents types d'objets doivent être mis en relation. Des implémentations ne peuvent pas être soumises à un raisonnement formel puisqu'elles ne sont pas des objets formels. Par conséquent, il n'est pas possible de définir directement une relation formelle entre implémentations et spécifications.

On suppose, dans la suite de la présente Recommandation, que toute instance IUT $\in$ *IMPS* peut être modélisée par un élément m_{IUT} dans un formalisme *MODS* (par exemple, systèmes de transition étiquetés, machines d'états finis). Cette hypothèse est appelée *hypothèse de test*. L'activité de test consiste à extraire des informations de l'instance IUT en la testant, de sorte que, à partir de ces informations, le modèle m_{IUT} puisse être construit de manière suffisamment détaillée pour décider de sa conformité.

NOTE 1 – On ne prend comme hypothèse que la possibilité de construire un modèle et non le fait que le modèle est *a priori* connu.

NOTE 2 – Le formalisme *MODS* utilisé pour modéliser le comportement d'une implémentation peut être le même que le formalisme *SPECS* qui est utilisé pour la spécification.

Une implémentation peut avoir plusieurs modèles pour un choix spécifique de *MODS*. L'hypothèse de test implique qu'il n'est pas possible de faire la distinction entre ces modèles par des tests. Par conséquent, il est suffisant de modéliser une instance IUT par un seul élément m_{IUT} de l'ensemble de modèles possibles *MODS*.

NOTE 3 – On suppose que l'IUT peut être modélisée avec suffisamment de précision pour qu'elle puisse représenter l'IUT en termes de propriétés prescrites par la spécification.

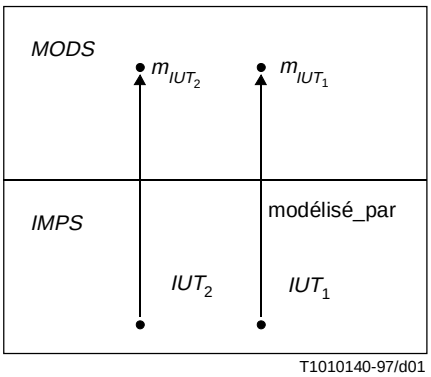

Figure 1/Z.500 – La relation entre éléments de *IMPS* et de *MODS*

6.4 Conformité d'une implémentation à une spécification formelle

La conformité entre une implémentation et une spécification existe lorsque l'implémentation est correcte par rapport à la spécification. La conformité est définie en deux parties:

- conformité statique,
- conformité dynamique.

Une implémentation est conforme à une spécification si, et uniquement si, sa conformité statique et sa conformité dynamique sont établies.

6.4.1 Conformité statique

La *conformité statique* implique l'instanciation correcte d'une spécification paramétrée. Une instance sous test, IUT, munie de la déclaration de conformité d'implémentation correspondante ICS_{IUT} est statiquement conforme à une spécification paramétrée $s : D_s \rightarrow SPECS$ si ICS_{IUT} est contenue dans le domaine de s, c'est-à-dire $ICS_{IUT} \in D_s$; ce qui signifie que $s(ICS_{IUT})$ est définie. Vérifier la conformité statique revient à effectuer une vérification de type du paramètre réel ICS_{IUT} par rapport au 'type' D_s.

La conformité statique signifie que la déclaration ICS_{IUT} est acceptée par la spécification paramétrée et, par conséquent, que la combinaison spécifique d'options d'implémentation décrite dans ICS_{IUT} est autorisée.

Les combinaisons admises et les gammes d'options d'implémentation contenues dans la déclaration ICS_{IUT}, c'est-à-dire la spécification de l'ensemble D_s, peuvent être décrites par des *conditions de conformité statique*. Une condition de conformité statique est une condition qui spécifie les limites imposées aux domaines et aux combinaisons d'options et de capacités mises en œuvre, admises dans une implémentation donnée par la spécification (voir CTMF, Partie 1, 3.4.4).

NOTE – Si des conditions de conformité statique vont au-delà du contrôle normal de type des valeurs réelles de paramètre dans la FDT utilisée, il est admis d'exprimer ces conditions dans la partie comportant le comportement de la description formelle (voir Annexe A).

6.4.2 Conformité dynamique

La *conformité dynamique* implique le comportement observable admis d'une implémentation dans des instances de communication telles que décrites par la spécification. La conformité dynamique entre une implémentation et une spécification est formellement caractérisée par une relation entre le modèle de l'implémentation et la spécification. Cette relation est appelée *relation d'implémentation*. Elle sera représentée par $\underline{\text{imp}}$, où $\underline{\text{imp}}$ a la signature suivante:

$$\underline{\text{imp}} \subseteq MODS \times SPECS$$

Une instance IUT est dynamiquement conforme à une spécification instanciée s par rapport à la relation $\underline{\text{imp}}$, si $m_{IUT}\ \underline{\text{imp}}\ s$. Dans ce cas, m_{IUT} est un modèle conforme de s par rapport à $\underline{\text{imp}}$. Etant donné qu'une telle relation d'implémentation $\underline{\text{imp}}$ exprime l'exactitude entre la spécification s et l'instance IUT.

Une spécification instanciée peut avoir plusieurs implémentations conformes. Pour une spécification $s \in SPECS$ et une relation d'implémentation $\underline{\text{imp}}$, l'ensemble M_s représente l'ensemble de tous les modèles conformes dans *MODS* et donnés par:

$$M_s = \{m \in MODS \mid m\ \underline{\text{imp}}\ s\}$$

NOTE – Pour définir la conformité, l'existence des objets suivants est une condition préalable: une spécification $s \in SPECS$, une instance IUT $\in IMPS$, la documentation du choix entre options ICS_{IUT} et une relation d'implémentation. La relation d'implémentation n'est pas universelle; pour différents domaines d'application, il est admis d'utiliser différentes relations d'implémentation. L'exemple ci-dessous fournit des types de relation d'implémentation pour Estelle, LOTOS et SDL; ceux-ci sont décrits de manière plus détaillée en Annexe A.

La Figure 2 illustre la manière dont une spécification instanciée $s \in SPECS$ détermine un ensemble d'implémentations conformes I_s. L'ensemble I_s représente l'ensemble d'implémentations qui peut être modélisé par des modèles dans M_s. Par conséquent, l'ensemble I_s est l'ensemble d'implémentations qui met en œuvre la spécification s de manière correcte.

En supposant que s soit une spécification de protocole, des éléments de I_s pourront communiquer en utilisant ce protocole. Si s est la spécification d'une pile complète de protocoles (un profil) et une application répartie, l'interopérabilité des éléments de I_s pourra s'effectuer.

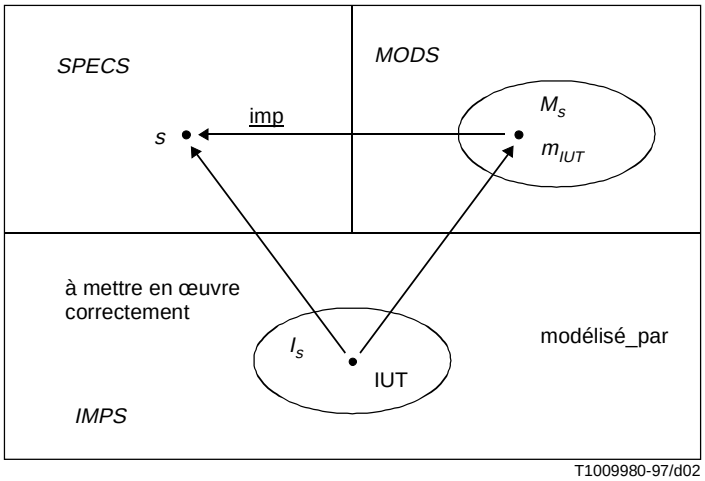

Figure 2/Z.500 – Relations entre *IMPS*, *MODS* et *SPECS*

Exemple

L'*équivalence de trace* et le *préclassement de trace* sont des exemples de relations d'implémentation souvent utilisés si, pour *MODS* et *SPECS*, on choisit Estelle ou SDL. Si une équivalence de trace est requise, l'ensemble des traces d'exécution de l'implémentation doit être égal à l'ensemble des traces de la spécification. Dans le cas du préclassement de trace, le premier ensemble doit être inclus dans le second. (Ce qui signifie qu'une partie du comportement spécifié doit être mise en œuvre.)

L'*équivalence de panne* et le *préclassement de panne* sont des exemples de relations d'implémentation souvent utilisés pour LOTOS. Si le préclassement de panne est exigé, l'ensemble des traces de l'implémentation doit être inclus dans l'ensemble des traces de la spécification et l'implémentation ne doit pas donner lieu à des blocages qui ne sont pas spécifiés.

6.4.3 Conditions de conformité dynamique

Dans le paragraphe 6, la conformité a été définie de manière abstraite au moyen d'une spécification instanciée ainsi qu'avec une relation d'implémentation entre les ensembles *MODS* et *SPECS*. D'autre part, dans le CTMF, la définition de la conformité dynamique est fondée sur le concept d'un recueil de conditions de conformité dynamique. Ces deux approches peuvent être utilisées pour spécifier un comportement observable conforme. Elles peuvent être utilisées séparément ou en combinaison. La conformité définie au moyen d'une spécification instanciée avec une relation d'instance est présentée au 6.4.2. Le présent sous-paragraphe montre que l'utilisation des conditions de conformité dynamique peut constituer une extension possible de cette définition et que les deux approches définissent le même concept avec la même puissance d'expression.

Une condition de conformité dynamique est une condition qui spécifie les comportements observables autorisés dans des instances de comportement (voir dans CTMF, Partie 1, 3.4.3). Il s'agit d'une propriété à laquelle il convient de satisfaire par (le modèle de) l'implémentation pour que celle-ci soit conforme.

Les conditions de conformité dynamique sont exprimées dans un langage de prescriptions *REQS* qui représente l'ensemble des conditions qui peuvent être exprimées dans ce langage particulier. Dans l'approche fondée sur les conditions, une spécification instanciée $s \in SPECS$ est exprimée comme un ensemble de conditions $R_s \subseteq REQS$, c'est-à-dire *SPECS* = *Ensemble de puissance*(*REQS*). Un élément de $r \in R_s$ représente une seule condition de conformité dynamique. En général, l'ensemble R_s peut être infini.

La conformité dynamique entre une implémentation et une spécification selon la méthode par conditions est formellement caractérisée par une relation entre le modèle de l'implémentation et la spécification. Cette relation est appelée *relation de satisfaction*. Elle sera représentée par $\underline{sat}$ où $\underline{sat}$ a la signature suivante:

$$\underline{sat} \subseteq MODS \times REQS$$

Une instance IUT est dynamiquement conforme à la spécification $R_S \subseteq REQS$ si le modèle m_{IUT} de IUT satisfait à toutes les conditions de conformité dans R_S. L'ensemble M_{R_S} de modèles d'implémentations conformes dans la méthode par conditions est donné par:

$$M_{R_S} = \{m \in MODS \mid \forall r \in R_S : m \underline{\text{sat}}\ r\}$$

Pour une condition de conformité particulière $r_i \in R_S$, l'ensemble M_{r_i} représente l'ensemble de tous les modèles dans *MODS* qui satisfont à la condition r_i, c'est-à-dire $M_{r_i} = \{m \in MODS \mid m \underline{\text{sat}}\ r_i\}$. La Figure 3 montre comment l'intersection des ensembles M_{r_i} détermine l'ensemble M_{R_S} d'implémentations conformes.

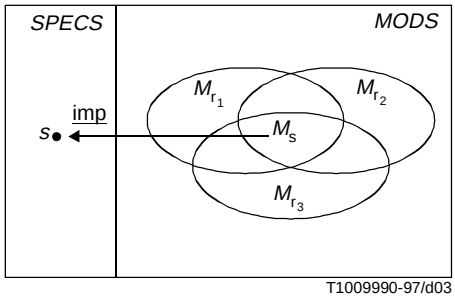

Figure 3/Z.500 – Conditions de conformité et implémentations conformes

6.4.4 Combinaison de spécifications

On peut combiner différentes spécifications formelles, qui ne sont pas nécessairement exprimées dans le même langage formel, pour définir un seul ensemble de modèles d'implémentations conformes. Si les spécifications s_1, s_2, s_3, .. s_n ayant des relations d'implémentation $\underline{\text{imp}}_1$, $\underline{\text{imp}}_2$, $\underline{\text{imp}}_3$, ... $\underline{\text{imp}}_n$, définies sur la même classe de modèles d'implémentations *MODS*, définissent l'ensemble d'implémentations conformes M_{s_1}, M_{s_2}, .. M_{s_n}, respectivement, alors l'ensemble d'implémentations conformes définies par s_1, s_2, s_3, .. s_n est $M_{s_1} \cap M_{s_2} \cap .. \cap M_{s_n}$.

Les spécifications s_1, s_2, s_3, .. s_n *sont cohérentes si l'intersection qu'elles constituent n'est pas vide.* La cohérence implique que le recueil de spécifications soit applicable. C'est une exigence pour la combinaison de spécifications.

Un cas particulier de combinaison de spécifications est celui qui concerne une spécification *s* de comportement instanciée (avec une relation d'implémentation $\underline{\text{imp}}$) accompagnée d'une spécification par condition *R* (avec une relation de satisfaction $\underline{\text{sat}}$). La condition dans *R* peut spécifier des conditions supplémentaires d'implémentations conformes ou elle peut servir de spécification de remplacement pour exactement le même ensemble d'implémentations conformes. Dans le dernier cas, *s* (avec $\underline{\text{imp}}$) et *R* (avec $\underline{\text{sat}}$) sont dits *compatibles*:

$$\forall\ m \in MODS : m\ \underline{\text{imp}}\ s \Leftrightarrow \forall\ r \in R : m\ \underline{\text{sat}}\ r$$

7 Concepts de test

7.1 Introduction

Les tests constituent un moyen d'extraire des connaissances sur un système donné en l'expérimentant. L'expérimentation est réalisée par l'exécution de tests élémentaires. L'exécution d'un test élémentaire donne lieu à une observation permettant de conclure qu'un système donné dispose ou ne dispose pas d'une certaine propriété. Le présent paragraphe définit les concepts de base utilisés pour décrire l'exécution de tests élémentaires – il n'est fait aucune référence à une notion de conformité ou à une propriété soumise à des tests. Ces concepts de base incluent l'environnement dans lequel les tests élémentaires sont exécutés, la modélisation formelle de cet environnement, des tests élémentaires et des suites de tests, l'exécution de tests élémentaires, les observations qui peuvent être effectuées pendant l'exécution du test, l'interprétation des observations ainsi que l'objectif d'un test lié à un test élémentaire et à une suite de tests.

7.2 Architecture de test

L'*architecture de test* est une description de l'environnement dans lequel l'IUT est soumise au test. Elle décrit les aspects pertinents de la manière dont l'IUT est encastrée dans d'autres systèmes pendant la procédure de test ainsi que la manière dont l'IUT communique, par le biais de ces systèmes d'encastrement, avec le testeur. La Figure 4 donne une vue abstraite de l'architecture de test.

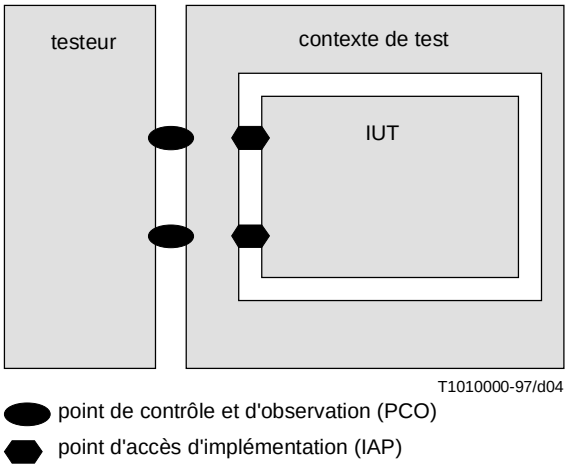

Figure 4/Z.500 – Architecture de test

Une *architecture de test* est constituée:

– d'un testeur;

– d'une instance sous test (IUT, *implementation under test*) (voir 3.1);

– d'un contexte de test;

– des points de contrôle et d'observation (PCO, *points of control and observation*) (voir 3.1);

– des points d'accès d'implémentation (IAP, *implementation access point*).

Le *testeur* est l'implémentation d'une suite de tests (voir 7.3). Il réalise des expériences en exécutant les tests élémentaires et en observant les résultats. Le testeur communique avec le contexte de test par l'intermédiaire des PCO et indirectement avec l'IUT par l'intermédiaire du contexte de test. Le testeur peut être sous-structuré en différents éléments (par exemple, en un testeur inférieur et un testeur supérieur, voir CTMF).

Le *contexte de test* est le système auquel l'IUT est encastrée et par l'intermédiaire duquel il communique avec le testeur. Il fait référence à des événements qui ont lieu au niveau des PCO au cours de la communication entre le contexte de test et le testeur, à des événements qui apparaissent au niveau des IAP au cours de la communication entre le contexte de test et l'IUT.

Un *point d'accès d'implémentation* (IAP) est un point d'interaction dans l'architecture de test où l'IUT interagit avec son environnement, c'est-à-dire le contexte de test et par l'intermédiaire du contexte de test (indirectement) avec le testeur.

NOTE – Dans certains cas, il est admis que les IAP et les PCO coïncident.

7.3 Modèle formel de l'architecture de test

Pour pouvoir réfléchir sur la procédure de test dans un cadre formel, les entités d'architecture de test doivent être formellement modélisées.

suite de tests – La description formelle du testeur est fournie par une *suite de tests*. Une suite de tests spécifie l'ensemble complet des expériences qui doivent être exécutées par le testeur. Le testeur est l'implémentation de la suite de tests.

Les expériences que constitue une suite de tests sont appelées des *tests élémentaires*. Chaque test élémentaire précise le comportement du testeur dans une expérimentation séparée qui teste un aspect de l'IUT et qui donne lieu à une observation et à un verdict (voir 7.4.1).

Le langage formel utilisé pour exprimer un test élémentaire est appelé *notation de test*. La notation de test est représentée par *TESTS*. En conséquence, une suite de tests étant un ensemble de tests élémentaires, est un élément de *Ensemble de puissance*(*TESTS*).

On suppose que les spécifications de tests élémentaires et de suite de tests sont correctement mises en œuvre dans le testeur conformément à la sémantique de la notation de test. Cette hypothèse est raisonnable étant donné que ces spécifications sont bien moins complexes que les spécifications du système.

instance sous test – Si l'on s'en tient à l'hypothèse de test (voir 6.3), l'IUT est formellement exprimée par son modèle $m_{IUT} \in MODS$.

contexte de test – Le modèle formel d'un contexte de test est une transformation du comportement tel qu'il est observé au niveau des IAP en un comportement tel qu'il est observé, après modélisation au niveau des PCO. Il est décrit comme une fonction *C* sur *MODS*:

$$C : MODS \rightarrow MODS$$

Il s'ensuit que le comportement de l'IUT tel qu'observé au niveau des PCO est formellement exprimé par $C(m_{IUT})$.

point de contrôle et d'observation et point d'accès d'implémentation – Les PCO et les IAP sont formellement modélisés comme des points d'interaction dans *MODS*. Les propriétés de ces points d'interaction dépendent de la manière dont les interactions entre les entités sont définies dans le formalisme *MODS* qui a été choisi. Il faut s'assurer que la modélisation des interactions dans *MODS* modélise fidèlement les interactions au fur et à mesure qu'elles apparaissent au niveau des PCO et des IAP dans l'architecture de test.

7.4 Exécution des tests

L'exécution d'une suite de tests $T \subseteq TESTS$ consiste à faire tourner le testeur qui met en œuvre *T* en association à l'IUT et le contexte de test pour chaque test élémentaire $t \in T$. Le lancement d'un test élémentaire est appelé *exécution de test élémentaire*. L'objectif de l'exécution de test élémentaire est d'expérimenter l'IUT afin de s'assurer de sa réaction correcte à un certain comportement.

7.4.1 Exécution de test élémentaire

L'exécution de test élémentaire consiste à lancer l'implémentation d'un test élémentaire $t \in TESTS$ en combinaison avec l'IUT et le contexte de test. Pendant le déroulement du test, il effectue une *observation*. Ceci peut comprendre une consignation des interactions qui ont lieu, un verdict (préliminaire) ou toute autre opération qui est considérée importante pour la détermination du résultat de l'exécution du test élémentaire. Lorsque l'exécution de test élémentaire donne lieu à une observation – $\sigma \in OBS$ – le résultat est défini par l'attribution d'un verdict $\underline{verd}_t$ qui peut varier en fonction du test élémentaire $t \in T$:

$$\underline{verd}_t : OBS \rightarrow \{\text{succès, non concluant, échec}\}$$

Une instance sous test IUT *passe* un test élémentaire *t* (correctement) mis en œuvre (dans un contexte donné de test particulier), si et uniquement si l'exécution de test de l'IUT avec *t* donne lieu à une observation σ pour laquelle le verdict "succès" est rendu:

$$\text{IUT } \underline{\text{réussit}}\ t \Leftrightarrow \underline{verd}_t\ (\sigma) = \text{succès}$$

Exemple

Le tableau suivant donne des exemples d'observations. Chaque ligne du tableau contient des résultats de mesure. La séquence de mesures comprend l'observation de l'exécution d'un test.

La mesure d'un événement donné est représentée comme étant le type de l'événement observé à (@) l'emplacement de l'interaction observée.

Temps	Evénement	Paramètres
0003	connexion @ PCO-1	émetteur = "1223" adresse = "4545" niveau = 3
0005	acceptation @ PCO-1	émetteur = "4545" adresse = "1223" niveau = 2
0012	déconnexion @ PCO-1	émetteur = "1223" adresse = "4545" raison = 5

7.4.2 Modèle formel d'exécution de test élémentaire

Pour l'interprétation du résultat de l'exécution de test élémentaire, il faut définir par une fonction un modèle d'exécution de test élémentaire. La fonction exec calcule les observations pour des modèles d'IUT (m_{IUT}) contenus dans un modèle de contexte de test (C):

$$\underline{\text{exec}} : TESTS \times MODS \rightarrow OBS$$

L'expression $\underline{\text{exec}}(t,C(m_{IUT}))$ modélise l'observation qui est effectuée par le test élémentaire t de l'IUT modélisée en tant que m_{IUT} dans le contexte de test C.

Si exec modélise correctement l'exécution de test, c'est-à-dire si l'exécution de test donne lieu à une observation σ si et seulement si $\underline{\text{exec}}(t,C(m_{IUT})) = \sigma$, on peut conclure de l'exécution de test d'une IUT avec un test élémentaire t que:

$$IUT \ \underline{\text{réussit}} \ t \ \Leftrightarrow \underline{\text{verd}}_t \ (\underline{\text{exec}}(t,C(m_{IUT}))) = \text{succès}$$

Le sous-ensemble de *MODS* pour lequel $\underline{\text{verd}}_t \ (\underline{\text{exec}}(t,C(m))) = \text{succès}$, est appelé l'*objectif de test formel* P_t :

$$P_t = \{m \in MODS \mid \underline{\text{verd}}_t \ (\underline{\text{exec}}(t,C(m))) = \text{succès}\}$$

Ainsi, l'objectif de test d'une IUT avec le test élémentaire t conclura si le modèle de l'IUT fait partie de son objectif de test formel P_t, c'est-à-dire si l'IUT réussit t si et uniquement si m_{IUT} est un élément de P_t.

Il est admis qu'une description textuelle accompagne l'objectif de test élémentaire formel. Cette description textuelle est appelée l'objectif de test (*informel* ou en *langage naturel*) (voir 3.1).

Un objectif de test formel en tant que sous-ensemble de *MODS* peut être spécifié au moyen des mêmes méthodes que celles utilisées pour spécifier les ensembles d'implémentations conformes, c'est-à-dire soit par une spécification de comportement instanciée munie d'une relation d'implémentation, soit par une condition ou un ensemble de conditions muni d'une relation de satisfaction. Il est également admis d'appeler une telle expression formelle objectif de test formel étant donné qu'elle représente un objectif de test formel.

Exemple

Un diagramme de séquences de messages (MSC, *message sequence chart*) décrit une séquence de comportement. Associés à la relation d'implémentation, le *préclassement de trace inverse* (toutes les séquences de comportement du MSC doivent être contenues dans l'implémentation) et le MSC spécifient un objectif de test formel, notamment l'ensemble des modèles d'implémentations qui contiennent le comportement spécifié par le MSC.

NOTE – L'objectif de test formel P_t est lié au test élémentaire t. L'ordre qui lie ces deux événements n'est pas prescrit; étant donné un test élémentaire t, son objectif de test formel P_t peut être calculé ou, étant donné un objectif de test formel p (par exemple, donné comme condition formelle) un test élémentaire t_p peut être développé tel que $P_{t_p} = p$. Il est admis que ce test élémentaire n'existe pas.

7.4.3 Exécution de suite de tests

La procédure d'exécution d'une suite de tests consiste à exécuter consécutivement chaque test élémentaire appartenant à la suite de tests. Comme indiqué au 7.4.1, un verdict est rendu pour chaque exécution de test élémentaire. Pour être exécutable, une suite de tests doit être finie. Cependant, lors de la modélisation de l'exécution d'une suite de tests avec exec, cette condition de finitude n'est pas nécessaire.

Une instance sous test IUT passe une suite de tests $T \subseteq TESTS$ si, et seulement si, elle réussit tous les tests élémentaires dans la suite de tests:

$$\text{IUT} \ \underline{\text{réussit}} \ T \Leftrightarrow \forall t \in T : \text{IUT} \ \underline{\text{réussit}} \ t$$

Si l'IUT passe une suite de tests, il s'ensuit que le modèle d'IUT est contenu dans tous les objectifs de test formel des tests élémentaires de la suite de tests:

$$IUT \ \underline{\text{réussit}} \ T \Leftrightarrow m_{IUT} \in \bigcap_{t \in T} P_t$$

L'ensemble $\bigcap_{t \in T} P_t$ est l'objectif de test formel de T. Il est représenté par P_T. Comme pour les objectifs de test formel des tests élémentaires, un objectif de test (langage informel ou naturel) peut accompagner l'objectif de test formel afin de fournir une description textuelle de l'objectif des essais avec T.

8 Test de conformité

8.1 Introduction

Le présent paragraphe utilise les concepts de tests définis dans le paragraphe 7 afin de tester la propriété de conformité définie au paragraphe 6. Ce paragraphe comprend les définitions des tests de conformité, de la génération de tests à partir de la spécification formelle, de la couverture des suites de tests et des méthodes permettant de limiter la taille des suites de tests.

8.2 Définition des tests de conformité

Le test de conformité est l'évaluation, au moyen de tests, de la conformité d'une implémentation à sa spécification. Le test de conformité est destiné à rassembler des informations permettant de construire un modèle m_{IUT} du comportement de l'implémentation IUT. Ce modèle est utilisé pour décider si m_{IUT} est un élément de l'ensemble de modèles d'implémentations conformes, c'est-à-dire si $m_{IUT} \in M_s$.

En général, il n'est pas possible d'obtenir des certitudes concernant la conformité entre implémentations et spécifications, ceci étant dû, par exemple, à l'absence de déterminisme dans l'implémentation, aux restrictions d'observabilité et de contrôlabilité de l'implémentation et aux limites pratiques résultant du fait que seul un nombre fini de tests peut être exécuté. Une suite de tests T peut avoir les propriétés suivantes en fonction de la relation entre P_T et M_s.

exhaustive: la suite de tests T est exhaustive si l'ensemble P_T de tous les modèles qui passent la suite de tests T est un sous-ensemble de l'ensemble des modèles conformes M_s: $P_T \subseteq M_s$. Ceci signifie que toutes les implémentations qui ont passé les tests sont conformes.

saine: la suite de tests T est saine si l'ensemble de modèles conformes M_s est un sous-ensemble de l'ensemble P_T de modèles qui passent l'instance IUT: $M_s \subseteq P_T$. Ceci signifie que toutes les implémentations qui ne passent pas ne sont pas conformes.

complète: la suite de tests T est complète si elle est à la fois saine et exhaustive, ce qui signifie que l'ensemble des modèles conformes est égal à l'ensemble des modèles qui passent l'implémentation: $P_T = M_s$.

Si la suite de tests T n'est ni saine, ni exhaustive, on ne peut alors tirer aucune conclusion quant à la conformité, au moyen de tests.

Dans l'idéal, une suite de tests est complète. En général, il n'est pas possible de construire une suite de tests exhaustive finie.

NOTE – Il est peu probable de rencontrer dans la pratique des suites de tests exhaustives. L'exhaustivité est un concept utile pour l'argumentation théorique sur un modèle (non fini) de suites de tests ainsi qu'un moyen de comparaison de suites de tests réalistes (voir 8.5: couverture).

8.3 Génération de tests

Dans la procédure de génération de tests, une suite de tests (c'est-à-dire un ensemble de tests élémentaires) est générée à partir d'une spécification formelle. L'utilisation d'une FDT pour la spécification est une exigence préalable de la génération automatique de tests.

La génération de tests est définie comme une fonction gen qui fournit une suite de tests pour une spécification instanciée, étant donné une relation d'implémentation imp et un contexte de test C:

$$\underline{\text{gen}}^{C,\underline{\text{imp}}} : \textit{SPECS} \rightarrow \textit{Ensemble de puissance}(\textit{TESTS})$$

La suite de tests générée est exprimée en une notation de test (voir 7.3: notation de test).

Il est exigé que la suite de tests générée soit saine (voir 8.2). Par conséquent, une suite de tests générée doit satisfaire la propriété suivante:

$$\forall m \in \textit{MODS}, m \ \underline{\text{imp}} \ s \Rightarrow m \in P_T$$

où

$$T = \underline{\text{gen}}^{C,\underline{\text{imp}}}(s)$$

La génération de test, telle que définie ci-dessus, opère sur des spécifications instanciées: $\underline{\text{gen}}^{C,\underline{\text{imp}}} : \textit{SPECS} \rightarrow \textit{Ensemble de puissance(TESTS)}$. Une suite de tests pour une spécification $s : D_s \rightarrow \textit{SPECS}$ et *ICS* est obtenue par $\underline{\text{gen}}^{C,\underline{\text{imp}}}(s(\textit{ICS}))$.

Il est possible de différer l'instanciation de la spécification paramétrée $s : D_s \rightarrow SPECS$ avec la déclaration *ICS*, jusqu'à ce que le test soit généré. Ceci signifie que les générations de tests opèrent sur des spécifications paramétrées, en générant une suite de tests paramétrée sur la déclaration *ICS*:

$$\underline{\text{pgen}}^{C,\underline{\text{imp}}} : (D \rightarrow SPECS) \rightarrow (D \rightarrow \textit{Ensemble de puissance}(TESTS))$$

où *D* est le domaine de toutes les déclarations d'implémentation possibles.

Une suite de tests pour une spécification $s : D_s \rightarrow SPECS$ et une déclaration de conformité d'implémentation *ICS* est obtenue par:

$$(\underline{\text{pgen}}^{C,\underline{\text{imp}}} : (s : D_s \rightarrow SPECS))(ICS)$$

Il est nécessaire que les suites de tests paramétrées ainsi générées soient saines pour chaque instanciation possible avec une déclaration *ICS*.

8.4 Réduction de la taille de suite de tests

Chaque suite de tests utilisée pour des tests de conformité doit être saine (voir 8.3), ce qui signifie que chaque exécution de test entraînant un verdict d'échec indique effectivement une erreur dans l'instance sous test (voir 8.2). L'exhaustivité des suites de tests n'est pas requise, ce qui signifie qu'on ne détectera pas nécessairement toutes les erreurs dans une instance sous test (voir 8.2). En général, le nombre de tests élémentaires qui serait nécessaire pour tester de manière exhaustive l'implémentation est très important ou même infini et, en conséquence, des tests exhaustifs ne sont pas faisables dans la pratique. Pour réduire la taille d'une suite de tests par rapport à la taille d'une suite de tests exhaustive idéale et afin que l'exécution des suites de tests soit, dans la pratique, faisable, différentes stratégies de réduction de la taille des suites de tests ont été identifiées.

8.4.1 Modèle de défaut

Les stratégies de réduction de la taille des suites de tests peuvent être présentées en termes d'un *modèle de défaut*. Un modèle de défaut *F* est un ensemble de modèles d'implémentations non conformes. Il est exprimé comme un sous-ensemble de $MODS - M_S$:

$$F \subseteq MODS - M_S$$

Un modèle de défaut peut être décrit comme une modification (un mutant) de la spécification $s \in SPECS$. Soit $\Delta_s \in SPECS$ contenant une modification par rapport à la spécification originale *s*; dans ce cas, le modèle de défaut décrit par Δ_s est $(M_{\Delta_s} - M_s)$.

Le modèle de défaut le plus grand est l'ensemble de toutes les implémentations non conformes $MODS - M_S$.

Exemple

Prenons comme hypothèse l'ensemble des spécifications *SPECS* et l'ensemble des modèles *MODS*, tous deux égaux à l'ensemble des Automates Finis à Entrées/Sorties. Pour une spécification particulière *s*, on peut reconnaître deux types de mutants:

a) des mutants avec des défauts de transfert, ce qui signifie que l'état final des transitions testées peut être différent de l'état final prescrit par la spécification *s*;

b) des mutants avec défaut de sortie, ce qui signifie que la sortie observée, après une entrée spécifique, peut ne pas être celle qui est prescrite par la spécification.

8.4.2 Stratégies des limitations de la taille des suites de tests

Les stratégies des limitations de la taille des suites de tests garantissent l'obtention d'une suite de tests saine, soit à partir d'une suite de tests *T* saine, mais probablement trop grande, ou soit à partir d'une spécification *s*, d'une relation d'implémentation $\underline{\text{imp}}$ et d'une fonction de génération de tests $\underline{\text{gen}}^{C,\underline{\text{imp}}}$ qui génère des suites de tests $\underline{\text{gen}}^{C,\underline{\text{imp}}}(s)$ saines mais probablement trop importantes. Les stratégies suivantes sont identifiées:

1) prenons tout sous-ensemble T' de la suite de tests *T*. Le modèle de défaut pour lequel le sous-ensemble de tests T' est le complément de son objectif de test formel: $F = MODS - P_{T'}$;

2) assouplir la spécification *s* en s' de façon que $M_s \subset M_{s'}$ (ce qui équivaut à mettre moins de conditions sur l'implémentation) et générer une suite de tests saine pour cette spécification plus souple: $\underline{\text{gen}}^{C,\underline{\text{imp}}}(s')$ est une suite de tests saine pour *s* par rapport à $\underline{\text{imp}}$.

Pour une spécification de condition $R \subseteq REQS$, il existe un cas spécial qui est la spécification plus souple $R' \subset R$ qui est constituée d'une sélection de conditions de conformité dynamique;

3) affaiblir la relation d'implémentation $\underline{\text{imp}}$ en $\underline{\text{imp}}'$ de sorte que $\{m \in MODS \mid m \ \underline{\text{imp}} \ s\} \subset \{m \in MODS \mid m \ \underline{\text{imp}}' \ s\}$ (c'est-à-dire permettre à plusieurs implémentations d'être correctes) et générer une suite de tests saine pour cette relation d'implémentation plus faible: $\underline{\text{gen}}^{C,\underline{\text{imp}}'}(s)$ est une suite saine pour s par rapport à $\underline{\text{imp}}'$;

4) proposer une hypothèse de test plus forte (voir 6.3), c'est-à-dire au lieu de supposer $m_{IUT} \in MODS$, supposer $m_{IUT} \in MODS'$ avec $MODS' \subset MODS$;

5) spécifier de manière explicite un modèle de défaut $F \subset MODS - M_s$, par exemple, en tenant compte des mutants Δ_s de s et générer une suite de tests saine qui détecte au minimum toutes les implémentations non conformes dans F, c'est-à-dire $P_T \cap F = \emptyset$.

Les différentes stratégies de réduction de taille des suites de tests contenues dans cette liste (qui n'est pas nécessairement exhaustive) peuvent être combinées pour réduire la taille d'une suite de tests.

Exemple

Si nous avons une spécification d'automate couvrant l'ensemble des entiers et par conséquent avec un espace d'état (théoriquement) infini, $MODS = SPECS$ et si la relation d'implémentation exige que l'implémentation dispose de toutes les séquences de comportement (*traces*) de la spécification, dans ce cas, la séquence suivante de stratégies permettant de limiter la taille de la suite de tests peut être appliquée séquentiellement:

- renforcer l'hypothèse de test selon laquelle le nombre d'états de toute implémentation est restreint à un nombre particulier n;
- tenir compte de la relation d'implémentation la plus faible: la relation d'implémentation doit disposer de toutes les traces de la spécification jusqu'à la longueur l;
- prendre en compte la spécification la plus souple s' tel que s' contienne un sous-ensemble fini des traces de s.

A présent, toute fonction de génération de tests qui génère des suites de tests saines pour la relation d'implémentation la plus faible, génère, lorsqu'elle est appliquée à s', une suite de tests saine pour la spécification originale par rapport à la relation d'implémentation originale.

NOTE – On notera qu'il y a une différence entre un mutant Δ_s et une spécification plus souple s' de s. Un mutant décrit les défauts prévus pour lesquels les tests sont générés (le modèle de défaut est un sous-ensemble de l'ensemble d'implémentations spécifié par Δ_s). Une spécification plus souple prescrit des implémentations qui *ne* sont *pas* testées (le modèle de défaut est un sous-ensemble du *complément* des implémentations prescrites par s').

8.5 Couverture de défaut

La *couverture de défaut* est une mesure normalisée de la portée de l'exhaustivité d'une suite de tests donnée par rapport à un modèle de défaut F. Ceci signifie qu'elle exprime une quantification de la qualité d'une suite de tests en termes de capacités de détection d'erreur. Une mesure de couverture peut être utilisée pour comparer des suites de tests; une couverture élevée exprime un niveau de qualité élevé.

La couverture de défaut est exprimée comme une fonction ayant la signature suivante:

$$\underline{\text{cov}}_F : \textit{Ensemble de puissance}(TESTS) \rightarrow [0,1]$$

conditionnée par le fait que la couverture doit augmenter si des implémentations plus erronées sont détectées dans F [P_T est l'objectif de test formel de la suite de tests T (voir 7.4.1); $F - P_T$ est le sous-ensemble de F de tous modèles d'implémentations qui échouent aux tests avec T]:

$$F - P_{T_1} \subseteq F - P_{T_2} \Rightarrow \underline{\text{cov}}_F(T_1) \leq \underline{\text{cov}}_F(T_2)$$

Si F est omis, on suppose qu'il s'agit de l'ensemble de toutes les implémentations erronées: $\underline{\text{cov}}(T) = \underline{\text{cov}}_{MODS - M_s}(T)$.

8.6 Coût de suite de tests

Les coûts connus pour générer, maintenir, mettre en œuvre et exécuter une suite de tests sont exprimés par le terme de coût de cette suite de tests. Ceci signifie que le coût exprime une quantification des efforts en termes d'argent, de temps, etc., nécessaires pour une suite de tests donnée. Une mesure du coût peut être utilisée pour comparer des suites de tests; un faible coût exprime des efforts moindres.

Le coût peut être exprimé comme une fonction ayant la signature suivante:

$$\underline{\text{coût}} : \textit{Ensemble de puissance}(\textit{TESTS}) \rightarrow R_{\geq 0}$$

conditionnée par le fait que le coût augmente avec la taille d'une suite de tests:

$$T_1 \subseteq T_2 \Rightarrow \underline{\text{coût}}(T_1) \leq \underline{\text{coût}}(T_2)$$

Le coût dépend en général du nombre et de la longueur de tests élémentaires dans la suite de tests.

9 Conformité

9.1 Introduction

Le terme *conformité* (au sens du terme anglais "compliance") fait référence à la satisfaction aux prescriptions spécifiées dans la présente Recommandation. Le terme est utilisé pour faire la distinction entre conformité à la présente Recommandation et *conformité* (au sens du terme anglais "conformance") d'une instance sous test à une spécification donnée (voir CTMF).

La présente Recommandation définit un cadre pour l'utilisation des méthodes formelles dans les essais de conformité. Elle est destinée aux personnes chargées des implémentations, des tests et des spécifications, impliquées dans des essais de conformité afin de les aider à définir la conformité ainsi que les procédures d'essai d'une implémentation donnée, eu égard à une spécification fournie comme description formelle.

Le cadre de la présente Recommandation est présenté à un niveau élevé d'abstraction; elle est, par exemple, détachée des algorithmes de génération de tests spécifiques, voire même d'une technique de description formelle spécifique. Le cadre définit la terminologie, les concepts abstraits ainsi que les exigences minimales applicables aux concepts et les relations entre ces concepts. Par conséquent, l'utilisation du cadre nécessite l'instanciation de ces concepts avec des choix spécifiques pour la technique de description formelle (*SPECS*), pour l'ensemble de modèles (*MODS*), pour la relation d'implémentation (imp), pour les algorithmes de génération de tests, etc., tout en démontrant les conditions qui leur sont applicables et les relations entre eux.

Le présent paragraphe fournit de manière explicite les prescriptions pour qu'une telle instanciation soit conforme à la présente Recommandation. Par ailleurs, il identifie les hypothèses qui constituent la base de la procédure de tests de conformité dont la validation ne s'inscrit pas dans le domaine d'implémentation de la procédure de tests de conformité proprement dit. Les prescriptions applicables à chacun des paragraphes sont fournies dans des sous-paragraphes séparés du présent paragraphe. L'Annexe A fournit des instanciations possibles pour des techniques de description formelle.

9.2 Paragraphe 6: la signification de la conformité

Pour satisfaire aux dispositions du paragraphe 6 de la présente Recommandation, les parties impliquées dans la procédure de tests de conformité doivent identifier et convenir des éléments suivants:

1) une spécification paramétrée appelée $s : D_s \rightarrow \textit{SPECS}$ dans le paragraphe 6. Le paramètre espace D_s identifie toutes les possibilités d'options d'implémentation de la spécification. *SPECS* est un formalisme des spécifications instanciées. La spécification paramétrée est supposée être correcte et validée et sert de point de référence à la procédure de tests de conformité;

2) une instance sous test IUT ainsi que la déclaration de conformité d'implémentation correspondante appelée ICS_{IUT} au paragraphe 6. L'instance IUT munie de la déclaration ICS_{IUT} correspondante doit être statiquement conforme à $s : D_s \rightarrow \textit{SPECS}$, c'est-à-dire $ICS_{IUT} \in D_s$;

3) un formalisme de modélisation appelé *MODS* dans le paragraphe 6 tel que toute implémentation possible prise en compte peut être supposée modélisée par un élément de *MODS*;

4) une relation d'implémentation appelée imp dans le paragraphe 6 telle que $\underline{\text{imp}} \subseteq \textit{MODS} \times \textit{SPECS}$ ou si $\textit{SPECS} = \textit{Ensemble de puissance}(\textit{REQS})$ où *REQS* est un langage logique ou propriétaire, une relation de satisfaction désignée par sat dans le paragraphe 6 telle que $\underline{\text{sat}} \subseteq \textit{MODS} \times \textit{REQS}$.

Il est entendu que l'IUT est conforme à $s : D_s \rightarrow \textit{SPECS}$ si et seulement si $ICS_{IUT} \in D_s$ et le modèle de l'IUT m_{IUT} est en relation imp avec $s(ICS_{IUT})$, c'est-à-dire m_{IUT} imp $s(ICS_{IUT})$.

Si on utilise plus d'une spécification pour définir l'ensemble des modèles d'implémentation, les spécifications doivent alors être cohérentes (voir 6.4.4). L'instance IUT est conforme au recueil de spécifications si et uniquement si elle est conforme à chacune des spécifications.

9.3 Paragraphe 7: concepts de test

Pour satisfaire aux dispositions du paragraphe 7 de la présente Recommandation, les parties impliquées dans la procédure de tests de conformité doivent identifier et convenir des éléments suivants:

1) un testeur, une instance sous test IUT, un contexte de test et leurs interfaces mutuelles désignées par les points d'accès d'implémentation IAP (entre l'instance IUT et le contexte de test), les points de contrôle et d'observation PCO (entre l'instance IUT et le testeur);

2) une suite de tests désignée par *T* dans le paragraphe 7 telle que $T \subseteq TESTS$, où *TESTS* est une notation de test;

3) une fonction $C : MODS \rightarrow MODS$ modélisant le contexte de test;

4) pour chaque $t \in T$: une attribution de verdict $\underline{\text{verd}}_t : OBS \rightarrow$ {succès, échec} où *OBS* est un ensemble d'observations possibles effectuées pendant l'exécution de tests;

5) une fonction de modélisation de l'exécution de tests $\underline{\text{exec}} : TESTS \times MODS \rightarrow OBS$ qui est supposée modéliser correctement l'exécution d'un test élémentaire pour une IUT donnée contenue dans le contexte de test.

Il est entendu que l'exécution de tests de *T* sur IUT est réussie – IUT réussit *T* – si et seulement si $m_{IUT} \in P_T$ (voir 7.4.2).

9.4 Paragraphe 8: test de conformité

Pour satisfaire au paragraphe 8 de la présente Recommandation, les parties impliquées dans la procédure de tests de conformité doivent identifier et convenir des éléments suivants:

1) une fonction de génération de tests $\underline{\text{gen}}^{C,\underline{\text{imp}}} : SPECS \rightarrow$ *Ensemble de puissance*(*TESTS*) telle que les suites de tests générées soient saines pour le domaine applicable;

2) zéro, une ou plusieurs stratégies de réduction de la taille des suites de tests (voir 8.4.2);

3) facultativement, une fonction de couverture $\underline{\text{cov}}_F$: *Ensemble de puissance*(*TESTS*) $\rightarrow$ [0,1] par rapport à un modèle de défaut donné $F \subseteq MODS$. La fonction $\underline{\text{cov}}_F$ doit être démontrée par une fonction de couverture (voir 8.5);

4) facultativement, une fonction de coût coût : *Ensemble de puissance*(*TESTS*) $\rightarrow R_{\geq 0}$. Il doit être démontré que cette fonction est effectivement une fonction de coût (voir 8.6).

Il est entendu qu'une suite de tests générée, lorsqu'elle est exécutée conformément au paragraphe 9.3, fournit une indication de la conformité d'une IUT en conformité avec 9.2. Si ¬ (IUT réussit *T*), l'instance IUT n'est pas conforme; mais si l'instance IUT réussit *T*, on ne peut, dans ce cas, que conclure à l'absence d'une indication de non-conformité.

Annexe A

La présente annexe traite de l'interprétation de la terminologie dans les techniques FDT Estelle, LOTOS et SDL. Elle ne définit pas de nouveaux termes qui ne font pas déjà partie du corps principal de la présente Recommandation. Elle fournit une interprétation spécifique au langage pour les termes dont il est utile de tenir compte dans le contexte spécifique d'une de ces techniques FDT.

NOTE – La présente annexe ne fournit que des exemples à l'appui des définitions générales et n'entend pas indiquer un style recommandé de spécification pour la testabilité. Elle est de nature descriptive plutôt que prescriptive et informative plutôt que normative.

A.1 Spécifications

A.1.1 Introduction

Les spécifications sont des descriptions de comportement dans l'une des techniques FDT normalisées Estelle, LOTOS ou SDL. L'ensemble *SPECS* représente l'ensemble de toutes les spécifications instanciées rédigées dans une FDT particulière.

Il existe un exemple pour illustrer la plupart de la terminologie utilisée dans la présente Recommandation pour les techniques FDT Estelle, LOTOS et SDL. Ceci permet de comparer les différentes FDT et leur corrélation à la terminologie utilisée dans la présente Recommandation.

L'exemple commun décrit le comportement d'une entité demandeuse dans un protocole orienté connexion, c'est-à-dire la première fois qu'une connexion est construite avant que des données ne puissent être envoyées. Une description informelle du protocole est fournie ci-après. Cette description informelle n'est pas destinée à fournir une description précise et complète du comportement du protocole. Elle décrit le comportement du protocole dans une mesure suffisante pour son instanciation comme exemple commun dans les différentes FDT.

Exemple commun

L'exemple commun décrit une entité demandeuse dans un protocole orienté connexion. L'architecture est donnée dans la Figure A.1.

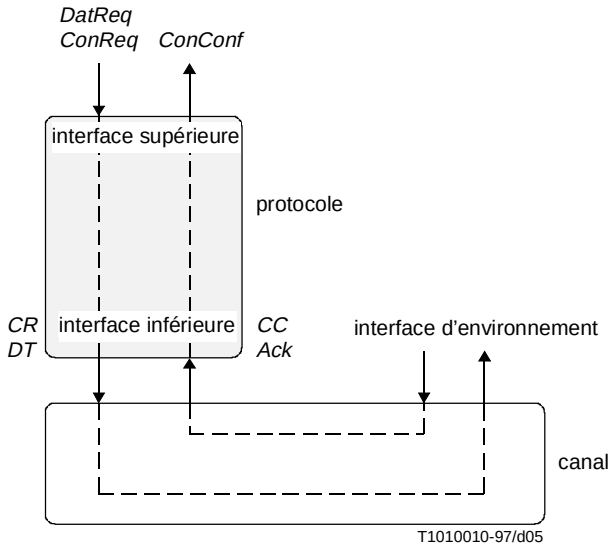

Figure A.1/Z.500 – Architecture de l'exemple commun

On peut distinguer deux phases: la *phase de connexion* (cette phase traite de l'établissement de la connexion) et la *phase de données* (au cours de cette phase, des données sont envoyées par le demandeur). Les contraintes suivantes s'appliquent à la phase de connexion:

– à l'état initial, le demandeur attend qu'apparaît une demande de connexion de niveau supérieur (*ConReq*);

– une demande de connexion de niveau supérieur (*ConReq*) est suivie d'une demande de connexion de niveau inférieur (*CR*);

– après une demande de connexion de niveau inférieur (*CR*), une confirmation de connexion de niveau inférieur (*CC*) est reçue de l'entité récepteur;

– une confirmation de connexion de niveau inférieur (*CC*) est suivie d'une confirmation de connexion de niveau supérieur (*ConConf*);

– à tout moment, l'entité demandeur peut recevoir une demande de déconnexion (*DisReq*) au niveau supérieur. Une demande de déconnexion ramène le protocole à l'état initial.

On entre dans la phase de données après établissement d'une connexion. Au cours de la phase de données, les données peuvent être transmises du demandeur vers le récepteur. Les restrictions suivantes s'appliquent à la phase de données:

– la phase de données ne peut commencer qu'après émission d'une confirmation de connexion de niveau supérieur (*ConConf*);

– une demande de données de niveau supérieur (*DatReq*) est suivie par une donnée de niveau inférieur (*DT*);

– après une donnée de niveau inférieur (*DT*), il peut y avoir un acquittement de niveau inférieur (*AK*);

– ce n'est qu'après réception d'un acquittement de niveau inférieur (*AK*) qu'une nouvelle demande de données de niveau supérieur (*DatReq*) peut avoir lieu;

– pendant la phase de données, l'entité demandeur peut recevoir une demande de déconnexion (*DisReq*) au niveau supérieur, ce qui ramène le protocole à l'état initial.

L'exemple commun comporte également une description de l'architecture de test, afin d'illustrer les concepts du 7.2. L'entité demandeur peut directement accéder à l'interface supérieure par le testeur tandis qu'on ne peut accéder à l'interface inférieure que par l'intermédiaire d'un *support d'informations*. Ce support d'informations est supposé ne transmettre que des messages correctes ou souples. Il ne peut créer, dupliquer ou redemander des messages.

A.1.2 Les spécifications dans Estelle

Une description Estelle de l'exemple commun présenté au A.1.1 est illustrée par la Figure A.2. Il est à noter qu'une approche non déterministe est utilisée pour modéliser le fait que le canal peut perdre des messages.

```
specification Example;
  default individual queue;
  timescale seconds;

  channel ISAP(User, Protocol);
    by User    : ConReq;DatReq;Dis;
    by Protocol: ConConf;

  channel MSAP(Protocol, Channel_S);
    by Protocol : CR;DT;
    by Channel_S: CC;Ack;

  module Protocol_M
    ip U: ISAP(Protocol);
       L: MSAP(Protocol)
  end;

  body Protocol_B for Protocol_M

  state disconnected, wait, connected, sending;

  initialize to disconnected begin end;

  trans

  from disconnected to wait
    when U.ConReq
      begin output L.CR end;

  from wait to connected
    when L.CC
      begin output U.ConConf end;


  from connected to sending
    when U.DatReq
      begin output L.DT end;

  from sending to connected
    when L.Ack

      begin end;

  from wait, connected, sending to disconnected
    when U.Dis
      begin end;

  end;
```

```
module Channel_M
   ip E: MSAP(Protocol)
      L: MSAP(Channel_S);
end;

body Channel_B for Channel_M

state empty;

initialize to empty begin end;

trans

when L.CR
   begin output E.CR end;
   begin end;

when E.CC
   begin output L.CC end;
   begin end;

when L.DT
   begin output E.DT end;
   begin end;

when E.Ack
   begin output L.Ack end;
   begin end;

end;

modvar Protocol_I: Protocol_M;
       Channel_I: Channel_M;


initialize
   begin
      init Protocol_I with Protocol_B;
      init Channel_I with Channel_B;
      connect Protocol_I.L to
   Channel_I.L
   end;

end.
```

Figure A.2/Z.500 – Spécification de l'exemple commun pour Estelle

Pour définir la conformité, des relations d'implémentation seront définies au A.4.2. Cependant, il n'est pas approprié de définir des relations d'implémentation significatives sur la base de la description syntactique d'Estelle. Ces relations sont, en général, définies en exprimant, en premier lieu, la sémantique de la spécification Estelle en une sorte de modèle (mathématique) plus fondamental et puis en formulant des relations d'implémentation sur la base de ce modèle mathématique.

La sémantique de la spécification Estelle est exprimée dans une sorte de système de transition étiqueté dans lequel on fait la distinction entre des actions en entrée et des actions en sortie. On appelle ces systèmes automates à entrées/sorties (IOSM, *input-output state machines*).

définition 1 (IOSM): un automate à entrées/sorties (IOSM) est un 4-tuple $M = \langle S, L, T, s_0 \rangle$ où:

– S est un ensemble d'états fini, non vide;

– L est un ensemble d'interactions fini, non vide;

– $T \subseteq S \times ((\{?,!\} \times L) \cup \{\tau\}) \times S$ est la relation de transition. Chaque élément de T est une transition, à partir d'un état origine vers un état destination. Cette transition est associée soit à une action observable (entrée *?a* ou sortie *!a*), soit à l'action interne τ;

– s_0 est l'état initial de l'automate IOSM.

L'ensemble *SPECS* est choisi comme étant l'ensemble de tous les automates IOSM. La Figure A.3 illustre le modèle IOSM de la spécification Estelle pour l'exemple commun. Il est à noter que le modèle IOSM ne permet pas d'indiquer à quel point d'interaction un message donné apparaît. C'est pourquoi des messages ont été renommés pour indiquer différents points d'interaction.

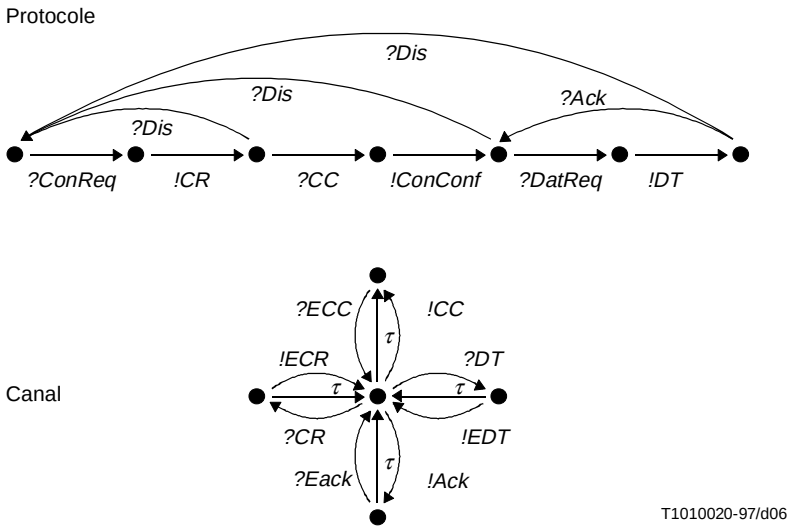

Figure A.3/Z.500 – Modèle IOSM de la spécification Estelle, à partir de la Figure A.2

NOTE – Il n'y a pas encore de travaux établis et bien acceptés à ce sujet. La présentation ci-dessus est fondée sur les travaux décrits dans [Phalippou 92] et [Phalippou 94], un de ceux qui sont actuellement disponibles dans ce domaine. D'autres modèles mathématiques pourraient être utilisés pour décrire la sémantique de la spécification Estelle d'une manière convenable pour une définition des relations d'implémentation.

Nous mentionnerons à cet égard:

– les machines Mealy [Hopcroft 79] sur lesquelles est fondée la plupart des méthodes de génération de tests classiques pour les systèmes à entrées/sorties (W, UIO, DS, etc.);

– un modèle dérivé des machines Mealy appelé PNFSM (extensions partiellement spécifiées et non déterministes des machines Mealy) [Luo 93].

A.1.3 Spécifications dans LOTOS

Une spécification LOTOS est une construction de langage satisfaisant à la syntaxe décrite dans [ISO 8807]. L'objectif d'une spécification LOTOS est de décrire le comportement d'un système. Des spécifications de système à différents niveaux d'abstraction peuvent être obtenues en omettant des détails de système interne non pertinents.

Les contraintes suivantes sont inhérentes aux spécifications LOTOS:

– LOTOS suppose que la communication est *synchrone*, ce qui signifie que des entités y participent par interactions, de manière synchrone;

– la communication est *atomique*, ce qui signifie que la communication entre les entités participantes est soit réussie pour toutes les entités, soit n'est réussie pour aucune d'entre elles;

– LOTOS n'est capable que de décrire des aspects de synchronisation temporelle; les aspects de synchronisation absolue ne peuvent être décrits.

Une spécification LOTOS de l'exemple commun décrit au A.1.1 est illustrée à la Figure A.4.

```
(* specification : Initiator (BASIC LOTOS)

   This specification describes the Initiator Protocol *)

specification Initiator_Protocol[ConReq, ConConf, DatReq, CR, CC
                                 DT,Ack, Dis]                    : exit
behaviour
   Initiator [ConReq, ConConf, DatReq, CR, CC, DT, Ack, Dis]
where
   process Initiator [ConReq, ConConf, DatReq, CR, CC, DT, Ack, Dis] : exit :=
           Connection_Phase [ConReq, CR, CC, ConConf, Dis ] >>
           Data_Phase[ConReq, ConConf, DatReq, CR, CC, DT, Ack, Dis ]
   endproc (* Initiator *)

   process Connection_Phase [ConReq, CR, CC, ConConf, Dis] : exit :=
           ConReq ; CR; (CC ; ConConf ; exit
                         []
                         Dis; Connection_Phase [ConReq, CR, CC, ConConf, Dis])
   endproc (* Connection_Phase *)

   process Data_Phase [ConReq, ConConf, DatReq, CR, CC,
                       DT, Ack, Dis]                    : exit :=
           DatReq ; DT ; (Ack ; Data_Phase[ConReq, ConConf, DatReq, CR, CC,
                                           DT, Ack, Dis]
                          [] Dis; Initiator [ConReq, ConConf, DatReq, CR,
                                             CC, DT, Ack, Dis])
           []
           Dis; Initiator [ConReq, ConConf, DatReq, CR, CC, DT, Ack, Dis]
   endproc (* Initiator *)
endspec (* Initiator_Protocol *)
```

Figure A.4/Z.500 – Spécification LOTOS du protocole demandeur

La sémantique des spécifications LOTOS est indiquée par l'expression systèmes de transition étiquetés (LTS). Les systèmes de transition étiquetés sont représentés par des graphes dirigés dont les bords sont étiquetés.

définition 2 (système de transition étiqueté): un système de transition étiqueté est 4-tuple $TS = \langle S, L, T, s_0 \rangle$ tel que:

– S est un ensemble d'états non vide (dénombrable);

– L est un ensemble d'actions observables (dénombrable);

– $T \subseteq S \times (L \cup \{\tau\}) \times S$ est la relation de transition, dans laquelle l'étiquette spéciale $\tau \notin L$ représente une action non observable (ou interne);

– $s_0 \in S$ est l'*état initial*.

Il n'est fait aucune distinction entre les actions d'entrée et les actions de sortie étant donné que la communication est synchrone et que, par conséquent, la notion d'entrées et de sorties n'existe pas. La communication asynchrone est modélisée dans LOTOS par modélisation explicite du support intermédiaire entre des entités communicantes (voir A.6.3).

Le formalisme des systèmes de transition étiquetés est très représentative des formalismes de description de comportement. De nombreux autres formalismes (par exemple, FSM, EFSM, IOSM) peuvent être exprimés en termes de systèmes de transition étiquetés.

Pour des spécifications écrites en langage LOTOS, l'ensemble *SPECS* est instancié par l'ensemble de tous les systèmes de transition étiquetés *LTS* qui peuvent représenter des spécifications LOTOS. La Figure A.5 illustre le système de transition étiqueté correspondant à l'exemple commun décrit au A.1.1. L'état initial est indiqué par un point noir.

A.1.4 Spécifications dans SDL

Une spécification en langage SDL de l'exemple commun décrit au A.1.1 est illustrée aux Figures A.11, A.12, A.13, A.14 et A.15.

La conformité dynamique concerne uniquement le comportement externe d'un système. Une spécification SDL n'exprime pas explicitement le comportement externe d'un système. Ainsi, pour tester la conformité dynamique, il est nécessaire d'avoir une représentation du comportement observable tiré de la spécification SDL. Une représentation ACT, *asynchronous communication tree* (arbre de communication asynchrone) ou LTS, *labelled transition system* (système de transition étiqueté) convient à l'expression du comportement externe dans une spécification SDL. Pour une définition formelle des représentations ACT, se reporter à [Hogrefe 88].

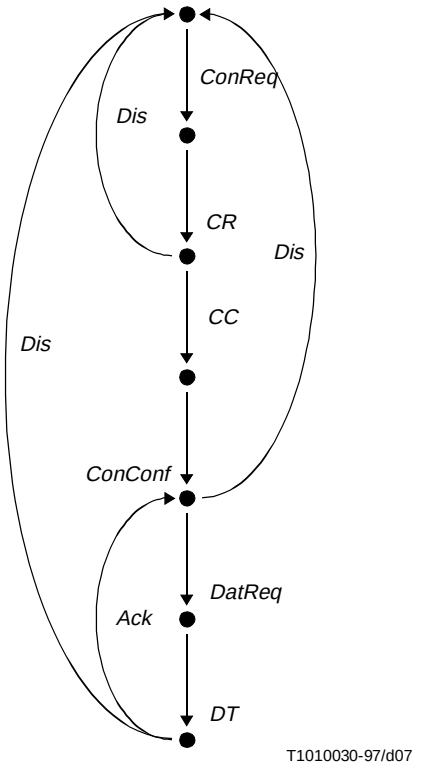

Figure A.5/Z.500 – Modèle LTS de spécification LOTOS

Ainsi, lorsqu'une SDL est utilisée pour spécifier des systèmes, l'ensemble *SPECS* est l'ensemble de toutes les ACT ou LTS qui peuvent résulter de toute spécification SDL. La Figure A.6. illustre, pour exemple, une petite partie de représentation ACT du comportement observable. La représentation ACT associe à chaque information d'état des signaux dans les différents canaux. La Figure A.6 montre uniquement les canaux qui, en l'état actuel, transportent un signal. Si tous les canaux du système sont vides, ceci est représenté par $Q_i = \langle\,\rangle$. L'information d'état est cependant inutile lorsque des tests élémentaires sont définis à partir de la représentation ACT.

Le terme spécification SDL ne fait référence, dans ce contexte, qu'à des *spécifications instanciées*. Des spécifications SDL génériques représentent un certain nombre de spécifications instanciées différentes, en d'autres termes, une spécification SDL générique représente un ensemble de représentations dans l'ensemble *SPECS*.

A.2 Options d'implémentation et spécifications instanciées

A.2.1 Introduction

Le formulaire ICS (IPf) d'une spécification de protocole formelle est décrit comme étant ses paramètres formels et la déclaration ICS d'une implémentation comme étant ses paramètres réels. Idéalement, la spécification formelle devrait être paramétrée de façon que la revue de conformité statique ne soit plus qu'un contrôle de type dans la FDT. Ceci donnerait à ces concepts une base sémantique bien étudiée.

Il existe deux aspects des options d'implémentation:

– Les variables, qui sont utilisées, par exemple, pour représenter les options (par exemple, des réponses à des questions Oui/Non).

– Des limites aux valeurs possibles de ces variables.

Le premier aspect est facilement modélisé dans n'importe quelle FDT qui permet de paramétrer des spécifications au moyen de valeurs. En ce qui concerne le second aspect, ces limites peuvent être vérifiées par un prédicat (fonction/expression booléenne) au moyen des paramètres. Cependant, il est difficile de modéliser les restrictions de manière telle que la revue de conformité statique serait réduite à un contrôle de type.

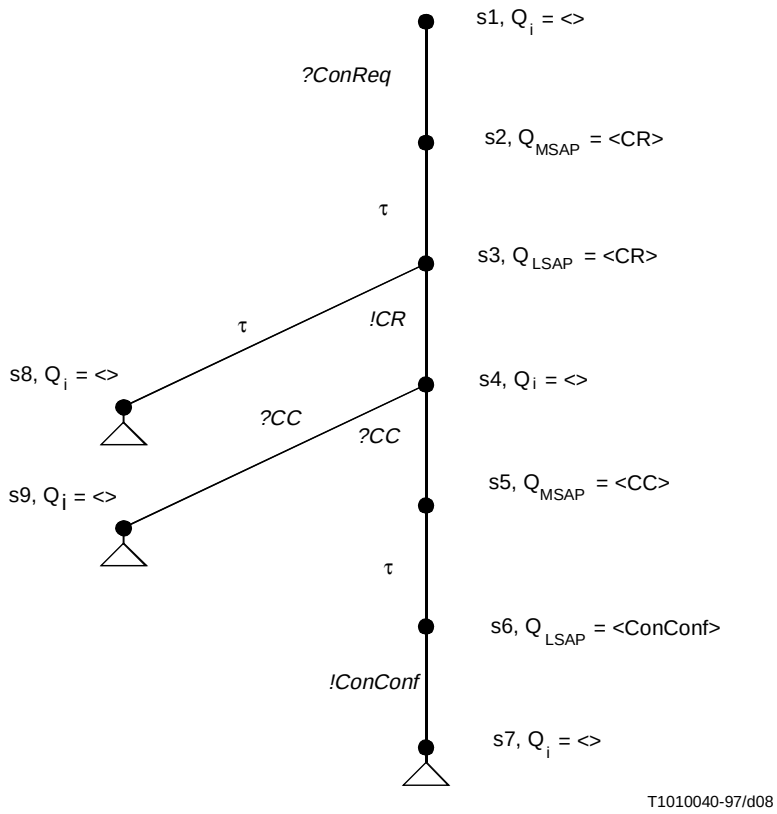

Figure A.6/Z.500 – Une représentation ACT d'une partie du comportement observable

Le sous-paragraphe ci-après donne des exemples (limités) qui, à l'heure actuelle, justifient la conclusion suivante:

> L'IPf d'une spécification de protocole formelle peut être décrit comme étant ses paramètres formels et la déclaration ICS d'une implémentation comme étant ses paramètres réels, de sorte que la revue de conformité statique est décrite par un prédicat dans la spécification formelle.

Une version simplifiée du protocole de transport est ici traitée pour exemple. Dans cette version, on tient uniquement compte des exigences relatives aux classes prises en charge. Les prescriptions sont tirées du Tableau D.5.2 du formulaire PICS du protocole de transport [ISO 8703] (limitées aux classes 0 à 4).

Chaque ligne du présent tableau identifie une option unique dont la valeur peut être oui ou non (colonne prise en charge). La lettre O dans le champ état signifie que la capacité est en option. Le réalisateur peut répondre par oui ou par non. O.1 dans C0 et C2 signifie que les deux sont en option, mais qu'il convient que le réalisateur prenne en charge l'une des deux (et, par conséquent, la réponse oui devrait être indiquée pour au moins l'une des deux). L'état C0:O dans la colonne C1 signifie que C1 est en option si C0 est pris en charge, sinon, C1 est interdit.

Tableau A.1/Z.500 – Classes du protocole de transport prises en charge

Indice	Classe	Références	Etat	Prise en charge
C0	classe 0	14	O.1	oui non
C1	classe 1	14	C0:O	oui non
C2	classe 2	14	O.1	oui non
C3	classe 3	14	C2:O	oui non
C4	classe 4	14	C2:O	oui non

Pour résumer, ce tableau impose les restrictions suivantes:

– la classe 0 ou la classe 2 doit au moins être mise en œuvre;

– la classe 1 nécessite la classe 0;

– les classe 3 et classe 4 nécessitent toutes deux la classe 2.

Les sous-paragraphes suivants contiennent des spécifications des techniques FDT Estelle, LOTOS et SDL qui représentent uniquement les principaux ingrédients, dans la mesure où la modélisation du formulaire PICS est concernée.

A.2.2 Options d'implémentation et spécifications instanciées dans Estelle

Estelle ne fournit pas de constructions intégrées pour les options de modélisation; il est cependant possible de modéliser les formulaires de déclaration ICS et PICS en utilisant des constructions en langage normalisé. Ceci peut être réalisé de diverses manières. L'exemple ci-dessous illustre une méthode de modélisation de l'option d'implémentation et des spécifications instanciées dans Estelle.

Le formulaire de déclaration ICS est modélisé par un enregistrement avec champs booléens qui représentent les champs *index* du formulaire ICS. Les contraintes sur ces champs (c'est-à-dire les champs *état* du formulaire ICS) sont vérifiées par une fonction appelée `Static_conformance_review`. La spécification paramétrée étant modélisée par un module Estelle paramétré, la spécification instanciée est alors créée en attribuant une valeur à ce paramètre dans la transition `initialize`.

```
specification Transport_Protocol;

type PICS_Proforma =
     record
       C0 : boolean; (* ref. 14 *)
       C1 : boolean; (* ref. 14 *)
       C2 : boolean; (* ref. 14 *)
       C3 : boolean; (* ref. 14 *)
       C4 : boolean; (* ref. 14 *)
     end;

var PICS : PICS_Proforma;

function Static_Conformance_Review (P : PICS_Proforma) : boolean;
        (* check static conformance requirements: *)
        (* (C0 or C2) and (C1 => C0) and ((C3 or C4) => C2) *)
        begin
          Static_Conformance_Review := ((P.C0 or P.C2) and
                                        (not(P.C1) or P.C0) and
                                        (not(P.C3 or P.C4) or P.C2));
        end;

procedure Get_PICS_Value (var P : PICS_Proforma);
primitive; (* get PICS value *)

(* Channel, Module header, Module body and Module variable definitions *)

initialize
   begin
     Get_PICS_Value(PICS);
     if (Static_Conformance_Review(PICS)) then
     begin
      (* instantiate appropriate modules *)
       ...
     end
   end;
end.
```

A.2.3 Options d'implémentation et spécifications instanciées dans LOTOS

Une spécification LOTOS peut être explicitement paramétrée. Les types de données ("sortes") utilisés dans la liste de paramètres sont définis globalement. Dans l'exemple, le seul type de données utilisé est booléen, tiré de la bibliothèque normalisée de types prédéfinis. Des améliorations futures dans LOTOS permettront de décrire les conditions de conformité statique de façon que la revue de conformité statique revienne à l'exécution d'un contrôle de type.

```
SPECIFICATION Transport_Protocol [t,n] (c0,c1,c2,c3,c4 : bool) : NOEXIT

LIBRARY Boolean
ENDLIB

BEHAVIOUR ...

ENDSPEC (* Transport_Protocol *)
```

Il est admis d'utiliser des paramètres dans les expressions booléennes et, en particulier, dans ce qu'on appelle les protections de manière à sélectionner/limiter un comportement ultérieur.

```
SPECIFICATION Transport_Protocol [t,n] (c0,c1,c2,c3,c4 : bool) : NOEXIT

LIBRARY Boolean
ENDLIB

BEHAVIOUR TPEntity[t,n](c0,c1,c2,c3,c4)

WHERE

   PROCESS TPEntity [t,n] (c0,c1,c2,c3,c4 : bool) : NOEXIT :=
           ...
           [c4] -> Splitting[t,n]
           []
           [not(c4)] -> NoSplitting[t,n]
           ...
   ENDPROC (* TPEntity *)

ENDSPEC (* Transport_Protocol *)
```

L'exemple suivant illustre l'utilisation des conditions de conformité statique.

```
SPECIFICATION Transport_Protocol [t,n] (c0,c1,c2,c3,c4 : bool) : NOEXIT

LIBRARY Boolean
ENDLIB

BEHAVIOUR [ClassesConform(c0,c1,c2,c3,c4)] -> TPEntity[t,n](c0,c1,c2,c3,c4)

WHERE

   TYPE StaticConformance IS Boolean
   OPNS ClassesConform : bool, bool, bool, bool, bool -> bool
   EQNS FORALL c0, c1, c2, c3, c4 : bool
        OFSORT bool
        ClassesConform(c0,c1,c2,c3,c4) = (c0 or c2) and
                                         ((c3 or c4) implies c2) and
                                         (c1 implies c0)
   ENDTYPE (* StaticConformance *)

   PROCESS TPEntity [t,n] (c0,c1,c2,c3,c4 : bool) : NOEXIT :=
           ...
   ENDPROC (* TPEntity *)

ENDSPEC (* Transport_Protocol *)
```

NOTE – Cette méthode de spécification des conditions de conformité statique présente cependant un inconvénient: si les paramètres réels ne satisfont pas à la conformité de classes (ClassesConform), le comportement spécifié est STOP (arrêt). Ceci signifie qu'une implémentation qui ne fait rien du tout serait considérée conforme! Des remarques similaires peuvent également s'appliquer à des exemples tirés d'autres techniques FDT. Ce problème est résolu en utilisant l'une des améliorations de LOTOS, c'est-à-dire le concept module. Ensuite, un choix est donné qui n'est certainement pas recommandé, étant donné qu'il donne lieu à des spécifications très obscures. Il est uniquement fourni pour illustrer la manière dont le problème ci-dessus peut être éludé. On utilise un type de données dont les valeurs sont toutes les combinaisons possibles de valeurs statiquement conformes. Cette solution est également utilisable pour les autres techniques FDT.

```
SPECIFICATION Transport_Protocol [t,n] (classes : ValidClasses) : NOEXIT

LIBRARY Boolean
ENDLIB

TYPE  ValidClasses
SORTS ValidClasses
OPNS  only0, only01, only2, only23, only24, only234,
      only02, only023, only024, only0234,
      only012, only0123, only0124, only01234:        -> ValidClasses
ENDTYPE (* ValidClasses *)

BEHAVIOUR

  LET c0:bool = has0(classes),
      c1:bool = has1(classes),
      c2:bool = has2(classes),
      c3:bool = has3(classes),
      c4:bool = has4(classes)
  IN TPEntity[t,n](c0,c1,c2,c3,c4)
```

```
  WHERE
  TYPE SupportedClasses IS ValidClasses, Boolean
  OPNS has0, has1, has2, has3, has4 : ValidClasses -> bool
  EQNS OFSORT bool
       has0(only0)  = true
       has0(only01) = true
       ...
  ENDTYPE (* SupportedClasses *)

  ...

ENDSPEC (* Transport_Protocol *)
```

A.2.4 Options d'implémentation et spécifications instanciées dans SDL

Dans le cadre général FMCT, des spécifications peuvent être paramétrées de manière à permettre différentes *options d'implémentation*. Pour une implémentation spécifique, les options sélectionnées sont définies dans le document ICS. Lorsque ces valeurs de paramètre sont appliquées à la spécification paramétrée, le résultat en est une *spécification instanciée*.

Dans le système générique SDL, des spécifications sont utilisées pour spécifier les systèmes paramétrés. Lorsque les paramètres ont été appliqués, la spécification instanciée est désignée par le terme *spécification système spécifique*.

Dans l'exemple suivant, des paramètres sont indiqués en les déclarant `EXTERNAL`. Afin de vérifier la déclaration ICS de conformité statique, il est introduit un canal particulier: le canal erreur. Si la spécification est paramétrée par une déclaration ICS statiquement non conforme, la valeur booléenne `FALSE` est envoyée sur ce canal; autrement, rien n'est transmis sur le canal.

```
SYSTEM TransportProtocol

SYNONYM C0 Boolean = EXTERNAL;
SYNONYM C1 Boolean = EXTERNAL;
SYNONYM C2 Boolean = EXTERNAL;
SYNONYM C3 Boolean = EXTERNAL;
SYNONYM C4 Boolean = EXTERNAL;

CHANNEL ErrorChannel
  /* Channel to convey FALSE if the static conf. review fails */
  FROM StatConfReview TO ENV
  WITH Boolean;
ENDCHANNEL;

BLOCK StatConfReview;
  /* This block contains only one process which performs
     the static conformance review */

  SIGNALROUTE ErrorRoute
  FROM StaticReviewer TO ENV
  WITH Boolean;

  CONNECT ErrorChannel AND ErrorRoute;

  PROCESS StaticReviewer (1, 1);
          DCL correct Boolean;

          START;
          TASK correct := (C0 OR C2) AND ((C3 OR C4) => C2) AND (C1 => C0);
          /* if correct = true, then the PICS conform */
          DECISION correct;
                   (TRUE) : STOP;          /* conforming: do nothing    */
                   (FALSE): OUTPUT FALSE; /* not conf: send FALSE via
                                             channel ErrorChannel       */
          ENDDECISION;
  ENDPROCESS;
ENDBLOCK;

BLOCK
  ...
ENDBLOCK;

...

ENDSYSTEM;
```

L'exemple illustre la manière dont la déclaration ICS peut être codée dans la spécification SDL, en termes de paramétrage de la spécification SDL.

Il est admis que des spécifications de système soient définies pour permettre la mise en œuvre d'options différentes. Il peut y avoir des dépendances entre les options d'implémentation. Les options d'implémentation et leurs éventuelles dépendances sont définies dans le formulaire de déclaration ICS. Dans la norme [ISO 9646], les options réelles mises en œuvre dans une implémentation donnée sont spécifiées dans la déclaration de conformité de l'implémentation. L'exemple illustre la manière dont la déclaration ICS peut être codée dans la spécification SDL en termes de paramétrage de la spécification SDL.

A.3 Implémentations et modèles d'implémentations

A.3.1 Introduction

Une implémentation est un objet physique, non formel. Pour toutes les techniques FDT, l'ensemble *IMPS* représente l'ensemble de toutes les implémentations. Les implémentations étant des objets non formels, il n'est pas possible de les soumettre à un raisonnement mathématique. Un modèle est une abstraction formelle de l'implémentation qui convient à des raisonnements mathématiques sur des implémentations.

Les tests sont destinés à construire un modèle d'une implémentation fondée sur les observations qui peuvent être tirées d'expérimentations effectuées sur l'implémentation. Le type d'observations qui peut être effectué sur l'implémentation après expérimentation détermine le niveau d'abstraction du modèle construit. En général, plus on observe l'implémentation et plus le modèle que l'on peut construire pour cette implémentation sera détaillé.

Dans la pratique, l'implémentation ne fait l'objet que d'un nombre limité d'expérimentations. Du fait de la faisabilité pratique, il n'est pas possible de modéliser de manière exacte chaque aspect de l'implémentation. Il convient que le modèle obtenu au moyen de tests reflète uniquement le comportement de l'implémentation pour les expérimentations effectuées.

A.3.2 Implémentations et modèles d'implémentations dans Estelle

Les implémentations sont des objets physiques dont l'ensemble est représenté par *IMPS*. Cependant, pour des études de test à un niveau théorique, il est nécessaire de modéliser des implémentations par des objets mathématiques. Pour la modélisation des implémentations, les choix sont semblables à ce dont disposent les spécifications. Une possibilité consisterait à décrire des implémentations dans Estelle. Cependant, les objectifs de test (où le but est de définir des relations d'implémentation), il est plus pertinent de représenter des implémentations par des IOSM (automates à entrées/sorties). Par conséquent, l'ensemble *MODS* est choisi pour être l'ensemble de tous les automates IOSM *IOSM*.

Des exemples de ces implémentations du protocole de la Figure A.3 sont illustrés à la Figure A.7. Nous nous sommes limités à des implémentations qui sont "proches" de la spécification car, lors du développement des produits, il est peu probable que la résultante soit très éloignée de ce qui était prévu. Cependant, comme cela est expliqué dans le sous-paragraphe suivant, certaines de ces implémentations sont conformes et quelques autres ne le sont pas.

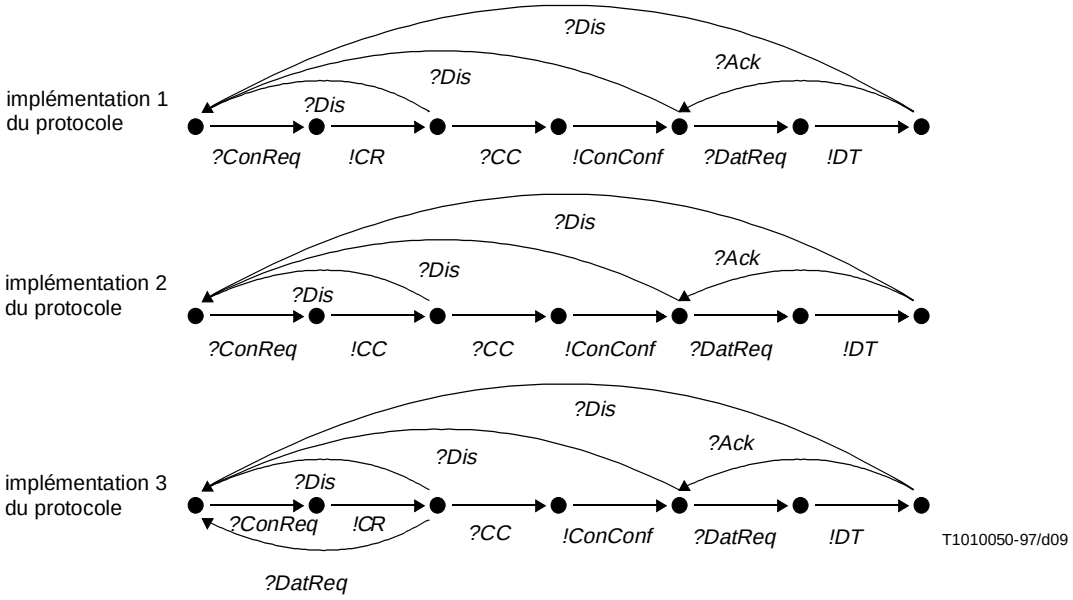

Figure A.7/Z.500 – Modèles IOSM d'implémentations

A.3.3 Implémentations et modèles d'implémentations dans LOTOS

Une condition préalable pour la construction d'un modèle d'implémentation lorsque l'on revendique la mise en œuvre d'un système décrit par une spécification LOTOS est l'existence d'un formalisme approprié d'expression du modèle (par exemple, le formalisme doit être capable d'exprimer tous les détails pertinents pour lesquels les tests sont réalisés). Le formalisme doit également être choisi de manière qu'il soit facile d'établir la conformité de l'implémentation à la spécification LOTOS [à savoir, si le modèle de l'implémentation est lié à la spécification LOTOS par la relation d'implémentation, (voir A.4.3)].

De nombreux formalismes peuvent être utilisés pour décrire des modèles d'implémentations (par exemple, des systèmes de transition). Cependant, pour des implémentations de systèmes décrits par des spécifications LOTOS, il est, dans la pratique, commun de choisir le formalisme d'un modèle *MODS* pour qu'il soit égal à l'ensemble du système de transition étiqueté *LTS*.

La Figure A.8 représente quelques modèles d'implémentations du système décrit par la spécification de la Figure A.5.

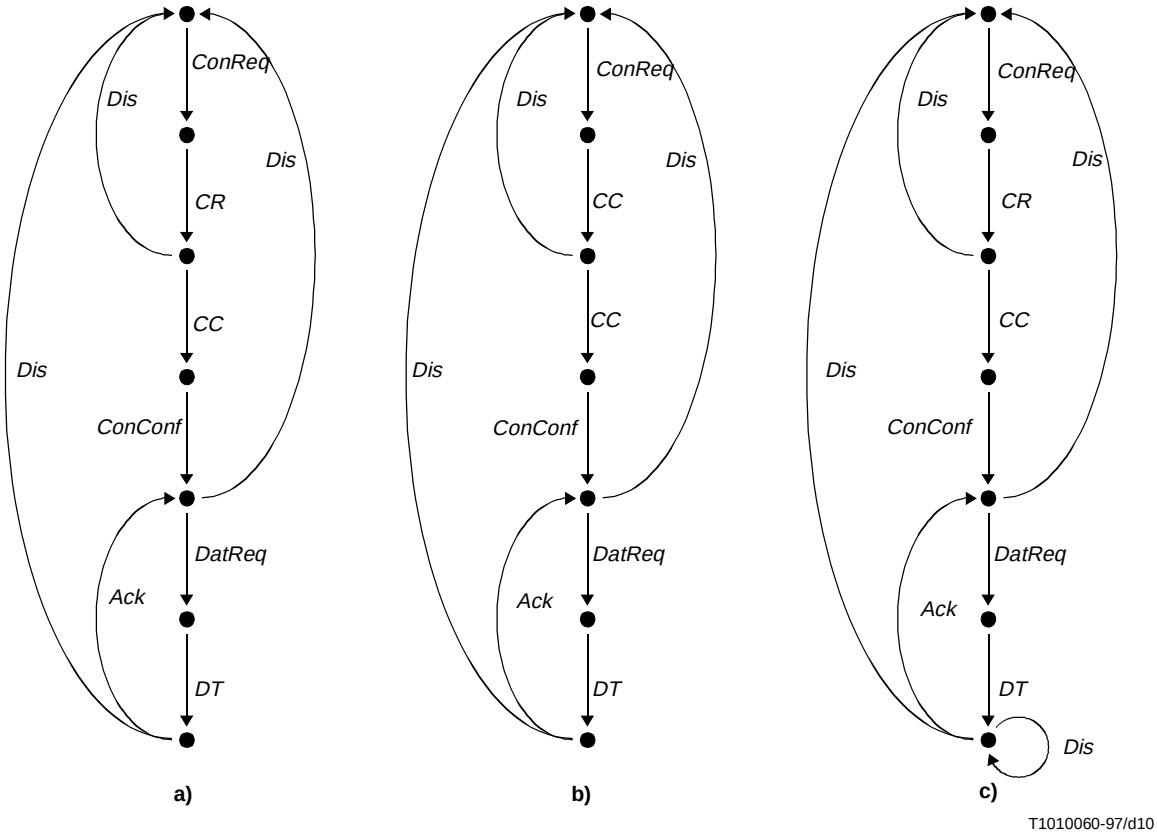

Figure A.8/Z.500 – Modèles LTS d'implémentations

A.3.4 Implémentations et modèles d'implémentations dans SDL

Les implémentations désignent des systèmes exécutables ou physiques mettant en œuvre des systèmes spécifiés dans SDL. L'ensemble *IMPS* contient toutes ces implémentations.

Des méthodes formelles de procédure de tests de conformité peuvent être appliquées uniquement lorsque les éléments de la procédure sont des éléments ayant une structure sémantique formelle. Par conséquent, l'implémentation de l'ensemble *IMPS* ne peut être utilisée à cette fin. On suppose, cependant, que pour chaque implémentation dans *IMPS* il existe un modèle formel qui représente les propriétés de l'implémentation. Cette hypothèse est appelée *hypothèse de test* et l'acceptation de cette hypothèse est primordiale pour l'ensemble de la méthode formelle.

Pour chaque implémentation dans *IMPS*, il existe au moins un modèle dans l'ensemble des modèles *MODS*. S'il existe plusieurs modèles pour une implémentation particulière, ceux-ci sont similaires dans la mesure où les tests ne peuvent pas les distinguer. L'ensemble *MODS* doit également être constitué de modèles qui peuvent servir de base à la définition de relations formalisées liées à l'ensemble des spécifications *SPECS*. Il est également admis que le formalisme utilisé dans les deux ensembles soit le même, par exemple, ACT ou LTS.

Pour illustrer les concepts du cadre formel, les implémentations suivantes du bloc protocole sont prises comme hypothèses. Les implémentations sont désignées par I_1, I_2, et I_3:

- I_1 est une implémentation qui dispose d'un modèle identique au modèle de la spécification. Ceci signifie que la description du bloc pour le protocole illustré aux Figures A.12 et A.13 peut également être perçue comme un modèle pour cette implémentation.
- I_2 utilise le protocole tel que spécifié à la Figure A.13 également, sauf pour ce qui concerne le comportement qui peut être exprimé en SDL comme illustré à la Figure A.9. Ce qui signifie que l'implémentation envoie un signal *DT* au lieu d'un *CR* lorsqu'une demande de connexion est reçue.
- de la même manière, I_3 met en œuvre le protocole tel que défini dans la spécification SDL. Par ailleurs, dans l'état *connecté*, elle peut recevoir une nouvelle demande de connexion et relancer la phase de connexion comme illustré à la Figure A.9.

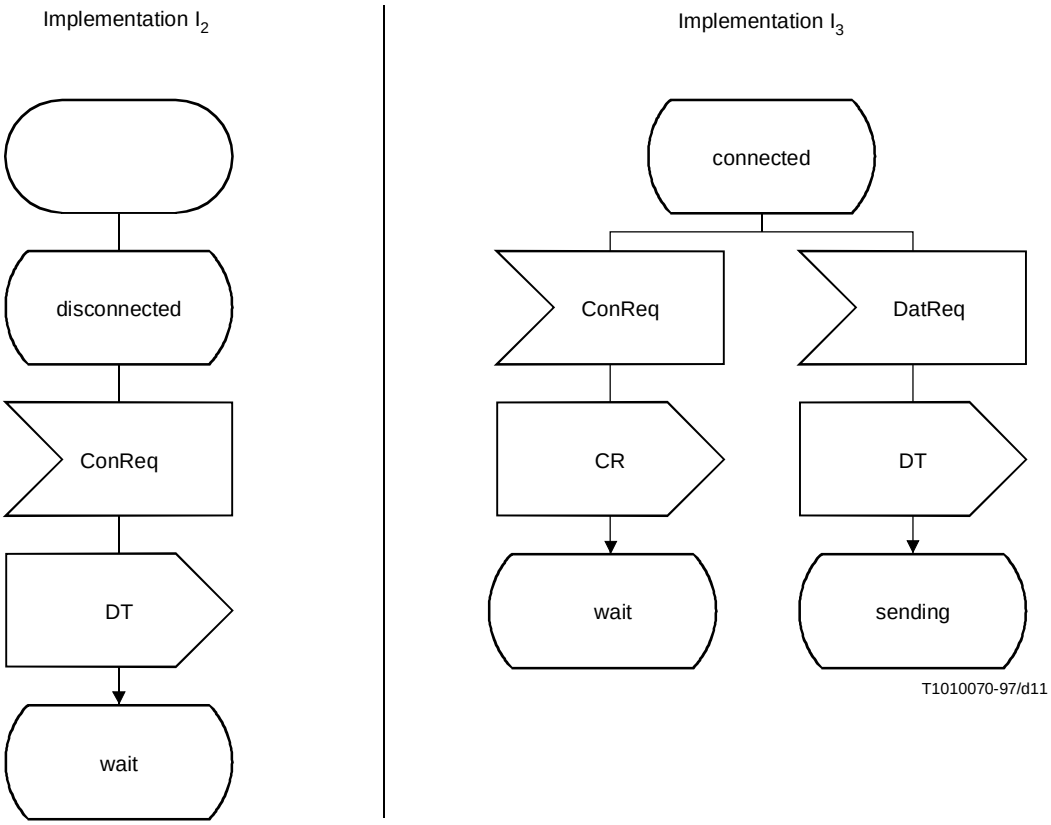

Figure A.9/Z.500 – Le comportement modifié des implémentations I_2 et I_3

A.4 Conformité par relations d'implémentation

A.4.1 Introduction

Une implémentation est conforme à sa spécification si le modèle de l'implémentation est lié à la spécification par une certaine relation d'implémentation imp. La relation imp $\subseteq$ *MODS* $\times$ *SPECS* constitue, en tant que tel, la notion de validité d'une implémentation par rapport à la spécification.

A.4.2 Conformité par relations d'implémentations dans Estelle

Comme indiqué ci-dessus, la définition des relations d'implémentation est fondée sur le modèle IOSM plutôt que sur une description syntactique Estelle. Des relations d'implémentation bien acceptées pour des automates à entrées/sorties comprennent des équivalences de trace ou des inclusions ou une variante de l'équivalence de trace qui est adaptée lorsqu'il s'agit d'automates dont la spécification est incomplète, désignée par le terme quasi-équivalence (définie sur PNFSM). Lorsque la quasi-équivalence est adaptée au modèle IOSM, cette relation devient la suivante [les sorties autorisées après une trace donnée dans σ IOSM S sont représentées par $O(\sigma,S)$]:

$$R_5(I,S) \text{ iff } (\forall\ \sigma \in Tr(S))\ (\sigma \in Tr(I) \Rightarrow (O(\sigma,I) = O(\sigma,S)))$$

Nous choisissons de prendre cette relation $R_5(I,S)$ comme exemple de relation imp. Selon cette relation d'implémentation:

– l'implémentation 1 de la Figure A.7 est conforme, ce qui n'est pas surprenant puisque cette implémentation est égale à la spécification;

– l'implémentation 2 de la Figure A.7 n'est pas conforme: il y a un bogue dans le programme du protocole et une unité de données de protocole CC est transmise au lieu d'une unité de données de protocole CR lorsqu'une demande *ConReq* est reçue par l'entité de protocole;

– l'implémentation 3 de la Figure A.7 est conforme. Ceci peut être surprenant étant donné qu'elle a un comportement étrange: si une demande *DatReq* est reçue par l'entité de protocole avant que la liaison ne soit établie, la connexion est rompue (ceci a le même effet qu'une demande de déconnexion). La conformité peut être expliquée de la manière suivante: ce comportement appartient à la partie non spécifiée du protocole et la relation d'implémentation choisie indique que l'implémentation est libre de faire n'importe quoi dans les parties non spécifiées de la spécification. Cependant, une relation d'implémentation différente peut être choisie, avec une interprétation différente des parties non spécifiées.

A.4.3 Conformité par relations d'implémentation dans LOTOS

Lorsque des spécifications sont décrites dans le langage LOTOS, une relation d'implémentation est une relation entre éléments de l'ensemble *LTS* (l'ensemble est choisi pour instancier *MODS*) et des éléments de l'ensemble *LTS* (l'ensemble choisi pour instancier *SPECS*). De nombreuses relations d'implémentation donnant une compréhension intuitive de la validité ont été proposées. Ces propositions ne sont pas discutées ici. L'utilisation des relations d'implémentation dans LOTOS sera illustrée en prenant un exemple particulier d'une telle relation.

Une implémentation bien acceptée pour les spécifications LOTOS est la relation de *conformité* désignée par conf. La relation conf utilise l'idée selon laquelle une implémentation *I* est correcte pour une spécification *S* si et seulement si l'implémentation *I* ne comporte pas de blocages qui n'ont pas été spécifiés dans *S*.

Une définition de la relation conf est donnée ci-dessous, où $Tr(S)$ représente l'ensemble des traces de *S*, *L* représente la globalité des actions observables, *a* représente une action observable et $\overset{\sigma}{\Rightarrow}$ représente la séquence σ d'actions observables:

$$
\begin{aligned}
I \ \underline{\text{conf}} \ S \ = \text{def} \quad & \forall \sigma \in Tr(S), \forall A \subseteq L : \\
& \text{si } \exists I' : I \overset{\sigma}{\Rightarrow} I' \text{ et } \forall a \in A : I' \overset{a}{\not\Rightarrow} \\
& \text{puis } \exists S' : S \overset{\sigma}{\Rightarrow} S' \text{ et } \forall a \in A : S' \overset{a}{\not\Rightarrow}
\end{aligned}
$$

Si la relation conf est choisie comme étant la relation d'implémentation (c'est-à-dire comme étant la notion de validité), on peut alors vérifier les modèles donnés à la Figure A.8 pour connaître ceux qui sont des implémentations correctes du modèle de spécification de la Figure A.5:

– le modèle **a)** de la Figure A.8 est un modèle correct (de l'implémentation) de la spécification visée par l'implémentation conf. Le modèle est notamment égal à la spécification pour tout ce qui concerne ses aspects comportementaux. Par conséquent, toute implémentation conforme à ce modèle est une implémentation correcte de la spécification décrite à la Figure A.5 en vertu de la relation d'implémentation conf;

– toute implémentation qui satisfait au modèle **b)** de la Figure A.8 n'est pas conforme à la spécification de la Figure A.5. Ceci est dû au fait que la spécification est capable d'exécuter la séquence d'actions $ConReq \cdot CR$, tandis que l'implémentation ne peut l'exécuter. Etant donné que la relation conf ne permet pas à cette dernière d'être un modèle correct d'implémentation, l'implémentation elle-même n'est pas conforme;

– une implémentation qui satisfait au modèle **c)** de la Figure A.8 n'est pas conforme à la spécification de la Figure A.5. Même si l'implémentation peut exécuter la séquence:

$$ConReq \cdot CR \cdot CC \cdot ConConf \cdot DatReq \cdot DT \cdot Dis \cdot Dis$$

et que la spécification ne le peut pas, aucun blocage non spécifié n'est introduit dans l'implémentation. Ainsi, on peut conclure, en vertu de la relation d'implémentation conf, que toutes les implémentations de modèle **c)** de la Figure A.8 sont des implémentations correctes de la spécification.

A.4.4 Conformité par relations d'implémentation dans SDL

Une *relation d'implémentation* définit un critère de conformité d'une implémentation donnée. Une implémentation est une implémentation conforme lorsque le groupe constitué par les modèles de l'implémentation et la spécification SDL fait partie de la relation d'implémentation.

Plusieurs des relations d'implémentation définies sont fondées sur l'existence de séquences d'événements similaires dans l'implémentation et la spécification. Les séquences d'événements observables peuvent être tirées des représentations ACT et LTS d'une spécification SDL. Une séquence d'événements observables est appelée *trace* et un *ensemble de traces Tr*(*S*) représente l'ensemble de toutes les traces possibles d'une spécification *S*. Une trace de la représentation ACT, illustrée à la Figure A.6, est ⟨*ConReq·CR·CC·ConConf*⟩.

Seule une catégorie très limitée de spécifications SDL spécifie un comportement qui est exprimé par un ensemble fini de traces. Ceci est dû au fait que la file d'attente des canaux SDL et les files d'attente d'entrées des procédés sont illimitées. Même pour des systèmes simples spécifiés dans SDL, il serait possible d'envoyer un nombre quelconque de signaux acheminés par le canal à partir de l'environnement du système. Ainsi, dans la spécification prise pour exemple, il serait possible d'envoyer un nombre quelconque de signaux *ConReq* au système. Par conséquent, dans la pratique, il n'est pas possible de réaliser un test exhaustif du comportement observable d'un système SDL. Néanmoins, les modèles de trace sont utiles pour définir une condition formelle de conformité du comportement dynamique d'une implémentation.

Une relation souvent utilisée comme relation d'implémentation est la relation d'inclusion de trace $\leq_{tr}$ (préclassement de trace). Deux modèles S_1 et S_2 satisfont la relation $S_1 \leq_{tr} S_2$ si et seulement si l'ensemble de traces de S_1 est un sous-ensemble de l'ensemble de traces de S_2.

La relation d'inclusion de trace peut être utilisée comme une relation d'implémentation de deux manières qui dépendent de la façon dont les conditions de la spécification sont interprétées. Si la spécification est supposée spécifiée le *comportement maximal autorisé* d'une implémentation, l'ensemble de traces d'une implémentation conforme est un sous-ensemble de l'ensemble de traces de la spécification. Pour une implémentation conforme *I* et une spécification *S*, ceci est représenté par $I \leq_{tr} S$ ou $(I,S) \in \leq_{tr}$.

Une autre interprétation des conditions prescrites par une spécification SDL est qu'elle définit le *comportement minimal exigé* d'une implémentation. Dans ce cas, l'ensemble de traces d'une spécification *S* est un sous-ensemble d'une implémentation conforme *I*, $I \geq_{tr} S$.

La relation d'implémentation de comportement maximal autorisé implique une très faible condition sur une implémentation conforme. Une implémentation qui ne peut exécuter aucune action externe est une implémentation conforme de toute spécification, étant donné que l'ensemble vide est un sous-ensemble de tout ensemble trace. Il n'est pas possible de tester la relation d'implémentation de comportement minimal exigé du fait que la plupart des spécifications SDL ont des ensembles de traces infinis.

Le Tableau A.2 montre les spécifications qui sont conformes par rapport à l'exemple indiqué (désignées ici par *S*) et aux deux relations d'implémentation.

Tableau A.2/Z.500 – Présentation générale de la conformité de I_1, I_2, et I_3 par rapport à différentes relations d'implémentation

(Impl, Spéc) satisfait	$\leq_{tr}$	$\geq_{tr}$
(I_1,S)	vrai	vrai
(I_2,S)	faux	faux
(I_3,S)	faux	vrai

Etant donné que l'ensemble de traces de l'implémentation I_1 est identique à celui de la spécification, les deux relations, comportement maximal autorisé et comportement minimal exigé, sont satisfaites pour I_1. L'implémentation I_2 ne satisfait pas à la relation $\leq_{tr}$ étant donné que la trace ⟨*ConReq·DT*⟩ ne fait pas partie de l'ensemble de traces de la spécification. De la même manière, la relation d'implémentation $\geq_{tr}$ n'est pas non plus satisfaite étant donné que l'ensemble de traces de I_2 n'inclut pas la trace ⟨*ConReq·CR*⟩. Pour l'implémentation I_3 la relation d'implémentation $\leq_{tr}$ n'est pas satisfaite étant donné que l'implémentation peut, par exemple, exécuter la trace ⟨*ConReq·CR·CC·ConConf·ConReq·CR*⟩ qui n'est pas une trace de la spécification. Etant donné que I_3 peut exécuter toute trace de la spécification, elle est conforme en vertu de la relation d'implémentation $\geq_{tr}$.

Le modèle d'une implémentation n'est pas connu à l'avance pour les tests de conformité. On ne peut qu'en faire des approximations en exécutant des expérimentations sur l'implémentation et en en observant les réactions. Des spécifications de système non déterministes sont telles qu'il est impossible de s'assurer qu'un modèle d'implémentation est une description complète du comportement possible.

Dans la spécification fournie pour exemple, il est admis qu'il ne soit pas possible de déterminer si une implémentation peut exécuter une trace spécifique. Ceci est le cas pour la trace ⟨*ConReq·CR*⟩. Le canal non fiable peut toujours ignorer le signal *CR* de sorte qu'il n'apparaisse jamais comme un événement observable dans l'environnement. Cependant, il n'est pas possible de déduire, à partir d'un certain nombre d'expérimentations dans lesquelles le signal *CR* n'a pas été observé, si la trace a été mise en œuvre. Ainsi, les relations d'implémentation préclassement de trace ne peuvent fournir une base saine en tant que critère de conformité que si l'hypothèse supplémentaire est posée sur la spécification et l'implémentation. Par exemple, il est admis de supposer qu'il y a fourniture d'informations sur la façon dont le non-déterminisme de la spécification est résolu dans l'implémentation.

Une autre méthode consiste à définir des relations d'implémentation qui tiennent compte des limites des tests de conformité réels des systèmes spécifiés dans SDL. Les relations **asco** et **aconf** sont deux des relations d'implémentation proposées dans [TrVe 92] pour être utilisées dans des systèmes SDL qui communiquent avec l'environnement par l'intermédiaire d'un canal unique.

Pour satisfaire ces relations d'implémentation, un seul sous-ensemble de l'ensemble de traces est pris en compte. Le sous-ensemble comprend uniquement des traces qui sont minimales par rapport à l'ajout de signaux à un canal et le reclassement des événements extérieurs du fait de retard sur les canaux. Par exemple, la trace ⟨*ConReq·CR*⟩ est une trace minimale sur la base de laquelle l'implémentation doit être testée. Des exemples de traces pour lesquels cette trace est minimale sont ⟨*ConReq·CR·ConReq*⟩ lorsqu'un signal d'entrée est annexé et ⟨*ConReq·ConReq·CR*⟩ lorsque le signal d'entrée *ConReq* est déplacé en face du signal de sortie *CR*.

La relation **asco** est satisfaite pour chaque trace minimale, la dernière action étant un signal de sortie, l'implémentation ne peut exécuter qu'un sous-ensemble des actions possibles conformément à la spécification. Pour **aconf**, ceci doit également être vrai pour toutes les traces préfixes des traces minimales. Le Tableau A.3 illustre la manière dont les deux relations font la distinction entre les trois implémentations.

Tableau A.3/Z.500 – Conformité de I_1, I_2, et I_3 par rapport aux relations d'implémentation asco et aconf

(Impl, Spéc) satisfait	asco	aconf
(I_1,*S*)	vrai	vrai
(I_2,*S*)	faux	faux
(I_3,*S*)	vrai	vrai

Pour l'implémentation I_1, les deux relations d'implémentation sont satisfaites étant donné que pour chaque trace le même ensemble d'événements peut être observé pour la spécification et l'implémentation. Pour l'implémentation I_2, la trace minimale ⟨*ConReq·CR*⟩ dans laquelle l'ensemble d'événements après ⟨*ConReq*⟩ pour l'implémentation est {*deadlock*,*DT*}, tandis que l'ensemble d'événements de la spécification est {*deadlock*,*CR*}. Etant donné que l'ensemble d'événements d'implémentation n'est pas un sous-ensemble de la spécification, I_2 est une implémentation non conforme pour les deux relations. Finalement, l'implémentation étant testée uniquement pour ce qui concerne les traces minimales de la spécification, le type de comportement supplémentaire, comme dans l'implémentation I_3, n'est pas pris en compte et I_3 est une implémentation conforme.

A.5 Conformité par exigences

La conformité entre une implémentation et une spécification peut également être caractérisée au moyen de conditions. Pour l'expression de ces conditions, un langage spécifique *REQS* est utilisé. Une implémentation est conforme à sa spécification si les propriétés dans les langages *REQS* qui correspondent à la spécification sont satisfaites par l'implémentation.

Par exemple, le langage *REQS* peut être instancié par des langages logiques, par exemple, Hennessy-Milner Logic, CTL. Le fait de vérifier la conformité d'un modèle d'implémentation à une certaine condition est appelé contrôle du modèle. Le contrôle du modèle ne peut être effectué que si le modèle d'une implémentation est explicitement disponible.

Un autre exemple est utilisé ci-dessous en introduisant un langage de propriété simple. Le langage de propriété peut être utilisé pour exprimer des propriétés sur des systèmes de transition étiquetés.

Pour $\sigma \in L^*$ et $A \subseteq L$ l'ensemble *REQS* est instancié par REQS = { **after** σ **from** A | σ ∈ L* and $A \subseteq L$ }. Un système de transition étiqueté T satisfait la propriété **after** σ **from** A si $\exists T' : T \stackrel{\sigma}{\Rightarrow} T'$ et $\forall a \in A : T' \stackrel{a}{\Rightarrow}$.

Dans la Figure A.8, les modèles **a)** et **c)** satisfont la propriété **after** *ConReq* **from** {*CR*}, mais le modèle **b)** ne satisfait pas à cette propriété.

A.6 Architecture de test

A.6.1 Introduction

L'architecture de test est une description de l'environnement dans lequel l'instance IUT est testée. Il est nécessaire de modéliser l'environnement de l'instance IUT car le comportement de l'IUT ne peut être directement observable par le testeur du fait des éventuelles limites d'observabilité imposées, par exemple, par un support de communication intermédiaire entre le testeur et l'instance IUT.

Chaque technique FDT dispose de fonctionnalités de modélisation de l'environnement d'une IUT.

A.6.2 Architecture de test dans Estelle

L'architecture de test est décrite dans Estelle au moyen des éléments de construction suivants:

– des éléments de construction *module* sont utilisés pour décrire les composantes (testeur, IUT, contexte de test);

– des éléments de construction *body* décrivent le comportement des diverses composantes;

– des *interaction points* (points d'interaction) représentent les points PCO et IAP.

Etant donné que les modules Estelle sont liés par des canaux qui ont une sémantique de file d'attente FIFO, ces choix de modélisation sont compatibles avec la sémantique PCO telle que décrite par l'ISO 9646 (et qui correspond à la sémantique PCO de la notation TTCN).

Cette approche fournit une modélisation explicite du contexte de test comme une composante en soi. Il y a eu quelques tentatives pour représenter le contexte de test d'une manière plus proche que celle décrite dans le texte principal de la présente Recommandation, c'est-à-dire en tant que fonction de transformation sur l'ensemble *MODS* (dans notre cas: sur l'ensemble de IOSM): voir [Phalippou 92] par exemple.

A.6.3 Architecture de test dans LOTOS

LOTOS ne fournit pas d'éléments de construction *spéciaux* pour la modélisation du comportement des environnements système. Par contre, l'environnement est spécifié comme n'importe quel autre processus. Il n'est fait aucune distinction entre des environnements de modélisation et des systèmes de modélisation au sein des environnements.

Etant donné que dans LOTOS, la communication est synchrone, la communication asynchrone est modélisée par modélisation explicite du support de communication intermédiaire (par exemple, files d'attente FIFO). L'exemple ci-dessous fournit une spécification LOTOS de base communiquant par l'intermédiaire d'une file d'attente. Dans cet exemple, toutes les entrées du système qui arrivent au niveau d'une porte a doivent passer par une file d'attente fiable. La communication entre la file d'attente et le système est cachée. Il est à noter que cette file d'attente n'est pas du type FIFO.

```
specification QueueCommunication[a, b, x, y] : noexit
behaviour
  hide ia in System [ia, b, x, y] |[ia]| Queue [ia, a]
where
  process System [a, b, x, y] : noexit:=
  a ;  x ;  stop [] b ;  (x ;  stop [] y ;  stop )
  endproc (* System *)

  process Queue[ia, a] : noexit:=
  a ; (ia ; stop ||| Queue [ia, a])
  endproc (* Queue *)
endspec (* QueueCommunication *)
```

Dans LOTOS, le testeur, l'instance IUT et le contexte de test sont modélisés par des processus LOTOS. Les points PCO et IAP sont modélisés par des portes (par exemple, des éléments de construction dans LOTOS pour modéliser des points d'interaction).

A.6.4 Architecture de test dans SDL

La seule exigence d'une spécification SDL sur l'environnement d'un système est qu'il obéisse aux contraintes imposées par la spécification du système. Ainsi, pour modéliser les propriétés d'un environnement dans lequel une implémentation doit être testée, le contexte de test doit être spécifié dans le cadre de la spécification du système. Dans l'exemple commun, le support fiable (*Canal*) peut être vu comme un contexte de test de ce genre pour le protocole.

Les entités de l'architecture de test sont définies dans une spécification SDL en utilisant les mêmes éléments de construction que ceux de la spécification système originale. Le testeur, le contexte de test et l'instance IUT peuvent être spécifiés au moyen d'éléments de construction *bloc*. Le comportement des entités est défini par les *processus* des blocs. La communication entre les entités de l'architecture de test (blocs) est modélisée par des canaux. Etant donné que dans SDL, les canaux sont des files d'attente FIFO, ils peuvent être utilisés pour modéliser les points PCO conformément à la norme [ISO 9646]. Les canaux sont également utilisés pour spécifier des points IAP.

Dans l'exemple commun, les entités de l'architecture de test peuvent être identifiées comme suit dans le schéma du système de la Figure A.13. Le bloc *Protocol_M* spécifie l'instance IUT et le bloc *Medium_M* spécifie les propriétés du contexte de test. Les points PCO d'architecture de test sont les canaux *ISAP* et *LSAP* tandis que les points IAP sont les canaux *ISAP* et *MSAP*. Dans cet exemple, l'instance IUT et le contexte de test sont tous deux spécifiés par un seul bloc, en général il est admis d'utiliser un ensemble de blocs pour spécifier l'une de ces entités.

A.7 Spécifications des tests

A.7.1 Introduction

Les tests sont des procédures utilisées pour vérifier la conformité des implémentations à la spécification. Les systèmes de test peuvent être vus comme des composantes réparties qui interagissent avec les implémentations. Etant donné qu'ils sont tout simplement des types particuliers de systèmes répartis, les tests peuvent être décrits au moyen des techniques FDT normalisées Estelle, LOTOS et SDL. Cet aspect est analysé aux A.7.2, A.7.3 et A.7.4.

Cependant, le cadre des tests de conformité de [ISO 9646] utilise certains concepts particuliers (tels que l'architecture de test, les points PCO, la structure de suite de tests, les verdicts) qui ne sont pas des éléments de construction intégrés des techniques FDT normalisées Estelle, LOTOS et SDL. Par contre, le langage TTCN a été spécifiquement conçu pour décrire formellement des tests élémentaires. Il est analysé au A.7.5.

A.7.2 Spécifications des tests dans Estelle

Estelle a été conçu comme un langage de spécification et non comme un langage de description de test. Par conséquent, contrairement au langage TTCN, Estelle ne dispose pas d'éléments de construction intégrés représentant les concepts de tests qui apparaissent dans le texte principal de la présente Recommandation (test élémentaire, suite de tests, événement de test, verdicts). Estelle peut, bien entendu, être perçu comme un langage de programmation et il est possible d'y décrire la partie comportement des tests élémentaires (qui, dans un cadre entrée/sortie, correspond aux entrées et sorties des messages). En outre, les concepts de test mentionnés ci-dessus peuvent être modélisés dans Estelle en repérant quelques choix particuliers (par exemple, l'utilisation d'une variable pour coder le verdict, l'utilisation d'un corps pour représenter le comportement d'un test élémentaire, etc.). Mais il ne s'agit là que de choix particuliers d'une modélisation compliquée. En conséquence, Estelle ne sera pas utilisé dans la présente annexe comme un langage de description de test. Lorsque des spécifications sont décrites dans Estelle, le langage de description de test utilisé sera la notation TTCN. Etant donné que les deux langages utilisent un mécanisme de communication d'entrée/sortie par file d'attente FIFO, leur corrélation pour l'exécution des tests ne pose pas de problème (voir A.8).

A.7.3 Spécifications des tests dans LOTOS

LOTOS peut lui-même être utilisé pour la spécification de tests [Bri 88, Tre 92]. Un test élémentaire peut être modélisé par un processus LOTOS lorsque chaque état est étiqueté par l'un de ces éléments {**succès**, **échec**, **non concluant**}. Un exemple d'un tel test est fourni à la Figure A.10.

Il n'est pas facile de modéliser des verdicts dans LOTOS. Une possibilité consiste à utiliser l'étiquette spéciale σ (terminaison réussie) pour réussir un verdict.

De la même manière qu'Estelle et SDL, LOTOS n'est pas conçu pour décrire des tests élémentaires. Le langage TTCN a été conçu pour l'expression des tests. Cependant, les expressions TTCN représentent des tests élémentaires qui sont par nature asynchrones. La communication dans LOTOS étant synchrone, ceci peut poser un problème lorsque les tests élémentaires sont décrits par des expressions TTCN.

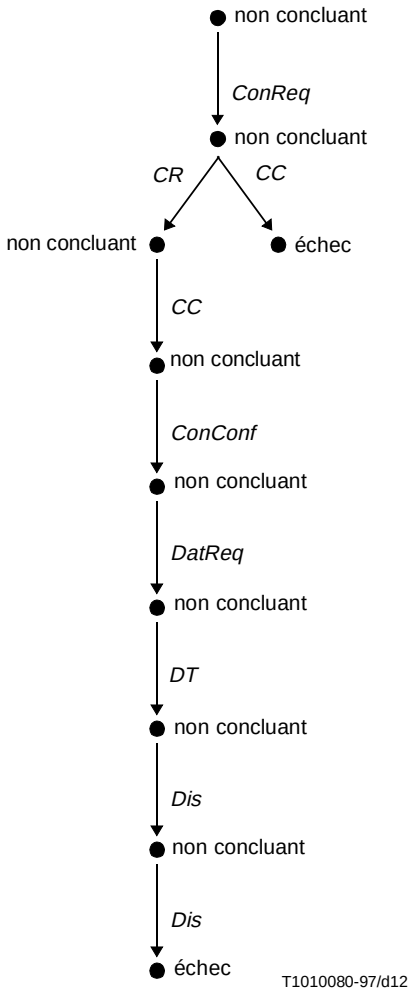

Figure A.10/Z.500 – Représentation d'un test élémentaire comme système de transition étiqueté

A.7.4 Spécifications des tests dans SDL

SDL est une technique de spécification polyvalente pour la modélisation des systèmes de communication. Elle peut également être utilisée pour la spécification des tests de conformité élémentaires. Cependant, le langage n'est pas spécifiquement conçu pour ce domaine d'application. En conséquence, l'utilisation de SDL pour la spécification de test élémentaire telle que prescrite dans la norme [ISO 9646], exige que les concepts de la présente Recommandation soient définis en termes d'éléments de construction SDL.

La plupart des concepts de la norme [ISO 9646] peuvent être modélisés directement dans SDL. Les concepts des points PCO et des échanges de messages peuvent être modélisés par des canaux et des instances de signaux. Un test élémentaire peut être modélisé comme une procédure SDL et une suite de tests comme un processus. Ainsi, l'ordre d'exécution des tests élémentaires est défini dans la définition du processus. Les attributions de verdict peuvent être modélisées par des variables associées à chaque test élémentaire. Les attributions de verdict qui résultent des exécutions de test élémentaire peuvent être transmises à l'environnement par le biais de signaux. Cette approche est définie dans [R1072]. Etant donné que la notation ASN.1 peut être utilisée en association au langage SDL pour les définitions de données [Z105], les règles de données contenues dans la norme [ISO 9646] peuvent être directement utilisées.

Quelques concepts, définis dans la méthodologie de test de la norme [ISO 9646] ne peuvent être spécifiés en langage SDL. Ceci est le cas des conditions en temps réel et du traitement des messages imprévus. Dans SDL, les conditions en temps réel ne peuvent être modélisées étant donné que la sémantique SDL définit uniquement un modèle discret pour le temps et rien n'est déclaré pour ce qui concerne le temps nécessaire à l'exécution d'une transition. Dans une spécification SDL, seuls des signaux déclarés ont une signification. Ainsi le traitement des messages imprévus qui, dans la notation TTCN est pris en charge par l'élément de construction *otherwise* (autrement), ne peut être modélisé que par déclaration explicite des signaux pour représenter ces messages.

A.7.5 Spécification des tests dans TTCN

Le langage normalisé TTCN, *tree and tabular combined notation* (notation combinée arborescente et tabulaire) a été conçu pour la représentation et la spécification de tests élémentaires de conformité qui peuvent être exprimés de manière abstraite en termes de contrôle et d'observation des unités de données de protocole et de primitives de service abstrait. Le langage TTCN est disponible sous deux formes:

– un format graphique lisible par l'homme;

– un format exploitable par machine qui convient à la transmission de descriptions TTCN entre machines.

Les tests élémentaires décrits par des expressions TTCN représentent des tests élémentaires *abstraits* qui sont compilés dans des tests élémentaires exécutables pour être lancés sur la machine de test physique.

TTCN est un langage de niveau élevé de spécification des tests élémentaires. Cependant, comme dans le cas des spécifications (voir A.1), il est plus facile de définir les objets formels sur un modèle plus basique. TTCN disposant d'une sémantique d'entrées/sorties, nous utilisons *IOSM* comme modèle sémantique des tests élémentaires TTCN.

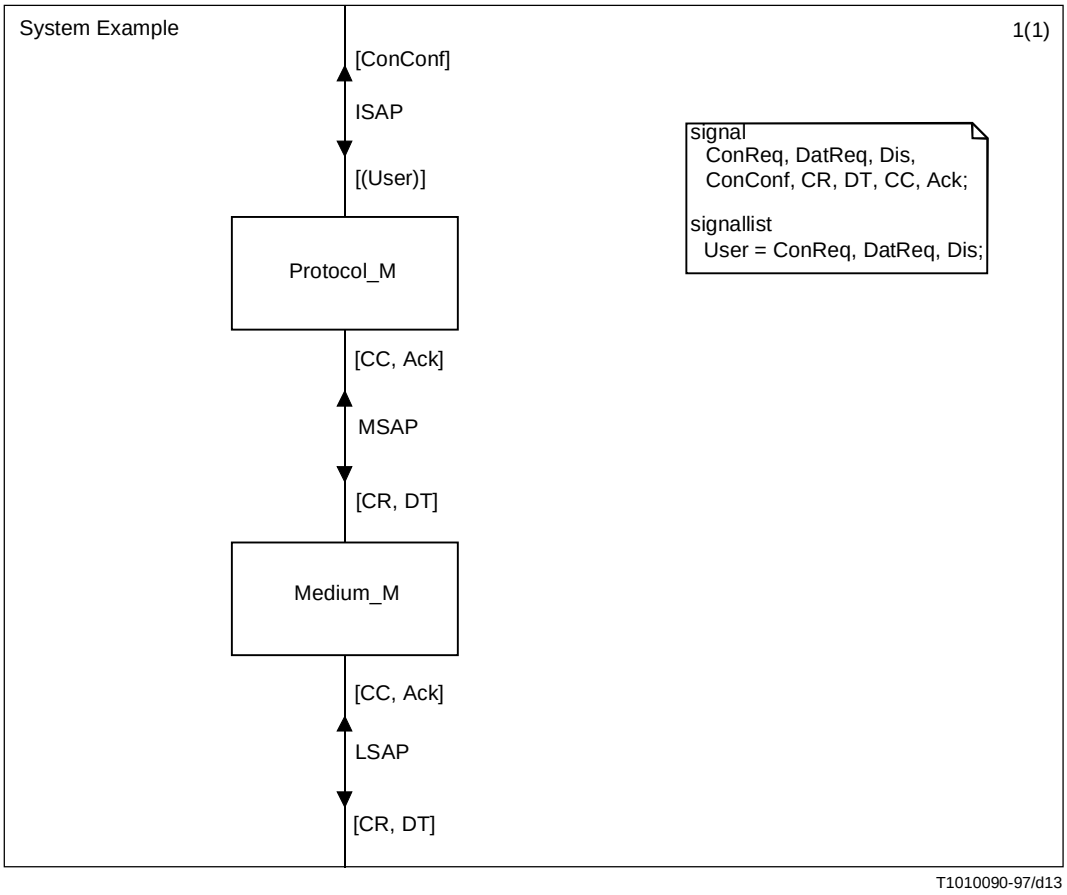

Figure A.11/Z.500 – Schéma système du protocole

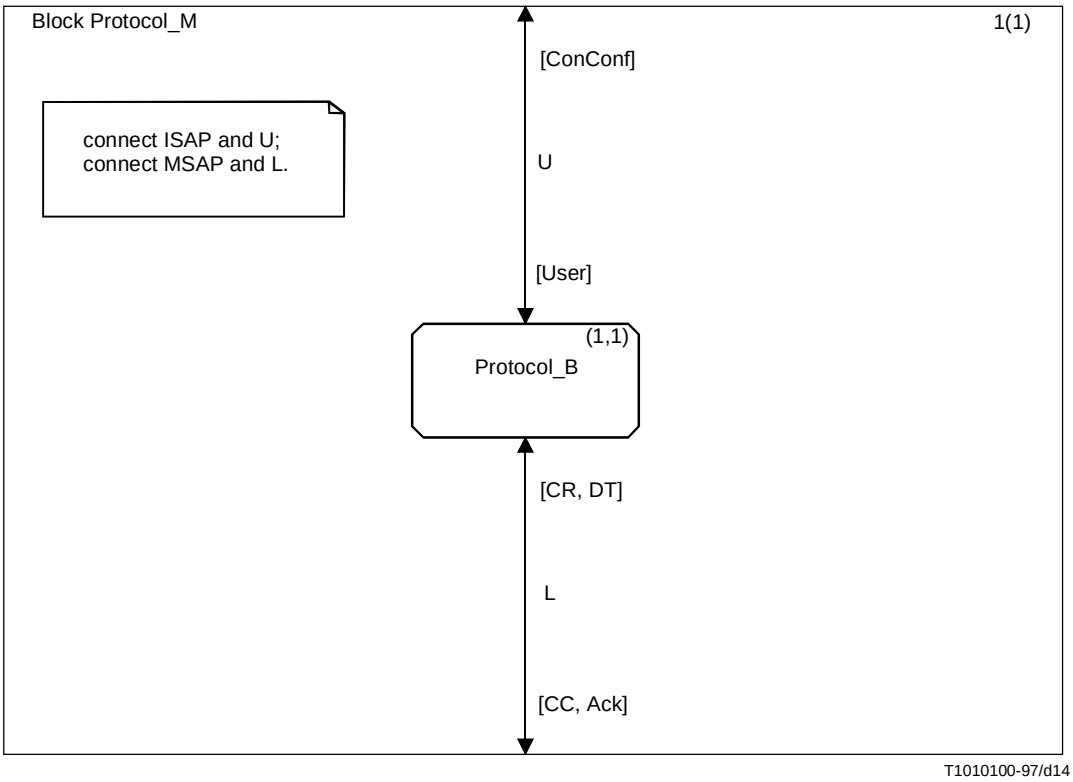

Figure A.12/Z.500 – Bloc du processus d'émission

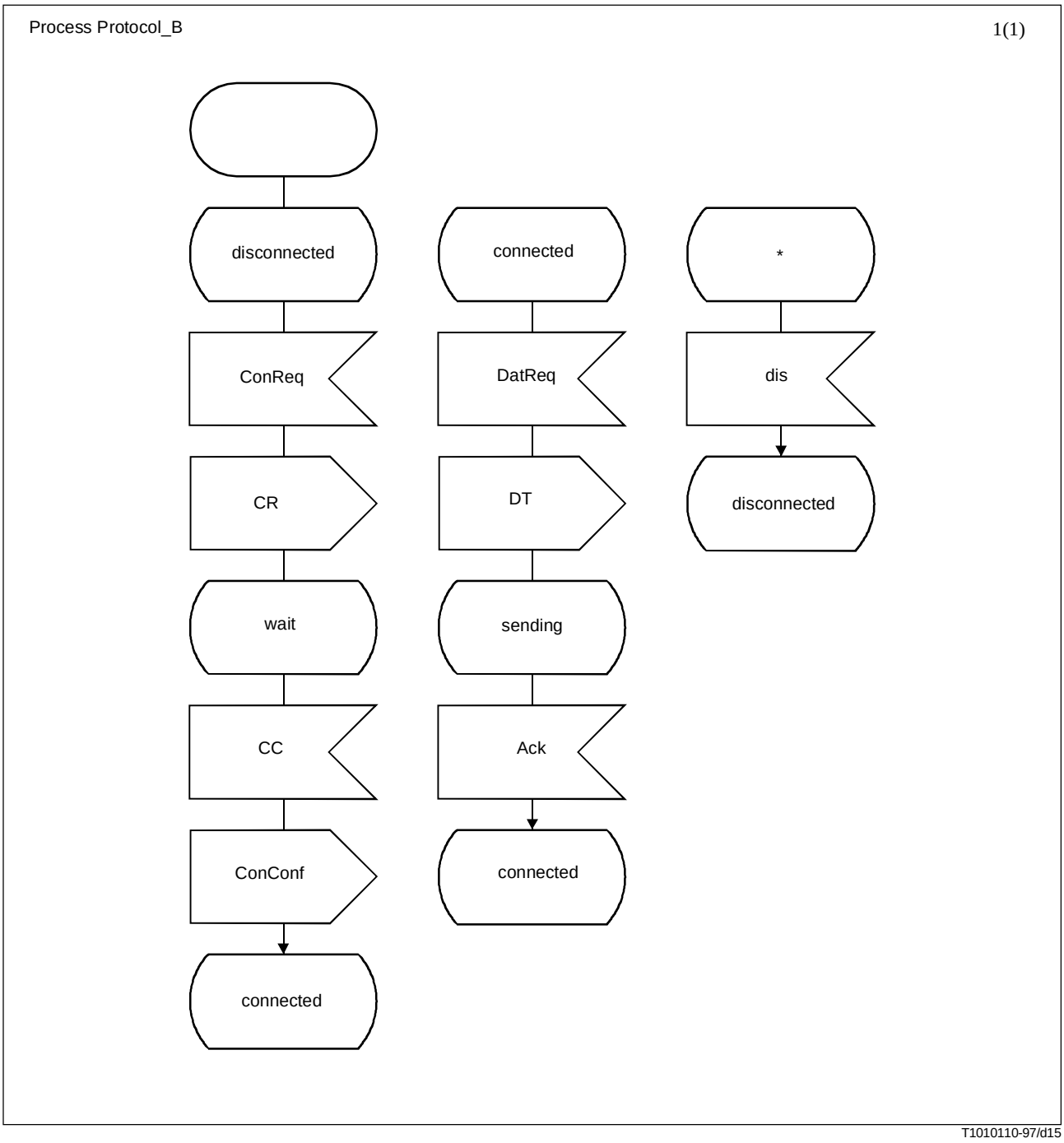

Figure A.13/Z.500 – Processus d'émission

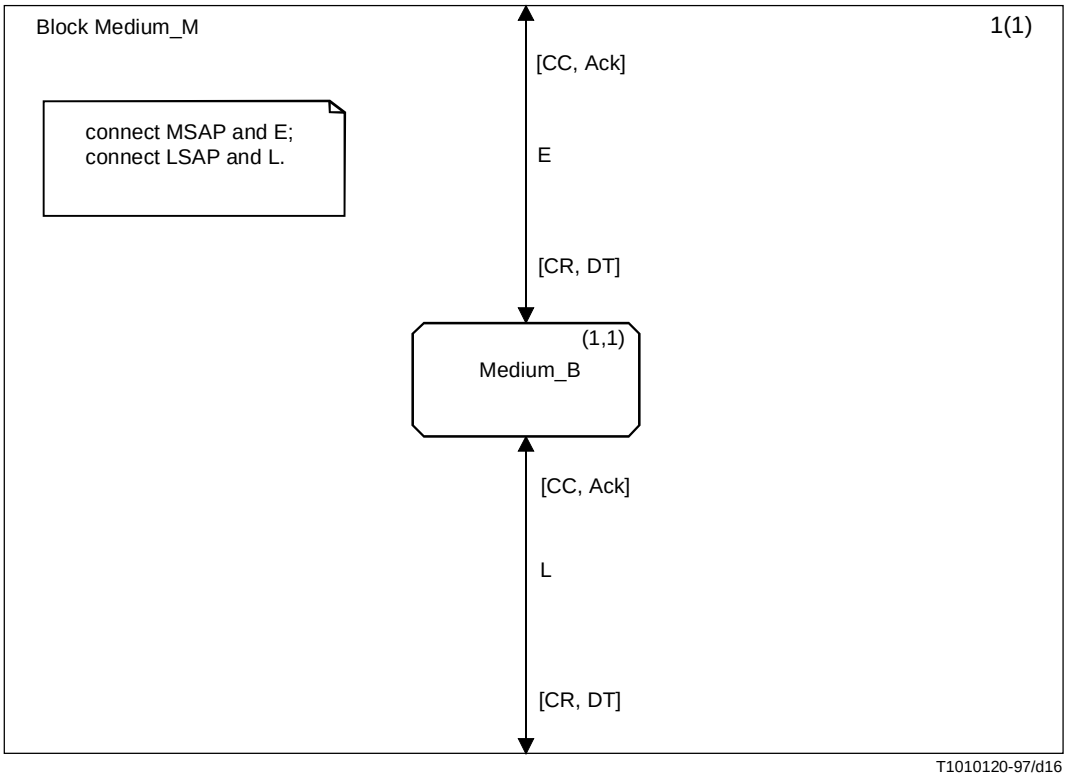

Figure A.14/Z.500 – Bloc de modélisation du support non fiable

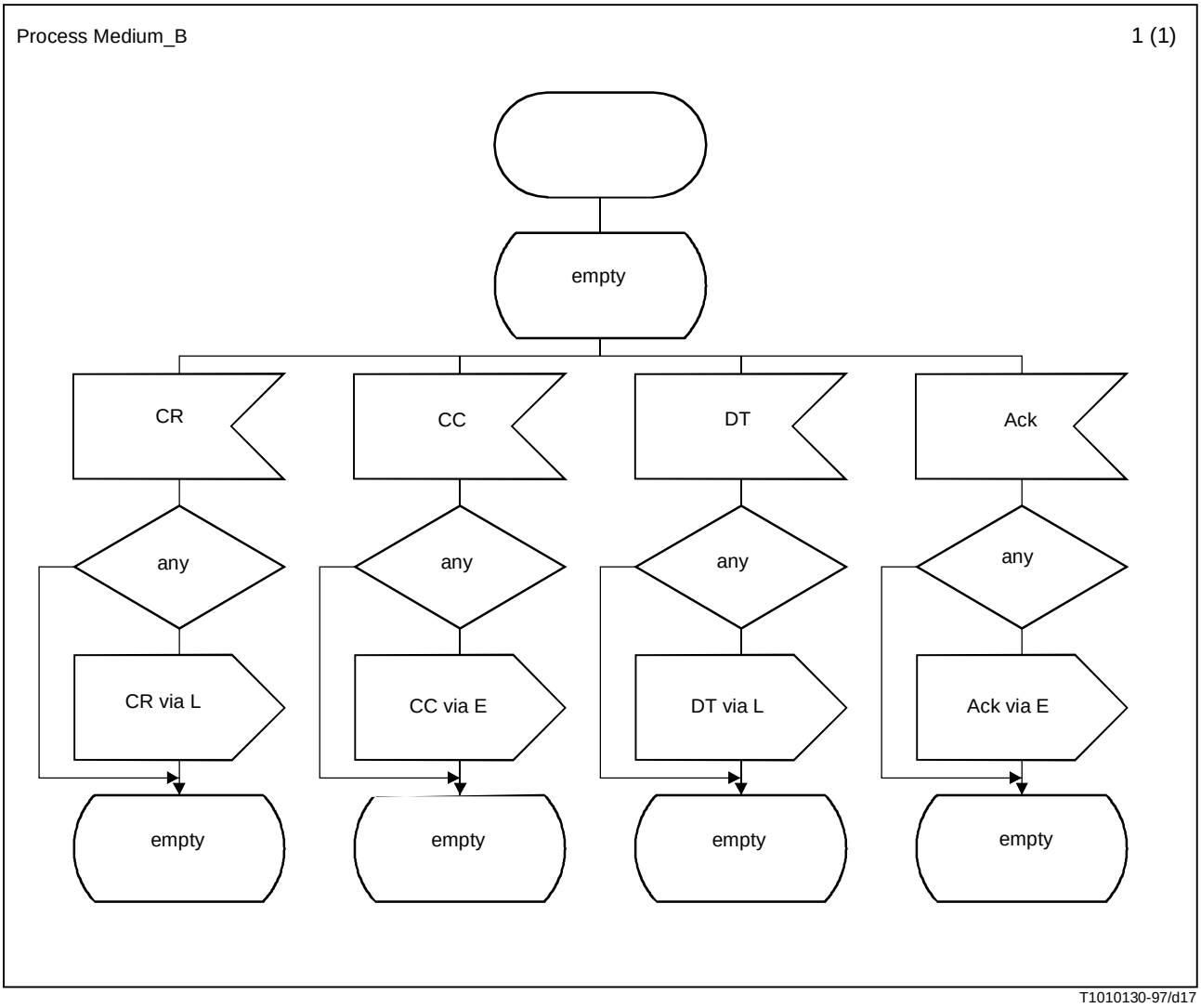

Figure A.15/Z.500 – Comportement d'un support non fiable

A.8 Références

[Bri 88] BRINKSMA (E.): A theory for the derivation of tests, *Protocol Specification*, Testing and Verification VIII, p. 63-74, North Holland, 1987.

[Hogrefe 88] HOGREFE (D.): Automatic generation of test cases from SDL specifications, *SDL Newsletter*, N° 12, 1988.

[Hopcroft 79] HOPCROFT (J.) et ULLMAN (J.): Introduction to automata theory, languages and computation, Addison-Wesley publishing company, 1979.

[ISO 8073] *Technologies de l'information – Télécommunications et échange d'informations entre systèmes – Interconnexion de systèmes ouverts* (*OSI*) *– Protocole pour fourniture du service de transport en mode connexion*, Norme Internationale IS-8073, 3e édition, 1992.

[ISO 8807] *Systèmes de traitement de l'information – Interconnexion de systèmes ouverts – LOTOS – Technique de description formelle basée sur l'organisation temporelle de comportement observationnel*, Norme Internationale IS-8807, ISO, 1989.

[ISO 9646] *Technologies de l'information – Interconnexion de systèmes ouverts – Cadre général et méthodologie des tests de conformité OSI*, Norme Internationale IS-9646, ISO.

[Luo 93] LUO (G.), PETRENKO (A.) et BOCHMANN (G.V.): Selecting test sequences for partially-specified nondeterministic finite state machines, Publication N° 864, Université de Montréal, février 1993.

[Phalippou 92] PHALIPPOU (M.): The limited power of testing, *Proceedings of the 5th International Workshop on Protocol Test Systems*, Montréal, septembre 1992.

[Phalippou 94] PHALIPPOU (M.): Relations d'implantation et hypothèses de test sur des automates à entrées et sorties, *Thèse de l'Université de Bordeaux I*, septembre 1994.

[R1072] ITACA, IBCN Testing Architecture for Conformance Assessment, R1072 ITACA – N° 365, 1992.

[Tret 92] TRETMANS (J.): A Formal Approach to conformance Testing, *PhD thesis*, University of Twente, 1992.

[TrVe 92] TRETMANS (J.) et VERHAARD (L.): A queue model relating synchronous and asynchronous communication, *Memorandum INF-92-04*, University of Twente, Enschede, The Netherlands. TFL RR 1992-1, *Tele Danmark Research*, 1992.

[Z105] Recommandation UIT-T Z.105, *Langage de description et de spécification combiné avec la notation de syntaxe abstraite numéro un*, UIT, 1995.

SÉRIES DES RECOMMANDATIONS UIT-T

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