Union internationale des télécommunications

UIT-T

SECTEUR DE LA NORMALISATION
DES TÉLÉCOMMUNICATIONS
DE L'UIT

Z.144

(03/2006)

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

Techniques de description formelle – Notation de test et de commande de test

Notation de test et de commande de test version 3 (TTCN-3): interface d'exécution

Recommandation UIT-T Z.144

RECOMMANDATIONS UIT-T DE LA SÉRIE Z

LANGAGES ET ASPECTS GÉNÉRAUX LOGICIELS DES SYSTÈMES DE TÉLÉCOMMUNICATION

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

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

Recommandation UIT-T Z.144

Notation de test et de commande de test version 3 (TTCN-3): interface d'exécution

Résumé

Le présente Recommandation contient la spécification de l'interface d'exécution pour les implémentations d'un système de test de Notation de test et de commande de test 3 (TTCN-3, *testing and test control notation 3*). L'interface d'exécution TTCN-3 assure l'adaptation recommandée, pour la temporisation et la communication entre, d'une part, un système de test et, d'autre part, une plate-forme de traitement donnée et le système sous test, respectivement. La présente Recommandation définit l'interface comme étant un ensemble d'opérations indépendantes du langage cible.

L'interface est définie pour être compatible avec la Rec. UIT-T Z.140. La présente Recommandation utilise le langage de définition d'interface (IDL, *interface definition language*) CORBA pour spécifier l'interface TRI de façon complète. Les paragraphes 6 et 7 détaillent les mappages de la spécification abstraite vers les langages cibles Java et ANSI-C. La spécification de l'interface fondée sur le langage IDL est récapitulée dans l'Annexe A.

Source

La Recommandation UIT-T Z.144 a été approuvée le 16 mars 2006 par la Commission d'études 17 (2005-2008) de l'UIT-T selon la procédure définie dans la Recommandation UIT-T A.8.

AVANT-PROPOS

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

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

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

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

NOTE

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

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

DROITS DE PROPRIÉTÉ INTELLECTUELLE

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

A la date d'approbation de la présente Recommandation, l'UIT n'avait pas été avisée de l'existence d'une propriété intellectuelle protégée par des brevets à acquérir pour mettre en œuvre la présente Recommandation. Toutefois, comme il ne s'agit peut-être pas des renseignements les plus récents, il est vivement recommandé aux développeurs de consulter la base de données des brevets du TSB sous http://www.itu.int/ITU-T/ipr/.

TABLE DES MATIÈRES

Introduction

La présente Recommandation comprend deux parties distinctes. La première partie décrit la structure d'implémentation d'un système de test TTCN-3 et la seconde partie présente la spécification de l'interface d'exécution TTCN-3.

La première partie donne un aperçu des quatre entités principales dont se compose un système de test TTCN-3:

- entité de gestion de test (TM);
- exécutable TTCN-3 (TE);
- adaptateur du système SUT (SA);
- adaptateur de plate-forme (PA).

La première partie définit en outre les interactions entre ces entités, c'est-à-dire les interfaces correspondantes.

La seconde partie de la présente Recommandation définit l'interface d'exécution TTCN-3 (TRI), sur la base des opérations qui sont implémentées dans le cadre d'une entité et qui sont appelées par d'autres entités du système de test. Pour chaque opération, la spécification de l'interface définit les structures de données associées, l'effet recherché sur le système de test et les éventuelles contraintes relatives à l'utilisation de l'opération. Il est à noter que la présente spécification d'interface définit uniquement les interactions entre l'interface du système de test (TSI) et le système sous test (SUT) ainsi que les opérations de temporisation.

Recommandation UIT-T Z.144

Notation de test et de commande de test version 3 (TTCN-3): interface d'exécution

1 Domaine d'application

La présente Recommandation contient la spécification de l'interface d'exécution pour les implémentations d'un système de test TTCN-3. L'interface d'exécution TTCN-3 assure une adaptation normalisée, pour la temporisation et la communication entre, d'une part, un système de test et, d'autre part, une plate-forme de traitement donnée et le système sous test, respectivement. La présente Recommandation définit l'interface comme étant un ensemble d'opérations indépendantes du langage cible.

L'interface est définie pour être compatible avec la norme TTCN-3 (voir référence ci-dessous). La présente Recommandation utilise le langage de définition d'interface (IDL, *interface definition language*) CORBA pour spécifier l'interface TRI de façon complète. Les paragraphes 6 et 7 présentent les mappages de cette spécification abstraite vers les langages cibles Java et ANSI-C. La spécification de l'interface fondée sur le langage IDL est récapitulée dans l'Annexe A.

1.1 Conformité

Pour être conforme à l'interface TRI, un système de test TTCN-3 doit respecter la spécification d'interface énoncée dans la présente Recommandation ainsi qu'un des mappages vers les langages cibles qui y sont indiqués.

EXEMPLE: si un fournisseur prend en charge le langage Java, les appels d'opération TRI et les implémentations, qui font partie de l'exécutable TTCN-3, doivent être conformes au mappage IDL → Java spécifié dans la présente Recommandation.

2 Références normatives

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

[1] Recommandation UIT-T X.290 (1995), *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 – Concepts généraux.*

ISO/CEI 9646-1:1994, *Technologies de l'information – Interconnexion de systèmes ouverts (OSI) – Cadre général et méthodologie des tests de conformité – Partie 1: Concepts généraux.*

[2] Recommandation UIT-T Z.140 (2006), *Notation de test et de commande de test version 3 (TTCN-3): langage noyau*.

[3] Recommandation UIT-T X.292 (2002), *Cadre et méthodologie des tests de conformité OSI pour les Recommandations sur les protocoles pour les applications de l'UIT-T – Notation combinée arborescente et tabulaire (TTCN).*

ISO/CEI 9646-3:1998, *Technologies de l'information – Interconnexion de systèmes ouverts – Essais de conformité – Méthodologie générale et procédures – Partie 3: Notation combinée, arborescente et tabulaire (TTCN).*

[4] Recommandation UIT-T Z.143 (2006), *Notation de test et de commande de test version 3 (TTCN-3): sémantique opérationnelle*.

3 Définitions et abréviations

3.1 Définitions

Dans la présente Recommandation, les termes et définitions figurant dans la Rec. UIT-T Z.140 [2] s'appliquent en plus de ce qui suit:

3.1.1 suite de tests abstraits (ATS, *abstract test suite*): voir la Rec. UIT-T X.290 [1].

3.1.2 port de communication: mécanisme abstrait facilitant la communication entre composants de test.

NOTE – Un port de communication est modélisé comme une file d'attente du type premier entré, premier sorti (FIFO, *first in, first out)* dans le sens réception. Les ports peuvent être en mode message, en mode procédure, ou utiliser un mélange des deux modes.

3.1.3 suite de tests exécutables (ETS, *executable test suite*): voir la Rec. UIT-T X.290 [1].

3.1.4 temporisateur explicite: temporisateur qui est déclaré dans une suite ATS TTCN-3 et auquel permettent d'accéder les opérations de temporisation TTCN-3.

3.1.5 informations supplémentaires sur l'implémentation destinées au test (IXIT, *implementation extra information for testing*): voir la Rec. UIT-T X.290 [1].

3.1.6 temporisateur implicite: temporisateur intégré au système, créé par l'exécutable TTCN-3 pour garder un appel TTCN-3 ou une opération d'exécution.

NOTE – Les temporisateurs implicites ne sont pas accessibles à l'utilisateur TTCN-3.

3.1.7 adaptateur de plate-forme (PA, *platform adapter*): entité qui adapte l'exécutable TTCN-3 à telle ou telle plate-forme d'exécution.

NOTE – L'adaptateur de plate-forme crée une notion de temps unique pour un système de test TTCN-3 et implémente des fonctions externes ainsi que des temporisateurs explicites et implicites.

3.1.8 adaptateur du système sous test (SA): entité qui adapte les opérations de communication TTCN-3 avec le système sous test sur la base de l'interface d'un système de test abstrait et qui implémente l'interface du système de test réel.

3.1.9 système sous test (SUT, *system under test*): voir la Rec. UIT-T X.290 [1].

NOTE – Tous les types sont connus au moment de la compilation, c'est-à-dire qu'ils sont statiquement associés.

3.1.10 test élémentaire: voir la Rec. UIT-T X.290 [1].

3.1.11 événement de test: données de test envoyées ou reçues (message ou appel de procédure) sur un port de communication faisant partie de l'interface du système de test.

3.1.12 entité de gestion de test (TM, *test management*): entité qui assure une interface avec l'utilisateur et gère le système de test TTCN-3.

3.1.13 système de test: voir la Rec. UIT-T X.290 [1].

3.1.14 interface du système de test: composant de test qui assure un mappage des ports disponibles dans le système de test (abstrait) TTCN-3 avec les ports offerts par un système de test réel.

3.1.15 identification de temporisateur (TID, *timer identification*): identification unique des instances du temporisateur explicite ou implicite qui est émise par l'exécutable TTCN-3.

3.1.16 interface de commande TTCN-3 (TCI, *TTCN-3 control interface*): actuellement, interface propriétaire qui définit l'interaction entre l'entité de gestion de test et l'exécutable TTCN-3 dans un système de test.

3.1.17 exécutable TTCN-3 (TE): partie d'un système de test assurant l'interprétation ou l'exécution d'une suite ETS TTCN-3.

3.1.18 interface d'exécution TTCN-3 (TRI, *TTCN-3 runtime interface*): interface qui définit l'interaction de l'exécutable TTCN-3 avec le système SUT et l'adaptateur de plate-forme dans un système de test.

3.2 Abréviations

La présente Recommandation utilise les abréviations suivantes:

ATS	suite de tests abstraits (*abstract test suite*)
CH	dispositif de traitement de composant (*component handler*)
ECD	codecs externes (*external codecs*)

EDS système de codage/décodage (*encoding/decoding system*)

ETS suite de tests exécutables (*executable test suite*)

IDL langage de définition d'interface (*interface definition language*)

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

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

MTC composant de test principal (*main test component*)

OMG groupe de gestion d'objets (*object management group*)

PA adaptateur de plate-forme (*platform adapter*)

SA adaptateur du système sous test (*SUT adapter*)

SUT système sous test (*system under test*)

T3RTS système d'exécution TTCN-3 (*TTCN-3 runtime system*)

TC commande d'exécution de test (*test control*)

TCI interface de commande TTCN-3 (*TTCN-3 control interface*)

TE exécutable TTCN-3 (*TTCN-3 executable*)

TID identification de temporisateur (*timer identification*)

TL consignation d'événements de test (*test logging*)

TM entité de gestion de test (*test management*)

TRI interface d'exécution TTCN-3 (*TTCN-3 runtime interface*)

TSI interface du système de test (*test system interface*)

TTCN notation de test et de commande de test (*testing and test control notation*)

TTCN-3 notation de test et de commande de test version 3 (*testing ans test control notation version 3*)

4 Structure générale d'un système de test TTCN-3

Théoriquement parlant, un système de test TTCN-3 peut être représenté sous la forme d'un ensemble d'entités interactives dans lequel chaque entité correspond à un élément de fonctionnalité donné dans une implémentation de système de test. Ces entités gèrent l'exécution du test, en interprétant ou en exécutant le code TTCN-3 compilé, entrent dûment en communication avec le système SUT, implémentent les fonctions externes et traitent les opérations de temporisation. (Voir Figure 1.)

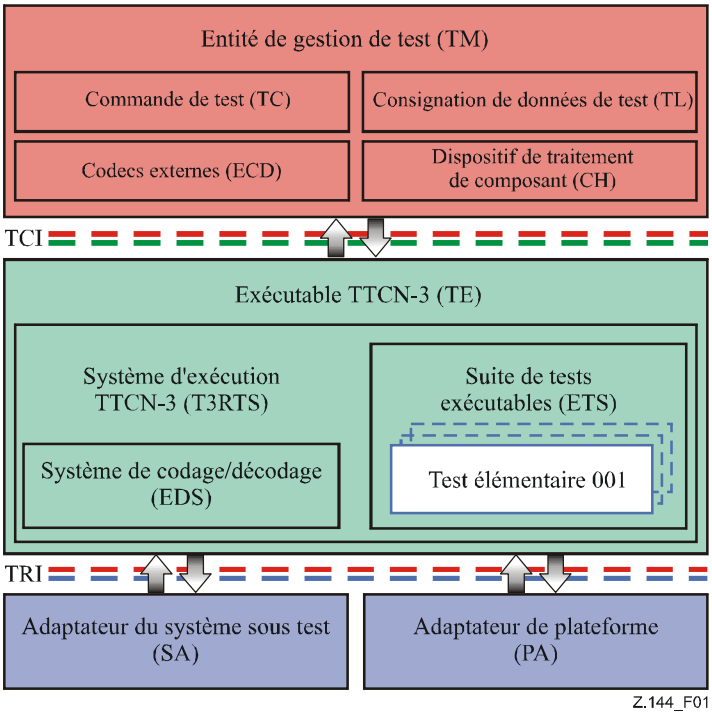

Figure 1/Z.144 – Structure générale d'un système de test TTCN-3

4.1 Entités d'un système de test TTCN-3

La structure d'implémentation d'un système de test TTCN-3 est représentée à la Figure 1. Il convient de noter que la décomposition de l'entité TM en ses entités constitutives, telles qu'elles sont représentées à la Figure 1 et utilisées dans les paragraphes suivants de la présente Recommandation, vise uniquement à faciliter la définition des interfaces du système de test TTCN-3.

La partie du système de test qui assure l'interprétation et l'exécution des modules TTCN-3, c'est-à-dire la suite de tests exécutables (ETS, *executable test suite*), fait partie de l'exécutable TTCN-3 (TE, *TTCN-3 executable*). Cette partie correspond au code exécutable produit par un compilateur TTCN-3 ou un interpréteur TTCN-3 dans une implémentation de système de test. On part du principe qu'une telle implémentation inclut la suite ETS obtenue à partir d'une suite ATS TTCN-3.

La partie restante du système de test TTCN-3, qui porte sur des aspects impossibles à déterminer à partir des seules informations présentes dans la suite ATS initale, est constituée de l'entité de gestion de test (TM, *test management*), de l'adaptateur du système sous test (SA, *SUT adapter*) et de l'adapteur de plate-forme (PA, *platform adapter*). En règle générale, ces entités ont trait à une interface utilisateur – système de test, la commande d'exécution de test, la consignation d'événements de test, ainsi que la communication avec le système SUT et l'implémentation de temporisateur.

4.1.1 Entité de gestion de test (TM)

Dans l'entité TM, on distingue deux fonctionnalités se rapportant respectivement à la commande d'exécution de test et à la consignation d'événements de test.

4.1.1.1 Commande d'exécution de test (TC, *test control*)

La gestion générale du système de test incombe à l'entité TC qui, une fois le système de test initialisé, lance l'exécution du test. L'entité TC est chargée d'assurer de manière appropriée l'invocation des modules TTCN-3, c'est-à-dire de transmettre les paramètres de ces modules et/ou les informations IXIT à l'entité TE si besoin est. En règle générale, cette entité implémentera également une interface utilisateur-système de test.

4.1.1.2 Consignation d'événements de test (TL, *test logging*)

L'entité TL est chargée de tenir à jour le registre de consignation des événements de test. Elle est expressément chargée par l'entité TE de consigner les événements de test. Elle dispose d'une interface unidirectionnelle où toute entité faisant partie de l'entité TE peut poster à son intention une demande de consignation. Une interface interne à l'entité TM peut aussi être utilisée pour enregistrer les informations de gestion de test émises par l'entité TC.

4.1.1.3 Codecs externes (ECD, *external codecs*)

Les entités ECD sont chargées, à titre facultatif, de coder et décoder les données associées aux communications en mode message ou en mode procédure à l'intérieur de l'entité TE. Les codecs externes peuvent être utilisés parallèlement aux codecs intégrés associés à l'entité TE ou la place de ceux-ci. Contrairement aux codecs intégrés, les codecs externes sont munis d'une interface normalisée qui permet de les porter d'un système ou d'un utilitaire TTCN-3 à un autre.

4.1.1.4 Dispositif de traitement de composant (CH, *component handler*)

L'entité CH est chargée de répartir les composants de test en parallèle. Ceux-ci peuvent être répartis à l'intérieur d'un même système physique ou entre plusieurs systèmes physiques. L'entité CH autorise l'entité de gestion de test à créer des systèmes de test répartis et à en assurer la commande d'une manière qui soit transparente et indépendante de l'entité TE.

4.1.2 Exécutable TTCN-3 (TE)

L'entité TE est chargée d'assurer l'interprétation ou l'exécution de la suite ATS TTCN-3. Théoriquement, l'entité TE peut se décomposer en trois entités interactives: une suite ETS, le système d'exécution TTCN-3 (T3RTS) et le système de codage/décodage (EDS, *encoding/decoding system*). Il est à noter que cette décomposition de l'entité TE en ses entités constitutives vise uniquement à faciliter, en théorie, la définition des interfaces du système de test TTCN-3; il n'est absolument pas nécessaire de reproduite cette décomposition dans les implémentations TRI.

Les paragraphes qui suivent définissent les attributions de chaque entité et examinent également le traitement des temporisateurs dans l'interface TRI.

4.1.2.1 Suite de tests exécutables (ETS)

L'entité ETS assure l'exécution ou l'interprétation des tests élémentaires, le séquencement et l'appariement des événements de test, tels qu'ils sont définis dans les modules TTCN-3 correspondants de la Rec. UIT-T Z.140 [2]. Elle interagit avec l'entité T3RTS pour envoyer, tenter de recevoir (ou d'apparier) et consigner des événements de test durant l'exécution des tests élémentaires, pour créer et supprimer des composants de test TTCN-3 ainsi que pour traiter des appels de fonctions externes, des opérations-actions et des temporisations. Il est à noter que l'entité ETS n'interagit pas directement avec l'entité SA via l'interface TRI.

4.1.2.2 Système d'exécution TTCN-3 (T3RTS)

L'entité T3RTS interagit avec les entités TM, SA et PA via les interfaces TCI et TRI, et gère les entités ETS et EDS. Elle initialise les adaptateurs ainsi que les entités ETS et EDS. Elle exécute toutes les actions nécessaires pour lancer de manière appropriée l'exécution d'un test élémentaire ou d'une fonction avec les paramètres de l'entité ETS. Elle interroge l'entité TM sur les valeurs des paramètres du module requises par l'entité ETS et lui envoie les informations de consignation. En outre, elle recueille et résout les verdicts associés retournés par l'entité ETS, comme défini dans la Rec. UIT-T Z.140 [2].

L'entité T3RTS implémente la création et la suppression de composants de test TTCN-3 ainsi que la sémantique TTCN-3 de communication en mode message et en mode procédure, d'appels de fonctions externes, d'opérations-actions et de temporisateurs. Cela suppose notamment qu'elle indique à l'adaptateur du système sous test (SA) quel message ou appel de procédure doit être envoyé au système SUT, ou qu'elle indique à l'adaptateur de plate-forme (PA) quelle fonction externe doit être exécutée ou quels temporisateurs doivent être déclenchés, arrêtés, interrogés ou lus. De même, l'entité T3RTS informe l'entité ETS des messages ou des appels de procédure entrant émanant du système SUT ainsi que des événements d'expiration de temporisation.

Avant d'envoyer ou de recevoir des messages et des appels de procédure à destination ou en provenance de l'entité SA, ou de traiter des appels de fonction et des opérations-actions dans l'entité PA pour l'entité ETS, l'entité T3RTS invoque l'entité EDS afin qu'elle procède au codage ou au décodage. L'entité T3RTS devrait implémenter toutes les opérations de communication en mode message et en mode procédure entre les composants de test, mais uniquement la sémantique TTCN-3 de communication en mode procédure avec le système SUT, c'est-à-dire bloquer et débloquer, éventuellement, l'exécution des composants de test, en utilisant des temporisateurs implicites pour la garde et en traitant les exceptions d'expiration de temporisation découlant de ces opérations de communication. Toutes les opérations de communication en mode procédure avec le système SUT doivent être réalisées et identifiées (dans le cas d'une opération de réception) dans l'entité SA du fait que ces opérations trouvent leur modalité d'implémentation la plus efficace dans une plate-forme. Il est à noter que les temporisateurs associés aux opérations d'appel de procédure, c'est-à-dire les temporisateurs implicites, sont implémentés dans l'adaptateur de plate-forme (PA).

L'exécutable TTCN-3 est tenu d'actualiser ses propres files d'attente de port (différentes de celles qui peuvent exister dans les entités SA ou PA) relatives aux événements de test d'entrée afin d'effectuer des instantanés des opérations de réception comme défini dans la Rec. UIT-T Z.140 [2]. Les événements d'expiration de temporisation, qui sont émis par les implémentations de temporisateur TTCN-3, de temporisateur d'appel ou de temporisateur des tests élémentaires,

doivent être conservés dans une liste de fins de temporisation telle que définie dans la Rec. UIT-T X.292 [3]. Comme il ressort de la Figure 2, cette fonctionnalité a été intégralement attribuée à l'entité T3RTS. Celle-ci est chargée d'enregistrer les événements que l'entité SA ou PA a signalés à l'entité TE, mais qui doivent encore être traités.

4.1.2.3 Système de codage/décodage (EDS)

L'entité EDS est chargée du codage et du décodage des données de test, y compris les données utilisées dans les opérations de communication avec le système SUT, comme spécifié dans le module TTCN-3 d'exécution. Si aucun codage n'a été spécifié pour un module TTCN-3, le codage des valeurs des données est fonction de l'utilitaire utilisé. Cette entité est invoquée par l'entité T3RTS et lui envoie les données de retour. Il est à noter que l'entité EDS n'interagit pas directement avec l'entité SA via l'interface TRI.

4.1.2.4 Temporisateurs de l'exécutable TTCN-3

Les temporisateurs qui ont été déclarés et nommés dans la suite ATS TTCN-3 peuvent théoriquement être classés comme étant des temporisateurs explicites de l'entité TE. Les temporisateurs qui sont créés par l'entité TE pour garder des appels de procédure TTCN-3 ou des opérations d'exécution sont connus dans l'entité TE comme étant des temporisateurs implicites. Les temporisateurs explicites et les temporisateurs implicites sont créés dans l'entité TE, mais ils sont implémentés par l'adaptateur de plate-forme (PA). Pour cela une identification de temporisateur (TID) unique est générée pour tout temporisateur créé dans l'entité TE. Cette identification TID unique devrait permettre à l'entité TE d'établir une distinction entre différents temporisateurs. L'identification TID doit être utilisée par l'entité TE pour interagir avec l'implémentation du temporisateur correspondant dans l'entité PA.

Il est à noter qu'il incombe à l'entité TE d'implémenter correctement les différentes sémantiques TTCN-3 applicables aux temporisateurs explicites et aux temporisateurs implicites, comme défini dans la Rec. UIT-T Z.140 [2]; ainsi l'utilisation des mots clés `any` et `all` avec des temporisateurs ne s'applique qu'aux temporisateurs explicites. Dans l'entité PA, tous les temporisateurs, implicites comme explicites, sont traités de la même manière.

4.1.3 Adaptateur du système sous test (SA)

L'entité SA adapte la communication en mode message et en mode procédure du système de test TTCN-3 avec le système SUT à telle ou telle plate-forme d'exécution du système de test. Elle a connaissance du mappage des ports de communication des composants de test TTCN-3 vers les ports de l'interface du système de test et implémente l'interface du système de test réel définie dans la Rec. UIT-T Z.140 [2]. Elle est chargée de transmettre au système SUT les demandes d'envoi et les opérations-actions SUT provenant de l'exécutable TTCN-3 (TE), et d'informer l'entité TE des événements de test reçus en les ajoutant aux files d'attente de port de l'entité TE.

Les opérations de communication en mode procédure avec le système SUT sont implémentées dans l'entité SA. Celle-ci est chargée de faire la distinction entre les différents messages de communication en mode procédure (appel, réponse et exception) et de les transmettre de la manière appropriée au système SUT ou à l'entité TE. La sémantique de communication en mode procédure TTCN-3, c'est-à-dire l'effet d'une telle opération sur l'exécution des composants de test TTCN-3, doit être traitée dans l'entité TE.

L'entité SA comporte une interface avec l'entité TE, qui est utilisée pour envoyer des messages SUT (émis dans des opérations-actions SUT TTCN-3) à l'entité SA et qui permet à ces deux entités de s'échanger des données de test codées dans des opérations de communication avec le système SUT.

4.1.4 Adaptateur de plate-forme (PA)

L'entité PA implémente les fonctions externes TTCN-3 et fournit un système de test TTCN-3 ayant une notion de temps unique. C'est dans cette entité que doivent être implémentés les fonctions externes ainsi que tous les temporisateurs. Il est à signaler que les instances de temporisateur sont créées dans l'entité TE. Un temporisateur de l'entité PA se reconnaît à sa seule identification de temporisateur (TID). C'est pourquoi l'entité PA traite les temporisateurs explicites et les temporisateurs implicites de la même manière.

L'interface avec l'entité TE permet d'invoquer des fonctions externes et de déclencher, lire et arrêter des temporisateurs ainsi que de se renseigner sur l'état des temporisateurs au moyen de leur identification (ID). L'entité PA signale à l'entité TE les temporisateurs arrivés à expiration.

4.2 Interfaces d'un système de test TTCN-3

Comme indiqué précédemment à la Figure 1, un système de test TTCN-3 comporte deux interfaces, l'interface de commande TTCN-3 (TCI) et l'interface d'exécution TTCN-3 (TRI), qui sont respectivement l'interface entre les entités TM (gestion de test) et TE (exécutable TTCN-3) et l'interface entre les entités TE, SA (adaptateur du système sous test) et PA (adaptateur de plate-forme).

La présente Recommandation définit l'interface TRI. L'interaction de l'entité TE avec les entités SA et PA sera définie ici pour ce qui est des opérations TRI. Bien que l'interface TRI et l'interface TCI doivent toutes deux être définies aux fins de l'implémentation complète d'un système de test TTCN-3, la spécification et l'implémentation de l'interface TCI sont actuellement considérées comme étant propriétaires.

4.3 Conditions d'exécution applicables à un système de test TTCN-3

Chaque appel d'opération TRI doit être traité comme une opération atomique dans l'entité appelante. L'entité appelée, qui implémente une opération TRI, doit redonner la commande à l'entité appelante dès que l'effet recherché de cette opération a été obtenu ou si l'opération ne peut pas être menée à bonne fin. L'entité appelée ne doit pas opérer de blocage au cours de l'implémentation de la communication en mode procédure. Mais elle doit opérer un blocage après l'invocation de l'implémentation d'une fonction externe et attendre de recevoir la valeur de retour de cette fonction. Il est à signaler que selon l'implémentation du système de test, la non-réception de la valeur de retour de l'implémentation d'une fonction externe peut entraîner un blocage d'une durée illimitée de l'exécution des composants de test, de l'exécutable TTCN-3, de l'adaptateur de plate-forme, voire du système de test tout entier.

Les conditions d'exécution énoncées ci-dessus peuvent être remplies dans une implémentation de système de test hautement intégrée. Ici, la totalité du système de test TTCN-3 est implémentée dans un seul exécutable ou processus dans lequel au moins un fil d'exécution est attribué à chaque entité du système de test. Les opérations TRI peuvent être implémentées ici sous forme d'appels de procédure.

Il est à noter qu'une implémentation de système de test moins intégrée demeure possible, par exemple une implémentation d'un système de test TTCN-3 avec plusieurs adaptateurs de système sous test dans un environnement informatique réparti. Dans ce cas, une petite partie seulement des adaptateurs de système sous test est hautement intégrée au reste du système de test TTCN-3, alors que les adaptateurs de système SUT (SA) effectivement mis en œuvre peuvent faire l'objet de processus séparés. Cette petite partie des entités SA peut alors n'implémenter qu'un routage de l'information fournie par les opérations TRI vers les processus des adaptateurs de système sous test voulus, éventuellement en cours d'exécution sur des serveurs distants, et vice versa.

5 Interface d'exécution TTCN-3 et opérations

Le présente paragraphe définit les opérations TRI pour ce qui est des moments où elles doivent être utilisées et l'effet qu'elles sont censées avoir dans une implémentation de système de test TTCN-3. Il définit également un ensemble de types de données abstraits qui est ensuite utilisé pour la définition des opérations TRI. Cette définition comporte en outre une description plus détaillée des paramètres d'entrée requis pour chaque appel d'opération TRI et la valeur de retour de l'opération.

5.1 Aperçu général de l'interface TRI

L'interface TRI définit l'interaction entre les entités TE (exécutable TTCN-3), SA (adaptateur de système sous test) et PA (adaptateur de plate-forme) dans une implémentation de système de test TTCN-3. Théoriquement, elle permet à l'entité TE d'envoyer des données de test au système SUT ou de manipuler des temporisateurs et, de même, d'informer l'entité TE des données de test reçues et de l'expiration des temporisations.

L'interface TRI peut être considérée comme étant constituée de deux sous-interfaces: une interface triCommunication et une interface triPlatform. L'interface triCommunication assure la communication d'une suite ETS TTCN-3 avec le système SUT qui est implémenté dans l'entité SA. L'interface triPlatform assure un ensemble d'opérations permettant d'adapter une suite ETS à une plate-forme d'exécution donnée.

Les deux interfaces sont bidirectionnelles de façon que l'appelant et l'appelé résident dans les entités TE, SA et PA du système de test. Le Tableau 1 donne un aperçu général plus détaillé de la relation appelant/appelé entre les différentes entités. Il est à signaler que ce tableau indique uniquement les interactions visibles au niveau de l'interface TRI. La communication interne entre les parties d'une même entité n'y est pas représentée, étant donné que la structure interne de l'entité TE, SA ou PA peut différer d'une implémentation de système de test TTCN-3 à une autre.

Tableau 1/Z.144 – Aperçu général des interfaces

Interface	Sens (entité appelante → entité appelée)	
Nom	TE → SA ou PA	SA ou PA → TE
triCommunication	TE → SA	SA → TE
triPlatform	TE → PA	PA → TE

5.1.1 Interface triCommunication

Cette interface assure les opérations qui sont nécessaires pour implémenter la communication de la suite ETS TTCN-3 avec le système SUT, à savoir les opérations d'initialisation de l'interface du système de test (TSI), d'établissement des connections avec le système SUT et de traitement de la communication en mode message et en mode procédure avec le système SUT. En outre, l'interface triCommunication assure une opération de réinitialisation de l'adaptateur de système sous test (SA).

5.1.2 Interface triPlatform

Cette interface assure toutes les opérations nécessaires pour adapter l'exécutable TTCN-3 à une plate-forme d'exécution donnée. L'interface triPlatform permet de déclencher, arrêter et lire un temporisateur, de se renseigner sur l'état de celui-ci et d'ajouter des événements d'expiration de temporisation à la liste de fins de temporisation. En outre, elle assure les opérations d'appel de fonctions externes TTCN-3 et de réinitialisation de l'adaptateur de plate-forme (PA). Il est à signaler qu'aucune distinction entre temporisateurs explicites et temporisateurs implicites n'est à établir au niveau de l'interface triPlatform. Au contraire, chaque temporisateur doit être traité de façon uniforme avec son identificateur (TID).

5.1.3 Corrélation entre l'invocation d'opérations TTCN-3 et l'invocation d'opérations TRI

Il existe une corrélation directe entre l'invocation de certaines opérations TTCN-3 et l'invocation d'une opération TRI (ou éventuellement deux dans le cas des opérations TTCN-3 `execute` et `call`), comme indiqué dans le Tableau 2. Pour l'invocation de toutes les autres opérations TRI, il peut n'y avoir aucune corrélation directe.

La corrélation indiquée pour les opérations de communication TTCN-3 (c'est-à-dire les opérations `send`, `call`, `reply`, et `raise`) ne vaut que si ces opérations sont invoquées sur un port de composant de test qui est mappé avec un port TSI. Néanmoins, cette corrélation vaut pour l'invocation de l'ensemble de ces opérations si aucun composant de système n'a été spécifié pour un test élémentaire, c'est-à-dire que seul le composant de test principal (MTC) est créé pour un test élémentaire à l'exclusion de tout autre composant de test.

Tableau 2/Z.144 – Corrélation entre l'invocation d'opérations TTCN-3 et l'invocation d'opérations TRI (* = s'il y a lieu)

Nom d'opération TTCN-3	Nom d'opération TRI	Nom d'interface TRI
execute	triExecuteTestCase triStartTimer*	TriCommunication TriPlatform
map	triMap	TriCommunication
unmap	triUnmap	TriCommunication
send	triSend (see Note 1)	TriCommunication
	triSendBC (see Note 2)	
	triSendMC (see Note 3)	
call	triCall (see Note 1)	TriCommunication
	triCallBC (see Note 2)	
	triCallMC (see Note 3)	
	triStartTimer*	TriPlatform
reply	triReply (see Note 1)	TriCommunication
	triReply (see Note 2)	
	triReply (see Note 3)	
raise	triRaise (see Note 1)	TriCommunication
	triRaise (see Note 2)	
	triRaise (see Note 3)	
action	triSUTactionInformal	TriCommunication

Tableau 2/Z.144 – Corrélation entre l'invocation d'opérations TTCN-3 et l'invocation d'opérations TRI (* = s'il y a lieu)

Nom d'opération TTCN-3	Nom d'opération TRI	Nom d'interface TRI
`start (timer)`	`triStartTimer`	`TriPlatform`
`stop (timer)`	`triStopTimer`	`TriPlatform`
`read (timer)`	`triReadTimer`	`TriPlatform`
`running (timer)`	`triTimerRunning`	`TriPlatform`
`TTCN-3 external function`	`triExternalFunction`	`TriPlatform`
NOTE 1 – Pour une communication en mode unidiffusion. NOTE 2 – Pour une communication en mode diffusion. NOTE 3 – Pour une communication en mode multidiffusion.		

Il est à noter que toutes les opérations TRI énumérées dans le Tableau 2 sont utilisées par l'entité TE et que celle-ci peut implémenter l'invocation de ces opérations différemment lors de l'évaluation d'un instantané TTCN dans la suite ETS TTCN-3.

5.2 Traitement des erreurs

Une procédure explicite de traitement des erreurs n'est définie que pour les opérations TRI appelées par l'exécutable TTCN-3 (TE). L'entité SA ou PA signale l'état d'une opération TRI dans la valeur de retour de cette opération. La valeur d'état peut indiquer la réussite locale (***TRI_OK***) ou l'échec local (***TRI_Error***) de l'opération TRI. En conséquence, l'entité TE peut réagir à une erreur qui s'est produite dans l'entité SA ou PA et émettre, par exemple, un message d'erreur de test élémentaire.

Pour les opérations TRI appelées par l'entité SA ou PA, aucune procédure explicite de traitement des erreurs n'est nécessaire puisque ces opérations sont implémentées dans l'entité TE. Ici, l'entité TE dirige l'exécution du test dans le cas où une erreur se produit au cours d'une telle opération TRI.

Il est à signaler que les codes d'erreur à utiliser ainsi que la détection et le traitement des erreurs dans les différentes entités du système de test ne relèvent pas de la spécification TRI actuelle.

5.3 Interface de données

Dans les opérations TRI, seules les données de test codées doivent être transmises. Il incombe à l'exécutable TTCN-3 (TE) de coder les données de test à envoyer et de décoder les données de test reçues au cours des différentes opérations TRI puisque les règles de codage peuvent être définies pour un module TTCN-3 ou à l'intérieur de celui-ci. Il est à signaler que l'entité TE est tenue de coder les données de test même si aucune information de codage n'a été communiquée dans une suite ATS TTCN-3. Dans ce cas, il appartient au fournisseur de l'utilitaire de définir un codage.

Au lieu de définir une interface de données explicite pour les types de données TTCN-3 et ASN.1, la norme TRI définit un ensemble de types de données abstraits. Ces types de données sont utilisés dans la définition suivante des opérations TRI pour indiquer les informations que l'entité appelante doit transmettre à l'entité appelée, et vice versa. La représentation concrète de ces types de données abstraits ainsi que la définition des types de données de base sont indiquées aux paragraphes 6 et 7 dans les mappages vers les différents langages.

Il est à signaler que les valeurs de tout type de données d'identification doivent être uniques dans l'implémentation du système de test; on entend par là qu'elles doivent être différentes de toutes les autres valeurs au niveau global à quelque moment que ce soit.

Les types de données abstraits suivants sont définis et utilisés pour la définition des opérations TRI.

5.3.1 Connexion

`TriComponentIdType` Une valeur de type `TriComponentIdType` comporte un identificateur, un nom et le type de composant. La valeur distincte de ce dernier est le nom de type de composant spécifié dans la suite ATS TTCN-3. Ce type abstrait est principalement utilisé pour résoudre des opérations de communication TRI sur des ports TSI qui ont des mappages avec de nombreux ports de composant de test.

`TriComponentIdListType` Une valeur de type `TriComponentIdListType` est une liste de `TriComponentIdType`. Ce type abstrait est utilisé pour les communications en mode multidiffusion dans l'interface TCI.

`TriPortIdType` Une valeur de type `TriPortIdType` comporte une valeur de type `TriComponentIdType` représentant le composant auquel le port appartient, un indice du port (s'il est présent) et le nom du port spécifié dans la suite ATS TTCN-3. Le type `TriPortIdType` est principalement requis pour la transmission de l'entité TE à l'entité SA des informations relatives à l'interface TSI et aux connexions avec cette interface.

`TriPortIdListType` Une valeur de type `TriPortIdListType` est une liste de `TriPortIdType`. Ce type abstrait est utilisé à des fins d'initialisation après l'invocation d'un test élémentaire TTCN-3.

5.3.2 Communication

`TriMessageType` Une valeur de type `TriMessageType` est constituée de données de test codées qui doivent être envoyées au système SUT ou qui ont été reçues en provenance de celui-ci.

`TriAddressType` Une valeur de type `TriAddressType` indique une adresse d'origine ou de destination figurant dans le système SUT. Ce type abstrait peut être utilisé dans des opérations de communication TRI; il s'agit d'un type ouvert, qui est opaque pour l'entité TE.

`TriAddressListType` Une valeur de type `TriAddressListType` est une liste de `TriAddressType`. Ce type abstrait est utilisé pour des communications en mode multidiffusion dans l'interface TRI.

`TriSignatureIdType` Une valeur de type `TriSignatureIdType` est le nom d'une procédure de signature spécifiée dans la suite ATS TTCN-3. Ce type abstrait est utilisé dans des opérations de communication TRI en mode procédure.

`TriParameterType` Une valeur de type `TriParameterType` comporte un paramètre codé et une valeur de `TriParameterPassingModeType` représentant le mode de transmission spécifié pour le paramètre dans la suite ATS TTCN-3.

`TriParameterPassingModeType` Une valeur de type `TriParameterPassingModeType` est un mot clé *in*, *inout*, ou *out*. Ce type abstrait est utilisé dans les opérations de communication TRI en mode procédure et pour les appels de fonctions externes.

`TriParameterListType` Une valeur de type `TriParameterListType` est une liste de `TriParameterType`. Ce type abstrait est utilisé dans les opérations de communication TRI en mode procédure et pour les appels de fonctions externes.

`TriExceptionType` Une valeur de type `TriExceptionType` est un type et une valeur codés d'une exception qui doivent être envoyés au système SUT ou qui ont été reçus en provenance de celui-ci. Ce type abstrait est utilisé dans les opérations de communication TRI en mode procédure.

5.3.3 Temporisateur

`TriTimerIdType` Une valeur de type `TriTimerIdType` désigne l'identificateur d'un temporisateur. Ce type abstrait est nécessaire pour toutes les opérations de temporisation TRI.

`TriTimerDurationType` Une valeur de type `TriTimerDurationType` indique la durée d'une temporisation en secondes.

5.3.4 Divers

`TriTestCaseIdType` Une valeur de type `TriTestCaseIdType` est le nom d'un test élémentaire tel que spécifié dans la suite ATS TTCN-3.

`TriFunctionIdType` Une valeur de type `TriFunctionIdType` est le nom d'une fonction externe telle que spécifiée dans la suite ATS TTCN-3.

`TriStatusType` Une valeur de type `TriStatusType` est soit ***TRI_OK*** soit ***TRI_Error*** indiquant la réussite ou l'échec d'une opération TRI.

5.4 Description des opérations

Toutes les définitions des opérations utilisent le langage de définition d'interface (IDL, *interface definition language*). Les mappages vers des langages concrets sont définis dans les paragraphes 6 et 7.

Pour chaque appel d'opération TRI, tous les Paramètres In, *inout* et *out* énumérés dans la définition de l'opération considérée sont obligatoires. La valeur d'un paramètre *in* est spécifiée par l'entité appelante. De même, la valeur d'un paramètre *out* est spécifiée par l'entité appelée. Dans le cas d'un paramètre *inout*, une valeur est d'abord spécifiée par l'entité appelante, mais cette valeur peut être remplacée par une nouvelle valeur spécifiée par l'entité appelée. Il est à noter que bien que la notation TTCN-3 utilise également les Paramètres In, *inout* et *out* pour les définitions de signature, les dénotations utilisées dans la spécification IDL TRI n'ont pas de lien avec celles qui figurent dans une spécification TTCN-3.

Les appels d'opération devraient utiliser une valeur réservée pour indiquer l'absence de paramètres définis comme étant facultatifs dans la description des paramètres TRI correspondants. Les valeurs réservées pour ces types sont définies dans les mappages vers les différents langages et seront désignées par la suite par la valeur `null`.

Toutes les fonctions de l'interface sont décrites selon le modèle suivant:

Numéro	**Nom d'opération** entité appelante → entité appelée
Signature	Signature-IDL.
Paramètres In	Description des données transmises sous forme de paramètres de l'opération, de l'entité appelante à l'entité appelée.
Paramètres Out	Description des données transmises sous forme de paramètres de l'opération, de l'entité appelée à l'entité appelante.
Paramètres InOut	Description des données transmises sous forme de paramètres de l'opération, de l'entité appelante à l'entité appelée, puis retransmises de l'entité appelée à l'entité appelante.
Valeur de retour	Description des données de retour de l'opération transmises à l'entité appelante.
Contraintes	Description des éventuelles contraintes que pose la procédure d'appel de l'opération.
Effet	Comportement exigé de l'entité appelée avant que l'opération puisse envoyer sa valeur de retour.

5.5 Opérations d'interface de communication

5.5.1 Opération triSAReset (TE → SA)

Signature	`TriStatusType triSAReset()`
Paramètres In	s.o.
Paramètres Out	s.o.
Valeur de retour	Etat de retour de l'opération `triSAReset`**.** L'état de retour indique la réussite locale (***TRI_OK***) ou l'échec local (***TRI_Error***) de l'opération.
Contraintes	Cette opération peut être appelée par l'entité TE à tout moment pour réinitialiser l'entité SA.
Effet	L'entité SA doit réinitialiser tous les moyens de communication dont elle assure la maintenance. Elle doit, par exemple, réinitialiser les connexions statiques aboutissant au système SUT, fermer les connexions dynamiques aboutissant au système SUT et rejeter les messages ou les appels de procédure en instance. L'opération `triResetSA` retourne ***TRI_OK*** dans le cas où l'opération a été exécutée avec succès, et ***TRI_Error*** dans le cas contraire.

5.5.2 Opérations de traitement de connexion

5.5.2.1 Opération triExecuteTestCase (TE → SA)

Signature	`TriStatusType triExecuteTestCase ( in TriTestCaseIdType testCaseId, in TriPortIdListType tsiPortList)`
Paramètres In	`testCaseId` identificateur du test élémentaire qui va être exécuté `tsiPortList` liste des ports de l'interface du système de test définis pour ce système
Paramètres Out	s.o.
Valeur de retour	Etat de retour de l'opération `triExecuteTestCase`**.** L'état de retour indique la réussite locale (***TRI_OK***) ou l'échec local (***TRI_Error***) de l'opération**.**
Contraintes	Cette opération est appelée par l'entité TE immédiatement avant l'exécution de tout test élémentaire. Le test élémentaire qui va être exécuté est indiqué par le paramètre `testCaseId`. Le paramètre `tsiPortList` contient tous les ports qui ont été déclarés dans la définition du composant du système pour le test élémentaire, c'est-à-dire les ports TSI. Si un composant du système n'a pas été explicitement défini pour le test élémentaire dans la suite ATS TTCN-3, alors le paramètre `tsiPortList` contient tous les ports de communication du composant de test principal (MTC). Les ports figurant dans le paramètre `tsiPortList` sont classés dans l'ordre dans lequel ils apparaissent dans la déclaration du composant TTCN-3 considéré.
Effet	L'entité SA peut établir des connexions statiques aboutissant au système SUT et initialiser des moyens de communication pour les ports TSI. L'opération `triExecuteTestCase` retourne ***TRI_OK*** dans le cas où l'opération a été exécutée avec succès, et ***TRI_Error*** dans le cas contraire.

5.5.2.2 Opération triMap (TE → SA)

Signature	`TriStatusType triMap ( in TriPortIdType compPortId, in TriPortIdType tsiPortId)`
Paramètres In	`compPortId` identificateur de port du composant de test à mapper `tsiPortId` identificateur de port de l'interface du système de test à mapper
Paramètres Out	s.o.
Valeur de retour	Etat de retour de l'opération `triMap`. L'état de retour indique la réussite locale (***TRI_OK***) ou l'échec local (***TRI_Error***) de l'opération.
Contraintes	Cette opération est appelée par l'entité TE lorsqu'elle exécute une opération de mappage TTCN-3.
Effet	L'entité SA peut établir une connexion dynamique aboutissant au système SUT pour le port TSI désigné. L'opération `triMap` retourne ***TRI_Error*** dans le cas où une connexion n'a pas pu être établie, et ***TRI_OK*** dans le cas contraire. L'opération devrait retourner ***TRI_OK*** dans le cas où le système de test n'a pas besoin d'établir de connexion dynamique.

5.5.2.3 Opération triUnmap (TE → SA)

Signature	`TriStatusType triUnmap ( in TriPortIdType compPortId, in TriPortIdType tsiPortId)`
Paramètres In	`compPortId` identificateur de port du composant de test à démapper `tsiPortId` identificateur de port de l'interface du système de test à démapper
Paramètres Out	s.o.
Valeur de retour	Etat de retour de l'opération `triUnmap`**.** L'état de retour indique la réussite locale (***TRI_OK***) ou l'échec local (***TRI_Error***) de l'opération.
Contraintes	Cette opération est appelée par l'entité TE lorsqu'elle exécute toute opération de démappage TTCN-3.
Effet	L'entité SA doit fermer une connexion dynamique aboutissant au système SUT pour le port TSI désigné. L'opération triUnmap retourne ***TRI_Error*** dans le cas où une connexion n'a pas pu être fermée ou dans le cas où cette connexion n'a pas été établie précédemment, et ***TRI_OK*** dans le cas contraire. L'opération devrait retourner ***TRI_OK*** dans le cas où aucune connexion dynamique n'a été établie par le système de test.

5.5.3 Opérations de communication en mode message

5.5.3.1 Opération triSend (TE → SA)

Signature	`TriStatusType triSend (` `in TriComponentIdType componentId,`
	`in TriPortIdType tsiPortId,`
	`in TriAddressType SUTaddress,`
	`in TriMessageType sendMessage)`

Paramètres In	`componentId` identificateur du composant de test d'origine `tsiPortId` identificateur du port de l'interface du système de test via lequel le message est envoyé à l'adaptateur du système sous test `SUTaddress` adresse de destination (facultative) figurant dans le système SUT `sendMessage` message codé à envoyer
Paramètres Out	s.o.
Valeur de retour	Etat de retour de l'opération `triSend`. L'état de retour indique la réussite locale (***TRI_OK***) ou l'échec local (***TRI_Error***) de l'opération.
Contraintes	Cette opération est appelée par l'entité TE lorsqu'elle exécute une opération d'envoi TTCN-3 en mode unidiffusion sur un port de composant qui a été mappé avec un port TSI. Cette opération est appelée par l'entité TE pour toutes les opérations d'envoi TTCN-3 si aucun composant de système n'a été spécifié pour un test élémentaire, c'est-à-dire que seul un composant de test MTC est créé pour un test élémentaire. Le codage du message `sendMessage` doit être effectué dans l'entité TE avant l'appel de cette opération TRI.
Effet	L'entité SA peut envoyer le message au système SUT. L'opération `triSend` retourne ***TRI_OK*** dans le cas où elle a été exécutée avec succès et ***TRI_Error*** dans le cas contraire. Il est à signaler que la valeur de retour ***TRI_OK*** ne signifie pas que le système SUT a reçu le message `sendMessage`.

5.5.3.2 Opération triSendBC (TE → SA)

Signature	`TriStatusType triSendBC (` `in TriComponentIdType componentId,`
	`in TriPortIdType tsiPortId,`
	`in TriMessageType sendMessage)`

Paramètres In	`componentId` identificateur du composant de test d'envoi `tsiPortId` identificateur du port de l'interface du système de test via lequel le message est envoyé à l'adaptateur du système sous test `sendMessage` message codé à envoyer
Paramètres Out	s.o.
Valeur de retour	Etat de retour de l'opération `triSend`. L'état de retour indique la réussite locale (***TRI_OK***) ou l'échec local (***TRI_Error***) de l'opération.
Contraintes	Cette opération est appelée par l'entité TE lorsqu'elle exécute une opération d'envoi TTCN-3 en mode diffusion sur un port de composant qui a été mappé avec un port TSI. Cette opération est appelée par l'entité TE pour toutes les opérations d'envoi TTCN-3 si aucun composant de système n'a été spécifié pour un test élémentaire, c'est-à-dire que seul un composant de test MTC est créé pour un test élémentaire. Le codage du message sendMessage doit être effectué dans l'entité TE avant l'appel de cette opération TRI.
Effet	L'entité SA peut envoyer le message en mode diffusion au système SUT. L'opération triSend retourne ***TRI_OK*** dans le cas où elle a été exécutée avec succès et ***TRI_Error*** dans le cas contraire. Il est à signaler que la valeur de retour ***TRI_OK*** ne signifie pas que le système SUT a reçu le message sendMessage.

5.5.3.3 Opération triSendMC (TE → SA)

Signature	`TriStatusType triSendMC (`	`in TriComponentIdType componentId,`
		`in TriPortIdType tsiPortId,`
		`in TriAddressListType SUTaddresses,`
		`in TriMessageType sendMessage)`

Paramètres In	`componentId`	identificateur du composant de test d'envoi
	`tsiPortId`	identificateur du port de l'interface du système de test via lequel le message est envoyé à l'adaptateur du système sous test
	`SUTaddresses`	adresses de destination figurant dans le système SUT
	`sendMessage`	message codé à envoyer
Paramètres Out	s.o.	
Valeur de retour	Etat de retour de l'opération `triSend`. L'état de retour indique la réussite locale (***TRI_OK***) ou l'échec local (***TRI_Error***) de l'opération.	
Contraintes	Cette opération est appelée par l'entité TE lorsqu'elle exécute une opération d'envoi TTCN-3 en mode multidiffusion sur un port de composant qui a été mappé avec un port TSI. Cette opération est appelée par l'entité TE pour toutes les opérations d'envoi TTCN-3 si aucun composant de système n'a été spécifié pour un test élémentaire, c'est-à-dire que seul un composant de test MTC est créé pour un test élémentaire. Le codage du message `sendMessage` doit être effectué dans l'entité TE avant l'appel de cette opération TRI.	
Effet	L'entité SA peut envoyer le message en mode multidiffusion au système SUT. L'opération `triSend` retourne ***TRI_OK*** dans le cas où elle a été exécutée avec succès et ***TRI_Error*** dans le cas contraire. Il est à signaler que la valeur de retour ***TRI_OK*** ne signifie pas que le système SUT a reçu le message `sendMessage`.	

5.5.3.4 Opération triEnqueueMsg (SA → TE)

Signature	`void triEnqueueMsg (`	`in TriPortIdType tsiPortId,`
		`in TriAddressType SUTaddress,`
		`in TriComponentIdType componentId,`
		`in TriMessageType receivedMessage)`

Paramètres In	`tsiPortId`	identificateur du port de l'interface du système de test via lequel le message est mis en file d'attente par l'adaptateur du système sous test
	`SUTaddress`	adresse d'origine (facultative) figurant dans le système SUT
	`componentId`	identificateur du composant de test de réception
	`receivedMessage`	message codé reçu
Paramètres Out	s.o.	
Valeur de retour	vide (`void`)	
Contraintes	Cette opération est appelée par l'entité SA après qu'elle a reçu un message émanant du système SUT. Elle ne peut être utilisée que lorsque l'identificateur `tsiPortId` a été préalablement mappé sur un port de `componentId` ou a été indiqué dans l'instruction `triExecuteTestCase` précédente. Durant l'invocation d'une opération `triEnqueueMsg`, le message `receivedMessage` doit contenir une valeur codée.	
Effet	Cette opération doit transmettre le message à l'entité TE en indiquant le composant `componentId` sur lequel le port TSI `tsiPortId` est mappé. Le décodage du message `receivedMessage` doit être effectué dans l'entité TE.	

5.5.4 Opérations de communication en mode procédure

5.5.4.1 Opération triCall (TE → SA)

Signature	`TriStatusType triCall (`	`in TriComponentIdType componentId,`
		`in TriPortIdType tsiPortId,`
		`in TriAddressType SUTaddress,`
		`in TriSignatureIdType signatureId,`
		`in TriParameterListType parameterList)`

Paramètres In	`componentId`	identificateur du composant de test qui émet l'appel de procédure
	`tsiPortId`	identificateur du port de l'interface du système de test via lequel l'appel de procédure est envoyé à l'adaptateur du système sous test.
	`SUTaddress`	adresse de destination (facultative) figurant dans le système SUT
	`signatureId`	identificateur de la signature de l'appel de procédure
	`parameterList`	liste des paramètres codés qui font partie de la signature indiquée. Les paramètres figurant dans la liste `parameterList` sont classés dans l'ordre dans lequel ils apparaissent dans la déclaration de signature TTCN-3
Paramètres Out	s.o.	
Valeur de retour	Etat de retour de l'opération `triCall`. L'état de retour indique la réussite locale (***TRI_OK***) ou l'échec local (***TRI_Error***) de l'opération.	
Contraintes	Cette opération est appelée par l'entité TE lorsqu'elle exécute une opération d'appel TTCN-3 en mode unidiffusion sur un port de composant, qui a été mappé avec un port TSI. Cette opération est appelée par l'entité TE pour toutes les opérations d'appel TTCN-3 si aucun composant de système n'a été spécifié pour un test élémentaire, c'est-à-dire que seul un composant de test MTC est créé pour un test élémentaire. Tous les paramètres de procédure *in* et *inout* contiennent des valeurs codées. Les paramètres de procédure sont les paramètres spécifiés dans le modèle de signature TTCN-3. Leur codage doit être effectué dans l'entité TE avant l'appel de cette opération TRI.	
Effet	A l'invocation de cette opération, l'entité SA peut lancer l'appel de procédure correspondant à l'identificateur de signature `signatureId` et à l'identificateur de port TSI `tsiPortId`. L'opération `triCall` doit envoyer sa valeur de retour sans attendre la valeur de retour de l'appel de procédure émis (voir note). Cette opération TRI retourne *TRI_OK* en cas de lancement réussi de l'appel de procédure et *TRI_Error* dans le cas contraire. L'entité SA doit indiquer l'absence d'erreur dans le cas où la valeur d'un paramètre *out* est différente de néant. Il est à signaler que la valeur de retour de cette opération TRI ne donne aucune indication sur la réussite ou l'échec de l'appel de procédure. Il est à noter qu'une valeur d'expiration de temporisation facultative, qui peut être spécifiée dans la suite ATS TTCN-3 pour une opération d'appel, *n'est pas* incluse dans la signature de l'opération `triCall`. Il appartient à l'entité TE de remédier à cette situation en déclenchant un temporisateur pour l'opération d'appel TTCN-3 dans l'entité PA avec l'appel d'une opération TRI distincte, à savoir l'opération `triStartTimer`.	
NOTE – Pour cela, une solution pourrait, par exemple, consister à créer un nouveau fil ou un nouveau processus. Ce traitement de cet appel de procédure dépend, toutefois, de l'implémentation de l'entité TE.		

5.5.4.2 Opération triCallBC (TE → SA)

Signature	`TriStatusType triCallBC (in TriComponentIdType componentId,`
	`in TriPortIdType tsiPortId,`
	`in TriSignatureIdType signatureId,`
	`in TriParameterListType parameterList)`

Paramètres In	`componentId` identificateur du composant de test qui émet l'appel de procédure `tsiPortId` identificateur du port de l'interface du système de test via lequel l'appel de procédure est envoyé à l'adaptateur du système sous test `signatureId` identificateur de la signature de l'appel de procédure `parameterList` liste des paramètres codés qui font partie de la signature indiquée. Les paramètres figurant dans la liste `parameterList` sont classés dans l'ordre dans lequel ils apparaissent dans la déclaration de signature TTCN-3
Paramètres Out	s.o.
Valeur de retour	Etat de retour de l'opération `triCall`. L'état de retour indique la réussite locale (***TRI_OK***) ou l'échec local (***TRI_Error***) de l'opération.
Contraintes	Cette opération est appelée par l'entité TE lorsqu'elle exécute une opération d'appel TTCN-3 en mode diffusion sur un port de composant qui a été mappée avec un port TSI. Cette opération est appelée par l'entité TE pour toutes les opérations d'appel TTCN-3 si aucun composant de système n'a été spécifié pour un test élémentaire, c'est-à-dire que seul un composant de test MTC est créé pour un test élémentaire. Tous les paramètres de procédure *in* et *inout* contiennent des valeurs codées. Les paramètres de procédure sont les paramètres spécifiés dans le modèle de signature TTCN-3. Leur codage doit être effectué dans l'entité TE avant l'appel de cette opération TRI.
Effet	A l'invocation de cette opération, l'entité SA peut lancer et diffuser l'appel de procédure correspondant à l'identificateur de signature `signatureId` et au port TSI `tsiPortId`. L'opération `triCall` doit envoyer sa valeur de retour sans attendre la valeur de retour de l'appel de procédure émis (voir note). Cette opération TRI retourne *TRI_OK* en cas de lancement réussi de l'appel de procédure et *TRI_Error* dans le cas contraire. L'entité SA doit indiquer l'absence d'erreur dans le cas où la valeur d'un paramètre *out* est différente de néant. Il est à signaler que la valeur de retour de cette opération TRI ne donne aucune indication sur la réussite ou l'échec de l'appel de procédure. Il est à noter qu'une valeur d'expiration de temporisation facultative, qui peut être spécifiée dans la suite ATS TTCN-3 pour une opération d'appel n'est pas incluse dans la signature de l'opération `triCall`. Il appartient à l'entité TE de remédier à cette situation en déclenchant un temporisateur pour l'opération d'appel TTCN-3 dans l'entité PA avec l'appel d'une opération TRI distincte, à savoir l'opération `triStartTimer`.
NOTE – Pour cela, une solution pourrait par exemple consister à créer un nouveau fil ou un nouveau processus. Ce traitement de cet appel de procédure dépend, toutefois, de l'implémentation de l'entité TE.	

5.5.4.3 Opération triCallMC (TE → SA)

Signature	`TriStatusType triCallMC(`	`in TriComponentIdType componentId,`
		`in TriPortIdType tsiPortId,`
		`in TriAddressListType SUTaddresses,`
		`in TriSignatureIdType signatureId,`
		`in TriParameterListType parameterList)`

Paramètres In	`componentId`	identificateur du composant de test qui émet l'appel de procédure
	`tsiPortId`	identificateur du port de l'interface du système de test via lequel l'appel de procédure est envoyé à l'adaptateur du système sous test
	`SUTaddresses`	adresses de destination figurant dans le système SUT
	`signatureId`	identificateur de la signature de l'appel de procédure
	`parameterList`	liste des paramètres codés qui font partie de la signature indiquée. Les paramètres figurant dans la liste `parameterList` sont classés dans l'ordre dans lequel ils apparaissent dans la déclaration de signature TTCN-3
Paramètres Out	s.o.	
Valeur de retour	Etat de retour de l'opération `triCall`**.** L'état de retour indique la réussite locale (***TRI_OK***) ou l'échec local (***TRI_Error***) de l'opération.	
Contraintes	Cette opération est appelée par l'entité TE lorsqu'elle exécute une opération d'appel TTCN-3 en mode multidiffusion sur un port de composant qui est mappé avec un port TSI. Cette opération est appelée par l'entité TE pour toutes les opérations d'appel TTCN-3 si aucun composant de système n'a été spécifié pour un test élémentaire, c'est-à-dire que seul un composant de test MTC est créé pour un test élémentaire. Tous les paramètres de procédure *in* et *inout* contiennent des valeurs codées. Les paramètres de procédure sont les paramètres spécifiés dans le modèle de signature TTCN-3. Leur codage doit être effectué dans l'entité TE avant l'appel de cette opération TRI.	
Effet	A l'invocation de cette opération, l'entité SA peut lancer et multidiffuser l'appel de procédure correspondant à l'identificateur de signature `signatureId` et au port TSI `tsiPortId`**.** L'opération `triCall` doit envoyer sa valeur de retour sans attendre la valeur de retour de l'appel de procédure émis (voir note). Cette opération TRI retourne *TRI_OK* en cas de lancement réussi de l'appel de procédure et *TRI_Error* dans le cas contraire. L'entité SA doit indiquer l'absence d'erreur dans le cas où la valeur d'un paramètre *out* est différente de néant. Il est à signaler que la valeur de retour de cette opération TRI ne donne aucune indication sur la réussite ou l'échec de l'appel de procédure. Il est à noter qu'une valeur d'expiration de temporisation facultative, qui peut être spécifiée dans la suite ATS TTCN-3 pour une opération d'appel, *n'est pas* incluse dans la signature de l'opération `triCall`**.** Il appartient à l'entité TE de remédier à cette situation en déclenchant un temporisateur pour l'opération d'appel TTCN-3 dans l'entité PA avec l'appel d'une opération TRI distincte, à savoir l'opération `triStartTimer`**.**	
NOTE – Pour cela, une solution pourrait, par exemple, consister à créer un nouveau fil ou un nouveau processus. Ce traitement de cet appel de procédure dépend, toutefois, de l'implémentation de l'entité TE.		

5.5.4.4 Opération triReply (TE → SA)

Signature	`TriStatusType triReply (`	`in TriComponentIdType componentId,`
		`in TriPortIdType tsiPortId,`
		`in TriAddressType SUTaddress,`
		`in TriSignatureIdType signatureId,`
		`in TriParameterListType parameterList,`
		`in TriParameterType returnValue)`

Paramètres In	`componentId`	identificateur du composant de test qui répond
	`tsiPortId`	identificateur du port de l'interface du système de test via lequel la réponse est envoyée à l'adaptateur du système sous test
	`SUTaddress`	adresse de destination (facultative) figurant dans le système SUT
	`signatureId`	identificateur de la signature de l'appel de procédure
	parameterList	liste des paramètres codés qui font partie de la signature indiquée. Les paramètres figurant dans la liste `parameterList` sont classés dans l'ordre dans lequel ils apparaissent dans la déclaration de signature TTCN-3
	`returnValue`	valeur de retour codée (facultative) de l'appel de procédure
Paramètres Out	s.o.	
Valeur de retour	Etat de retour de l'opération `triReply`. L'état de retour indique la réussite locale (***TRI_OK***) ou l'échec local (***TRI_Error***) de l'opération.	
Contraintes	Cette opération est appelée par l'entité TE lorsqu'elle exécute une opération de réponse TTCN-3 en mode unidiffusion sur un port de composant qui a été mappé avec un port TSI. Cette opération est appelée par l'entité TE pour toutes les opérations de réponse TTCN-3 si aucun composant de système n'a été spécifié pour un test élémentaire, c'est-à-dire que seul un composant de test MTC est créé pour un test élémentaire. Tous les paramètres de procédure *out* et *inout* et la valeur de retour contiennent des valeurs codées. La liste `parameterList` contient les paramètres d'appel de procédure. Ces paramètres sont les paramètres spécifiés dans le modèle de signature TTCN-3. Leur codage doit être effectué dans l'entité TE avant l'appel de cette opération TRI. Si aucun type de retour n'a été défini pour la signature de la procédure dans la suite ATS TTCN-3, la valeur distincte null doit être transmise comme valeur de retour.	
Effet	A l'invocation de cette opération, l'entité SA peut émettre la réponse à un appel de procédure correspondant à l'identificateur de signature `signatureId` et au port TSI `tsiPortId`. L'opération `triReply` retournera ***TRI_OK*** en cas d'exécution réussie de cette opération et ***TRI_Error*** dans le cas contraire. L'entité SA doit indiquer l'absence d'erreur dans le cas où la valeur d'un paramètre *in* ou une valeur de retour non définie est différente de néant.	

5.5.4.5 Opération triReplyBC (TE → SA)

Signature	`TriStatusType triReplyBC ( in TriComponentIdType componentId, in TriPortIdType tsiPortId, in TriSignatureIdType signatureId, in TriParameterListType parameterList, in TriParameterType returnValue)`	
Paramètres In	`componentId`	identificateur du composant de test qui répond
	`tsiPortId`	identificateur du port de l'interface du système de test via lequel la réponse est envoyée à l'adaptateur du système sous test
	`signatureId`	identificateur de la signature de l'appel de procédure
	`parameterList`	liste des paramètres codés qui font partie de la signature indiquée. Les paramètres figurant dans la liste `parameterList` sont classés dans l'ordre dans lequel ils apparaissent dans la déclaration de signatureTTCN-3
	`returnValue`	valeur de retour codée (facultative) de l'appel de procédure
Paramètres Out	s.o.	
Valeur de retour	Etat de retour de l'opération `triReply`. L'état de retour indique la réussite locale (***TRI_OK***) ou l'échec local (***TRI_Error***) de l'opération.	
Contraintes	Cette opération est appelée par l'entité TE lorsqu'elle exécute une opération de réponse TTCN-3 en mode diffusion sur un port de composant qui est mappé avec un port TSI. Cette opération est appelée par l'entité TE pour toutes les opérations de réponse TTCN-3 si aucun composant de système n'a été spécifié pour un test élémentaire, c'est-à-dire que seul un composant de test MTC est créé pour un test élémentaire. Tous les paramètres de procédure *out* et *inout* et la valeur de retour contiennent des valeurs codées. La liste `parameterList` contient les paramètres de l'appel de procédure. Ces paramètres sont les paramètres spécifiés dans le modèle de signature TTCN-3. Leur codage doit être effectué dans l'entité TE avant l'appel de cette opération TRI. Si aucun type de retour n'a été défini pour la signature de la procédure dans la suite ATS TTCN-3, la valeur distincte `null` doit être transmise comme valeur de retour.	
Effet	A l'invocation de cette opération, l'entité SA peut diffuser la réponse aux appels de procédure correspondant à l'identificateur de signature `signatureId` et au port TSI `tsiPortId`. L'opération `triReply` retournera ***TRI_OK*** en cas d'exécution réussie de cette opération et ***TRI_Error*** dans le cas contraire. L'entité SA doit indiquer l'absence d'erreur dans le cas où la valeur d'un paramètre in ou une valeur de retour non définie est différente de néant.	

5.5.4.6 Opération triReplyMC (TE → SA)

Signature	`TriStatusType triReplyMC (  in TriComponentIdType componentId,`
	`in TriPortIdType tsiPortId,`
	`in TriAddressListType SUTaddresses,`
	`in TriSignatureIdType signatureId,`
	`in TriParameterListType parameterList,`
	`in TriParameterType  returnValue)`	

Paramètres In	`componentId`	identificateur du composant de test qui répond
	`tsiPortId`	identificateur du port de l'interface du système de test via lequel la réponse est envoyée à l'adaptateur du système sous test
	`SUTaddresses`	adresses de destination figurant dans le système SUT
	`signatureId`	identificateur de la signature de l'appel de procédure
	`parameterList`	liste des paramètres codés qui font partie de la signature indiquée. Les paramètres figurant dans la liste `parameterList` sont classés dans l'ordre dans lequel ils apparaissent dans la déclaration de signature TTCN-3
	`returnValue`	valeur de retour codée (facultative) de l'appel de procédure
Paramètres Out	s.o.	
Valeur de retour	Etat de retour de l'opération `triReply`. L'état de retour indique la réussite locale (***TRI_OK***) ou l'échec local (***TRI_Error***) de l'opération.	
Contraintes	Cette opération est appelée par l'entité TE lorsqu'elle exécute une opération de réponse TTCN-3 en mode multidiffusion sur un port de composant qui a été mappé avec un port TSI. Cette opération est appelée par l'entité TE pour toutes les opérations de réponse TTCN-3 si aucun composant de système n'a été spécifié pour un test élémentaire, c'est-à-dire que seul un composant de test MTC est créé pour un test élémentaire. Tous les paramètres de procédure *out* et *inout* et la valeur de retour contiennent des valeur codées. La liste `parameterList` contient les paramètres de l'appel de procédure. Ces paramètres sont les paramètres spécifiés dans le modèle de signature TTCN-3. Leur codage doit être effectué dans l'entité TE avant l'appel de cette opération TRI. Si aucun type de retour n'a été défini pour la signature de procédure dans la suite ATS TTCN-3, la valeur distincte `null` doit être transmise comme valeur de retour.	
Effet	A l'invocation de cette opération, l'entité SA peut multidiffuser la réponse aux appels de procédure correspondant à l'identificateur de signature `signatureId` et au port TSI `tsiPortId`. L'opération `triReply` retournera ***TRI_OK*** en cas d'exécution réussie de cette opération et ***TRI_Error*** dans le cas contraire. L'entité SA doit indiquer l'absence d'erreur dans le cas où la valeur d'un paramètre *in* ou une valeur de retour non définie est différente de néant.	

5.5.4.7 Opération triRaise (TE → SA)

Signature	`TriStatusType triRaise ( in TriComponentIdType componentId, in TriPortIdType tsiPortId, in TriAddressType SUTaddress, in TriSignatureIdType signatureId, in TriExceptionType exc)`
Paramètres In	`componentId` identificateur du composant de test qui propage l'exception `tsiPortId` identificateur du port de l'interface du système de test via lequel l'exception est envoyée à l'adaptateur du système sous test `SUTaddress` adresse de destination (facultative) figurant dans le système SUT `signatureId` identificateur de la signature de l'appel de procédure auquel l'exception est associée `exc` exception codée
Paramètres Out	s.o.
Valeur de retour	Etat de retour de l'opération `triRaise`. L'état de retour indique la réussite locale (***TRI_OK***) ou l'échec local (***TRI_Error***) de l'opération.
Contraintes	Cette opération est appelée par l'entité TE lorsqu'elle exécute une opération de propagation (`raise`) TTCN-3 en mode unidiffusion sur un port de composant qui a été mappé avec un port TSI. Cette opération est appelée par l'entité TE pour toutes les opérations de propagation TTCN-3 si aucun composant de système n'a été spécifié pour un test élémentaire, c'est-à-dire que seul un composant de test MTC est créé pour un test élémentaire. Le codage de l'exception a été effectué dans l'entité TE avant l'appel de cette opération TRI.
Effet	A l'invocation de cette opération, l'entité SA peut propager une exception pour un appel de procédure correspondant à l'identificateur de signature `signatureId` et au port TSI `tsiPortId`. L'opération `triRaise` retourne *TRI_OK* en cas d'exécution réussie de l'opération et *TRI_Error* dans le cas contraire.

5.5.4.8 Opération triRaiseBC (TE → SA)

Signature	`TriStatusType triRaiseBC ( in TriComponentIdType componentId, in TriPortIdType tsiPortId, in TriSignatureIdType signatureId, in TriExceptionType exc)`
Paramètres In	`componentId` identificateur du composant de test qui propage l'exception `tsiPortId` identificateur du port de l'interface du système de test via lequel l'exception est envoyé à l'adaptateur du système sous test `signatureId` identificateur de la signature de l'appel de procédure auquel l'exception est associée `exc` exception codée
Paramètres Out	s.o.
Valeur de retour	Etat de retour de l'opération `triRaise`. L'état de retour indique la réussite locale (***TRI_OK***) ou l'échec local (***TRI_Error***) de l'opération.
Contraintes	Cette opération est appelée par l'entité TE lorsqu'elle exécute une opération de propagation (`raise`) TTCN-3 en mode diffusion sur un port de composant qui a été mappé avec un port TSI. Cette opération est appelée par l'entité TE pour toutes les opérations de propagation TTCN-3 si aucun composant de système n'a été spécifié pour un test élémentaire, c'est-à-dire que seul un composant de test MTC est créé pour un test élémentaire. Le codage de l'exception a été effectué dans l'entité TE avant l'appel de cette opération TRI.
Effect	A l'invocation de cette opération, l'entité SA peut propager et diffuser une exception pour les appels de procédure correspondant à l'identificateur de signature `signatureId` et au port TSI `tsiPortId`. L'opération `triRaise` retourne *TRI_OK* en cas d'exécution réussie de l'opération et *TRI_Error* dans le cas contraire.

5.5.4.9 Opération triRaiseMC (TE → SA)

Signature	`TriStatusType triRaiseMC ( in TriComponentIdType componentId, in TriPortIdType tsiPortId, in TriAddressListType SUTaddresses, in TriSignatureIdType signatureId, in TriExceptionType exc)`	
Paramètres In	`componentId`	identificateur du composant de test qui propage l'exception
	`tsiPortId`	identificateur du port de l'interface du système de test via lequel l'exception est envoyée à l'adapteur du système sous test
	`SUTaddresses`	adresses de destination figurant dans le système SUT
	`signatureId`	identificateur de la signature de l'appel de procédure auquel l'exception est associée
	`exc`	exception codée
Paramètres Out	s.o.	
Valeur de retour	Etat de retour de l'opération `triRaise`. L'état de retour indique la réussite locale (***TRI_OK***) ou l'échec local (***TRI_Error***) de l'opération.	
Constraintes	Cette opération est appelée par l'entité TE lorsqu'elle exécute une opération de propagation (raise) TTCN-3 en mode multidiffusion sur un port de composant qui a été mappé avec un port TSI. Cette opération est appelée par l'entité TE pour toutes les opérations de propagation TTCN-3 si aucun composant de système n'a été spécifié pour un test élémentaire, c'est-à-dire que seul un composant de test MTC est créé pour un test élémentaire. Le codage de l'exception a été effectué dans l'entité TE avant l'appel de cette opération TRI.	
Effect	A l'invocation de cette opération, l'entité SA peut propager et multidiffuser une exception pour les appels de procédure correspondant à l'identificateur de signature `signatureId` et au port TSI `tsiPortId`. L'opération `triRaise` retourne *TRI_OK* en cas d'exécution réussie de l'opération et *TRI_Error* dans le cas contraire.	

5.5.4.10 Opération triEnqueueCall (SA → TE)

Signature	`void triEnqueueCall ( in TriPortIdType tsiPortId, in TriAddressType SUTaddress, in TriComponentIdType componentId, in TriSignatureIdType signatureId, in TriParameterListType parameterList)`	
Paramètres In	`tsiPortId`	identificateur du port de l'interface du système de test via lequel l'appel de procédure est mis en file d'attente par l'adaptateur du système sous test
	`SUTaddress`	adresse d'origine (facultative) figurant dans le système SUT
	`componentId`	identificateur du composant de test de réception
	`signatureId`	identificateur de la signature de l'appel de procédure
	`parameterList`	liste des paramètres codés qui font partie de la signature indiquée. Les paramètres figurant dans la liste `parameterList` sont classés dans l'ordre dans lequel ils apparaissent dans la déclaration de signature TTCN-3. Description des données transmises par l'entité appelante à l'entité appelée, sous forme de paramètres de l'opération
Paramètres Out	s.o.	
Valeur de retour	vide (void)	
Contraintes	Cette opération peut être appelée par l'entité SA après qu'elle a reçu un appel de procédure émanant du système SUT. Elle ne peut être utilisée que lorsque le port `tsiPortId` a été préalablement mappé avec un port du composant `componentId` ou indiqué dans l'instruction `triExecuteTestCase` précédente. Durant l'invocation d'une opération `triEnqueueCall`, tous les paramètres de procédure *in* et *inout* contiennent des valeurs codées.	
Effet	L'entité TE peut mettre en file d'attente cet appel de procédure avec l'identificateur de signature `signatureId` au niveau du port du composant `componentId` avec lequel le port TSI `tsiPortId` est mappé. Le décodage des paramètres de procédure a été effectué dans l'entité TE. L'entité TE doit indiquer l'absence d'erreur dans le cas où la valeur d'un paramètre *out* est différente de néant.	

5.5.4.11 Opération triEnqueueReply (SA → TE)

Signature	`void triEnqueueReply ( in TriPortIdType tsiPortId, in TriAddressType SUTaddress, in TriComponentIdType componentId, in TriSignatureIdType signatureId, in TriParameterListType parameterList, in TriParameterType returnValue)`
Paramètres In	`tsiPortId` identificateur du port de l'interface du système de test via lequel la réponse est mise en file d'attente par l'adaptateur du système sous test `SUTaddress` adresse d'origine (facultative) figurant dans le système SUT `componentId` identificateur du composant de test de réception `signatureId` identificateur de la signature de l'appel de procédure `parameterList` liste des paramètres codés qui font partie de la signature indiquée. Les paramètres figurant dans la liste `parameterList` sont classés dans l'ordre dans lequel ils apparaissent dans la déclaration de signature TTCN-3 `returnValue` valeur de retour codée (facultative) de l'appel de procédure
Paramètres Out	s.o.
Valeur de retour	vide (void)
Contraintes	Cette opération peut être appelée par l'entité SA après qu'elle a reçu une réponse du système SUT. Elle ne peut être utilisée que lorsque le port `tsiPortId` a été préalablement mappé avec un port du composant `componentId` ou indiqué dans l'instruction `triExecuteTestCase` précédente. Durant l'invocation d'une opération `triEnqueueReply`, tous les paramètres de procédure *out* et *inout* et la valeur de retour contiennent des valeurs codées. Si aucun type de retour n'a été défini pour la signature de la procédure dans la suite ATS TTCN-3, la valeur distincte `null` doit être utilisée comme valeur de retour.
Effet	L'entité TE peut mettre en file d'attente cette réponse à l'appel de procédure avec l'identificateur de signature `signatureId` au niveau du port du composant `componentId` avec lequel le port TSI `tsiPortId` est mappé. Le décodage des paramètres de procédure a été effectué dans l'entité TE. L'entité TE doit indiquer l'absence d'erreur dans le cas où la valeur d'un paramètre *in* ou une valeur de retour non définie est différente de néant.

5.5.4.12 Opération triEnqueueException (SA → TE)

Signature	`void triEnqueueException ( in TriPortIdType tsiPortId, in TriAddressType SUTaddress, in TriComponentIdType componentId, in TriSignatureIdType signatureId, in TriExceptionType exc)`
Paramètres In	`tsiPortId` identificateur du port de l'interface du système de test via lequel l'exception est mise en file d'attente par l'adaptateur du système sous test `SUTaddress` adresse d'origine (facultative) figurant dans le système SUT `componentId` identifier of the receiving test component `signatureId` identificateur de la signature de l'appel de procédure auquel l'exception est associée `exc` exception codée
Paramètres Out	s.o.
Valeur de retour	vide (void)
Contraintes	Cette opération peut être appelée par l'entité SA après qu'elle a reçu une réponse du système SUT. Elle ne peut être utilisée que lorsque le port `tsiPortId` a été préalablement mappé avec un port du composant `componentId` ou indiqué dans l'instruction `triExecuteTestCase` précédente. Durant l'invocation d'une opération `triEnqueueException`, l'exception `exception` doit contenir une valeur codée.
Effet	L'entité TE peut mettre en file d'attente cette exception pour l'appel de procédure avec l'identificateur de signature `signatureId` au niveau du port du composant `componentId` avec lequel le port TSI `tsiPortId` est mappé. Le décodage de l'exception doit être effectué dans l'entité TE.

5.5.5 Opérations diverses

5.5.5.1 Opération triSUTactionInformal (TE → SA)

Signature	`TriStatusType triSUTactionInformal (in string description)`
Paramètres In	`description` description informelle d'une action à mettre en œuvre sur le système SUT
Paramètres Out	s.o.
Valeur de retour	Etat de retour de l'opération `triSUTactionInformal`. L'état de retour indique la réussite locale (***TRI_OK***) ou l'échec local (***TRI_Error***) de l'opération.
Contraintes	Cette opération est appelée par l'entité TE lorsqu'elle exécute une opération-action TTCN-3 sur le système SUT, qui ne contient qu'une seule chaîne.
Effet	A l'invocation de cette opération, l'entité SA doit lancer les actions décrites à mettre en œuvre sur le système SUT, par exemple mise sous tension, initialisation ou envoi d'un message au système SUT. L'opération `triSUTactionInformal` retourne *TRI_OK* en cas d'exécution réussie de l'opération et *TRI_Error* dans le cas contraire. Il est à signaler que la valeur de retour de cette opération TRI ne donne aucune indication sur la réussite ou l'échec des actions à mettre en œuvre sur le système SUT.

5.5.5.2 Opération triSUTactionTemplate (TE → SA)

Obsolète.

5.6 Opérations d'interface de plate-forme

5.6.1 Opération triPAReset (TE → PA)

Signature	`TriStatusType triPAReset()`
Paramètres In	s.o.
Paramètres Out	s.o.
Valeur de retour	Etat de retour de l'opération `triPAReset`. L'état de retour indique la réussite locale (***TRI_OK***) ou l'échec local (***TRI_Error***) de l'opération.
Contraintes	Cette opération peut être appelée par l'entité TE à tout moment pour réinitialiser l'entité PA.
Effet	L'entité PA doit réinitialiser toutes les activités de synchronisation qu'elle est en train d'exécuter, par exemple arrêter tous les temporisateurs en cours, rejeter toute expiration de temporisation en instance de temporisateurs arrivés à expiration. L'opération `triResetSA` retourne *TRI_OK* dans le cas où l'opération a été exécutée avec succès et *TRI_Error* dans le cas contraire.

5.6.2 Opérations de temporisation

5.6.2.1 Opération triStartTimer (TE → PA)

Signature	`TriStatusType triStartTimer ( in TriTimerIdType timerId, in TriTimerDurationType timerDuration)`
Paramètres In	`timerId` identificateur de l'instance de temporisation `timerDuration` durée de la temporisation en secondes
Paramètres Out	s.o.
Valeur de retour	Etat de retour de l'opération `triStartTimer`. L'état de retour indique la réussite locale (***TRI_OK***) ou l'échec local (***TRI_Error***) de l'opération.
Constraintes	Cette opération est appelée par l'entité TE lorsqu'il est nécessaire de déclencher un temporisateur.
Effet	A l'invocation de cette opération, l'entité PA doit déclencher le temporisateur indiqué pour la durée indiquée. Le temporisateur fonctionne à partir de la valeur zéro (0.0) jusqu'à la valeur maximale indiquée par le paramètre `timerDuration`. Si le temporisateur indiqué par le paramètre `timerId` est déjà en cours, il convient de le redéclencher. Lorsque le temporisateur expire, l'entité PA appelle l'opération `triTimeout()` avec le paramètre `timerId`. L'opération `triStartTimer` retourne *TRI_OK* si le temporisateur a été déclenché avec succès et *TRI_Error* dans le cas contraire.

5.6.2.2 Opération triStopTimer (TE → PA)

Signature	`TriStatusType triStopTimer ( in TriTimerIdType timerId)`
Paramètres In	`timerId` identificateur de l'instance de temporisation
Paramètres Out	s.o.
Valeur de retour	Etat de retour de l'opération `triStopTimer`. L'état de retour indique la réussite locale (***TRI_OK***) ou l'échec local (***TRI_Error***) de l'opération.
Constraintes	Cette opération est appelée par l'entité TE lorsqu'un temporisateur doit être arrêté.
Effet	A l'invocation de cette opération, l'entité PA doit utiliser le paramètre `timerId` pour mettre fin à l'instance de temporisation indiquée. L'arrêt d'un temporisateur inactif, c'est-à-dire un temporisateur qui n'a pas été déclenché ou qui est déjà arrivé à expiration, ne devrait avoir aucun effet. L'opération `triStopTimer` retourne ***TRI_OK*** si l'opération a été exécutée avec succès et ***TRI_Error*** dans le cas contraire. Il est à signaler que l'arrêt d'un temporisateur inactif est une opération valide. Dans ce cas, ***TRI_OK*** doit être retourné.

5.6.2.3 Opération triReadTimer (TE → PA)

Signature	`TriStatusType triReadTimer ( in TriTimerIdType timerId,` `out TriTimerDurationType elapsedTime)`
Paramètres In	`timerId` identificateur de l'instance de temporisation
Paramètres Out	`elapsedTime` valeur du temps écoulé depuis que le temporisateur a été déclenché (en secondes)
Valeur de retour	Etat de retour de l'opération `triReadTimer`. L'état de retour indique la réussite locale (***TRI_OK***) ou l'échec local (***TRI_Error***) de l'opération.
Constraintes	Cette opération peut être appelée par l'entité TE lorsqu'une opération de lecture de temporisateur TTCN-3 doit être exécutée sur le temporisateur indiqué (voir § 5.3.1).
Effet	A l'invocation de cette opération, l'entité PA doit utiliser le paramètre `timerId` pour avoir accès à la durée qui s'est écoulée depuis que ce temporisateur a été déclenché. La valeur de retour `elapsedTime` doit être exprimée en secondes. La lecture d'un temporisateur inactif, c'est-à-dire un temporisateur qui n'a pas été déclenché ou qui est déjà arrivé à expiration, doit retourner une valeur de durée écoulée de zéro. L'opération `triReadTimer` retourne ***TRI_OK*** si l'opération a été exécutée avec succès et ***TRI_Error*** dans le cas contraire.

5.6.2.4 Opération triTimerRunning (TE → PA)

Signature	`TriStatusType triTimerRunning ( in TriTimerIdType timerId,` `out boolean running)`
Paramètres In	`timerId` identificateur de l'instance de temporisation
Paramètres Out	`running` état du temporisateur
Valeur de retour	Etat de retour de l'opération `triTimerRunning`. L'état de retour indique la réussite locale (***TRI_OK***) ou l'échec local (***TRI_Error***) de l'opération.
Contraintes	Cette opération peut être appelée par l'entité TE lorsqu'une opération de temporisateur en cours TTCN-3 doit être exécutée sur le temporisateur indiqué (voir § 5.3.1).
Effet	A l'invocation de cette opération, l'entité PA doit utiliser le paramètre `timerId` pour avoir accès à l'état du temporisateur. L'opération met le paramètre `running` à la valeur booléenne `true` si et seulement si le temporisateur est actuellement en cours. L'opération `triTimerRunning` retourne ***TRI_OK*** si l'état du temporisateur a été déterminé avec succès et ***TRI_Error*** dans le cas contraire.

5.6.2.5 Opération triTimeout (PA → TE)

Signature	`void triTimeout ( in TriTimerIdType timerId)`
Paramètres In	`timerId` identificateur de l'instance de temporisation
Paramètres Out	s.o.
Valeur de retour	vide (void)
Contraintes	Cette opération est appelée par l'entité PA après qu'un temporisateur, qui a été préalablement déclenché au moyen de l'opération `triStartTimer`, est arrivé à expiration, c'est-à-dire qu'il a atteint sa valeur de durée maximale.
Effet	La fin de temporisation associée au paramètre `timerId` peut être ajoutée à la liste des fins de temporisation dans l'entité TE. L'implémentation de cette opération dans l'entité TE doit être effectuée de manière qu'elle prenne en considération les différentes sémantiques TTCN-3 pour les temporisateurs définis dans la Rec. UIT-T Z.143 [4] (voir aussi le § 5.3.1).

5.6.3 Opérations diverses

5.6.3.1 Opération triExternalFunction (TE → PA)

Signature	`TriStatusType triExternalFunction ( in TriFunctionIdType functionId, inout TriParameterListTypeparameterList, out TriParameterType returnValue)`	
Paramètres In	`functionId`	identificateur de la fonction externe
Paramètres Out	`returnValue`	valeur de retour codée (facultative)
Paramètres Inout	`parameterList`	liste de paramètres codés pour la fonction indiquée. Les paramètres figurant dans la liste `parameterList` sont classés dans l'ordre dans lequel ils apparaissent dans la déclaration de la fonction TTCN-3.
Valeur de retour	Etat retour de l'opération triExternalFunction. L'état de retour indique la réussite locale (***TRI_OK***) ou l'échec local (***TRI_Error***) de l'opération.	
Contraintes	Cette opération est appelée par l'entité TE lorsqu'elle exécute une fonction qui est définie comme étant une fonction externe TTCN-3 (c'est-à-dire que toutes les fonctions non externes sont implémentées dans l'entité TE). Durant l'invocation d'une opération `triExternalFunction` par l'entité TE, tous les paramètres de fonction *in* et *inout* contiennent des valeurs codées. Une indication d'absence d'erreur doit être donnée par l'entité PA dans le cas où la valeur d'un paramètre out est différente de néant.	
Effet	Pour chaque fonction externe spécifiée dans la suite ATS TTCN-3, l'entité PA doit implémenter le comportement. A l'invocation de cette opération, l'entité PA doit invoquer la fonction indiquée par l'identificateur `functionId`. Elle doit accéder aux paramètres de fonction *in* et *inout* spécifiés dans la liste `parameterList`, évaluer la fonction externe au moyen des valeurs de ces paramètres, et calculer les valeurs des Paramètres *Inout* et *out* figurant dans la liste `parameterList`. L'opération doit ensuite retourner les valeurs codées de tous les paramètres de fonction *inout* et *out* ainsi que la valeur de retour codée de la fonction externe. Si aucun type de retour n'a été défini pour cette fonction externe dans la suite ATS TTCN-3, la valeur distincte `null` doit être utilisée. L'opération `triExternalFunction` retourne ***TRI_OK*** si l'entité PA mène à bien l'évaluation de la fonction externe et ***TRI_Error*** dans le cas contraire. Il est à noter que, alors que toutes les autres opérations TRI sont considérées comme étant non bloquantes, l'opération `triExternalFunction` est considérée comme étant *bloquante*. Cela signifie que l'opération ne doit pas envoyer de valeur de retour avant que la fonction externe indiquée ait été entièrement évaluée. Les fonctions externes doivent être implémentées avec soin du fait qu'elles pourraient entraîner un blocage de l'exécution des composants de test, voire de l'implémentation du système de test en son entier.	

6 Mappage vers le langage Java

6.1 Introduction

Le présent paragraphe donne un aperçu du mappage vers le langage Java pour l'interface TRI. Pour des raisons d'efficacité, on utilise ici un mappage de langage spécialisé et non un mappage du langage de définition d'interface (IDL) du groupe de gestion d'objets (OMG) vers le langage Java.

Le mappage vers le langage Java pour l'interface d'exécution TTCN-3 (TRI) indique comment les définitions IDL énoncées au paragraphe 5 sont mappées vers le langage Java. Ce mappage est indépendant de la version Java utilisée, seules les structures du langage Java de base étant utilisées.

6.2 Noms et portées

6.2.1 Noms

Bien qu'il n'y ait pas d'incompatibilités entre les identificateurs utilisés dans la définition IDL et en langage Java, certaines règles de traduction de nom sont appliquées aux identificateurs IDL.

- Les identificateurs de paramètre en Java doivent commencer par une minuscule, l'élément ou les éléments suivants qui forment l'identificateur de paramètre devant commencer par une majuscule. Par exemple, l'identificateur de paramètre IDL **`SUTaddress`** correspond à `sutAddress` en Java.
- Les interfaces ou les identificateurs de classe en Java omettent l'élément `Type` de droite utilisé dans la définition IDL. Par exemple, le type IDL **`TriPortIdType`** correspond à `TriPortId` en Java.

Le mappage obtenu est conforme aux conventions de codage Java normalisées.

6.2.2 Portées

Le module IDL **`triInterface`** est mappé sur le paquetage Java `org.etsi.ttcn3.tri`. Toutes les déclarations de type IDL figurant dans ce module sont mappées sur les classes ou déclarations d'interface Java de ce paquetage.

6.3 Mappage des types

6.3.1 Mappage des types de base

Le Tableau 3 donne un aperçu général de la manière dont les types IDL de base utilisés sont mappés sur les types Java.

Tableau 3/Z.144 – Mappage des types de base

Type IDL	Type Java
`boolean`	`org.etsi.ttcn.tri.TriBoolean`
`string`	`java.lang.String`

Les autres types IDL de base ne sont pas utilisés dans la définition IDL.

6.3.1.1 Type boolean (booléen)

Le type **`boolean`** (`booléen`) IDL est mappé sur l'interface `org.etsi.ttcn.tri.TriBoolean`, de manière que les objets qui implémentent cette interface puissent faire office de contenants.

L'interface suivante est définie pour `org.etsi.ttcn.tri.TriBoolean`:

```
// TriBoolean
package org.etsi.ttcn.tri;
public interface TriBoolean {
    public void setBooleanValue(boolean value);
    public boolean getBooleanValue();
}
```

6.3.1.1.1 Méthodes

`- setBooleanValue(boolean value)`
Met ce type `TriBoolean` à la valeur booléenne `value`.

`- getBooleanValue()`
Retourne la valeur booléenne correspondant à ce type `TriBoolean`.

6.3.1.2 Type string (chaîne)

Le type **`string`** IDL est mappé sur la classe `java.lang.String` sans contrôle de portée ni limites pour les caractères de la chaîne. Toutes les chaînes possibles définies en notation TTCN-3 peuvent être converties en chaînes de la classe `java.lang.String`.

6.3.2 Mappage des types structurés

Dans la description IDL de l'interface TRI, les types définis par l'utilisateur sont qualifiés de types natifs. Dans le mappage vers le langage Java, ces types sont mappés sur les interfaces Java. Les interfaces définissent les méthodes et les attributs que peuvent utiliser les objets implémentant cette interface.

6.3.2.1 Type TriPortIdType

Le type **`TriPortIdType`** est mappé sur l'interface suivante:

```
// TRI IDL TriPortIdType
package org.etsi.ttcn.tri;
public interface TriPortId {
    public String getPortName();
    public TriComponentId getComponent();
    public boolean isArray();
    public int getPortIndex();
}
```

6.3.2.1.1 Méthodes

- `getPortName()`
  Retourne le nom du port indiqué dans la spécification TTCN-3.

- `getComponent()`
  Retourne l'identificateur de composant auquel cet identificateur `TriPortId` appartient, comme indiqué dans la spécification TTCN-3.

- `isArray()`
  Retourne `true` si le port considéré fait partie d'une matrice de ports et `false` dans le cas contraire.

- `getPortIndex()`
  Retourne l'indice de port si le port considéré fait partie d'une matrice de ports commençant à zéro. Si le port ne fait pas partie d'une matrice de ports, la valeur –1 est retournée.

6.3.2.2 Type TriPortIdListType

Le type **`TriPortIdListType`** est mappé sur l'interface suivante:

```
// TRI IDL TriPortIdListType
package org.etsi.ttcn.tri;
public interface TriPortIdList {
    public int size();
    public boolean isEmpty();
    public java.util.Enumeration getPortIds();
    public TriPortId get(int index);
}
```

6.3.2.2.1 Méthodes

`size()`
Retourne le nombre de ports figurant dans cette liste.

`isEmpty()`
Retourne `true` si cette liste ne contient pas de port..

`getPortIds()`
Retourne une énumération `Enumeration` de tous les ports figurant dans la liste. L'énumération indique les ports dans l'ordre dans lequel ils apparaissent dans la liste.

`get(int index)`
Retourne l'identificateur `TriPortId` à la position spécifiée.

6.3.2.3 Type TriComponentIdType

Le type **`TriComponentIdType`** est mappé sur l'interface suivante:

```
// TRI IDL TriComponentIdType
package org.etsi.ttcn.tri;
public interface TriComponentId {
    public String getComponentId();
    public String getComponentName();
    public String getComponentTypeName();
    public TriPortIdList getPortList();
    public boolean equals(TriComponentId port);
}
```

6.3.2.3.1 Méthodes

- `getComponentId()`
  Retourne une représentation de cet identificateur de composant unique.

- `getComponentName()`
  Retourne le nom du composant défini dans la spécification TTCN-3. Si celle-ci ne définit aucun nom, une chaîne vide est retournée.

- `getComponentTypeName()`
  Retourne le nom du type de composant défini dans la spécification TTCN-3.

- `getPortList()`
  Retourne la liste des ports du composant définie dans la spécification TTCN-3.

- equals(TriComponentId component)
  Compare le composant component avec l'identificateur TriComponentId pour déterminer s'ils sont égaux. Retourne true si et seulement si les deux composants ont la même représentation de l'identificateur de composant unique, et false dans le cas contraire.

6.3.2.4 Type TriComponentIdListType

Le type **TriComponentIdListType** est mappé sur l'interface suivante:

```
// TRI IDL TriComponentIdListType
package org.etsi.ttcn.tri;
public interface TriComponentIdListType {
    public int size();
    public boolean isEmpty();
    public java.util.Enumeration getComponents();
    public TriComponentId get(int index);
    public void clear();
    public void add(TriComponentId comp);
}
```

6.3.2.4.1 Méthodes

size()
  Retourne le nombre de composants figurant dans cette liste.

isEmpty()
  Retourne true si cette liste ne contient pas de composants.

getComponents()
  Retourne une énumération Enumeration de tous les composants figurant dans la liste. L'énumération indique les composants dans l'ordre dans lequel ils apparaissent dans la liste.

get(int index)
  Retourne l'identificateur TriComponentId à la position spécifiée.

clear()
  Supprime tous les composants de cette liste TriComponentIdList.

add(TriComponentId comp)
  Ajoute Comp à la fin de cette liste TriComponentIdList.

6.3.2.5 Type TriMessageType

Le type **TriMessageType** est mappé sur l'interface suivante:

```
// TRI IDL TriMessageType
package org.etsi.ttcn.tri;
public interface TriMessage {
    public byte[] getEncodedMessage();
    public void setEncodedMessage(byte[] message);
    public boolean equals(TriMessage message);
}
```

6.3.2.5.1 Méthodes

- getEncodedMessage()
  Retourne le message codé selon les règles de codage définies dans la spécification TTCN-3.

- setEncodedMessage(byte[] message)
  Positionne la représentation codée du message TriMessage to message.

- equals(TriMessage message)
  Compare le message message avec le message TriMessage pour déterminer s'ils sont égaux. Retourne true si et seulement si ces deux messages ont la même représentation codée, et false dans le cas contraire.

6.3.2.6 Type TriAddressType

Le type **TriAddressType** est mappé sur l'interface suivante:

```
// TRI IDL TriAddressType
package org.etsi.ttcn.tri;
public interface TriAddress {
```

```
    public byte[] getEncodedAddress();
    public void setEncodedAddress(byte[] address);
    public boolean equals(TriAddress address);
}
```

6.3.2.6.1 Méthodes

- `getEncodedAddress()`
  Retourne l'adresse codée.

- `setEncodedAddress(byte[] address)`
  Positionne la représentation codée de l'adresse `TriAddress` sur `address`.

- `equals(TriAddress address)`
  Compare l'adresse `address` avec l'adresse `TriAddress` f pour déterminer si elles sont égales. Retourne `true` si et seulement si ces deux adresses ont la même représentation codée, et `false` dans le cas contraire.

6.3.2.7 Type TriAddressListType

Le type **`TriAddressListType`** est mappé sur l'interface suivante:

```
// TRI IDL TriAddressListType
package org.etsi.ttcn.tri;
public interface TriAddressListType {
    public int size();
    public boolean isEmpty();
    public java.util.Enumeration getAddresses();
    public TriAddress get(int index);
    public void clear();
    public void add(TriAddress addr);
}
```

6.3.2.7.1 Méthodes

`size()`
  Retourne le nombre de composants figurant dans cette liste.

`isEmpty()`
  Retourne `true` si cette liste ne contient pas de composants.

`getAddresses()`
  Retourne une énumération `Enumeration` de tous les composants figurant dans la liste. L'énumération indique les adresses dans l'ordre dans lequel elles apparaissent dans la liste.

`get(int index)`
  Retourne l'adresse `TriAddress` à la position spécifiée.

`clear()`
  Supprime toutes les adresses de la liste `TriAddressList`.

`add(TriAddress addr)`
  Ajoute `addr` à la fin de la liste `TriAddressList`.

6.3.2.8 Type TriSignatureIdType

Le type **`TriSignatureIdType`** est mappé sur l'interface suivante:

```
// TRI IDL TriSignatureIdType
package org.etsi.ttcn.tri;
public interface TriSignatureId {
    public String getSignatureName();
    public void setSignatureName(String sigName);
    public boolean equals(TriSignatureId sig);
}
```

6.3.2.8.1 Méthodes

- `getSignatureName()`
  Retourne l'identificateur de signature défini dans la spécification TTCN-3.

- `setSignatureName(String sigName)`
  Positionne l'identificateur de signature `TriSignatureId` sur `sigName`.

- `equals(TriSignatureId sig)`
  Compare `sig` avec l'identificateur `TriSignatureId` pour déterminer s'ils sont égaux. Retourne `true` si et seulement si les deux signatures ont le même identificateur de signature, et `false` dans le cas contraire.

6.3.2.9 Type TriParameterType

Le type **`TriParameterType`** est mappé sur l'interface suivante:

```
// TRI IDL TriParameterType
package org.etsi.ttcn.tri;
public interface TriParameter {
    public String getParameterName();
    public void setParameterName(String name);
    public int getParameterPassingMode();
    public void setParameterPassingMode(in mode);
    public byte[] getEncodedParameter();
    public void setEncodedParameter(byte[] parameter);
}
```

6.3.2.9.1 Méthodes

- `getParameterName()`
  Retourne le nom du paramètre défini dans la spécification TTCN-3.
- `setParameterName(String name)`
  Positionne le nom du paramètre `TriParameter` sur `name`.
- `getParameterPassingMode()`
  Retourne le mode de transmission du paramètre.
- `setParameterPassingMode(in mode)`
  Positionne le mode du paramètre `TriParameter` sur `mode`.
- `getEncodedParameter()`
  Retourne la représentation codée du paramètre `TriParameter`, ou l'objet `null` si le paramètre contient la valeur distincte `null` (voir aussi le § 5.5.4.1).
- `setEncodedParameter(byte[] parameter)`
  Positionne la représentation codée du paramètre `TriParameter` sur `parameter`. Si cette représentation codée doit être positionnée sur la valeur distincte `null` pour indiquer que le paramètre ne contient aucune valeur, la valeur `null` Java doit être transmise comme paramètre `parameter` (voir aussi le § 5.5.4.1).

6.3.2.10 Type TriParameterPassingModeType

Le type **`TriParameterPassingModeType`** est mappé sur l'interface suivante:

```
// TRI IDL TriParameterPassingModeType
package org.etsi.ttcn.tri;
public interface TriParameterPassingMode {
    public final static int TRI_IN = 0;
    public final static int TRI_INOUT = 1;
    public final static int TRI_OUT = 2;
}
```

6.3.2.10.1 Constantes

- `TRI_IN`
  Sera utilisée pour indiquer qu'un paramètre `TriParameter` est un paramètre `in`.
- `TRI_INOUT`
  Sera utilisée pour indiquer qu'un paramètre `TriParameter` est un paramètre `inout`.
- `TRI_OUT`
  Sera utilisée pour indiquer qu'un paramètre `TriParameter` est un paramètre `out`.

6.3.2.11 Type TriParameterListType

Le type **`TriParameterListType`** est mappé sur l'interface suivante:

```
// TRI IDL TriParameterListType
package org.etsi.ttcn.tri;
public interface TriParameterList {
    public int size();
    public boolean isEmpty();
    public java.util.Enumeration getParameters();
    public TriParameter get(int index);
    public void clear();
    public void add(TriParameter parameter);
}
```

6.3.2.11.1 Méthodes

`size()`
Retourne le nombre de paramètres figurant dans cette liste.

`isEmpty()`
Retourne `true` si cette liste ne contient pas de paramètres.

`getParameters()`
Retourne une énumération `Enumeration` de tous les paramètres figurant dans la liste. L'énumération indique les paramètres dans l'ordre dans lequel ils apparaissent dans la liste.

`get(int index)`
Retourne le paramètre `TriParameter` à la position spécifiée.

`clear()`
Supprime tous les paramètres de la liste `TriParameterList`.

`add(TriParameter parameter)`
Ajoute `parameter` à la fin de la liste `TriParameterList`.

6.3.2.12 Type TriExceptionType

Le type **`TriExceptionType`** est mappé sur l'interface suivante:

```
// TRI IDL TriExceptionType
package org.etsi.ttcn.tri;
public interface TriException {
    public byte[] getEncodedException();
    public void setEncodedException(byte[] message);
    public boolean equals(TriException exc);
}
```

6.3.2.12.1 Méthodes

`- getEncodedException()`
Retourne l'exception codée selon les règles de codage définies dans la spécification TTCN-3.

`- setEncodedMessage(byte[] exc)`
Positionne la représentation codée de l'exception `TriException` sur `exc`.

`- equals(TriException exc)`
Compare `exc` avec l'exception `TriException` pour déterminer si elles sont égales. Retourne `true` si et seulement si les deux exceptions ont la même représentation codée, et `false` dans le cas contraire.

6.3.2.13 Type TriTimerIdType

Le type **`TriTimerIdType`** est mappé sur l'interface suivante:

```
// TRI IDL TriTimerIdType
package org.etsi.ttcn.tri;
public interface TriTimerId {
    public String getTimerName();
    public boolean equals(TriTimerId timer);
}
```

6.3.2.13.1 Méthodes

`- getTimerName()`
Retourne le nom de l'identificateur de temporisateur défini dans la spécification TTCN-3. Dans le cas de temporisateurs implicites, le résultat dépend de l'implémentation (voir le § 4.1.2)

`- equals(TriTimerId timer)`
Compare le temporisateur `timer` avec l'identificateur `TriTimerId` pour déterminer s'ils sont égaux. Retourne `true` si et seulement si les deux identificateurs de temporisateur correspondent au même temporisateur, et `false` dans le cas contraire.

6.3.2.14 Type TriTimerDurationType

Le type **`TriTimerDurationType`** est mappé sur l'interface suivante:

```
// TRI IDL TriTimerDurationType
package org.etsi.ttcn.tri;
public interface TriTimerDuration {
    public double getDuration();
    public void setDuration(double duration);
    public boolean equals(TriTimerDuration duration);
}
```

6.3.2.14.1 Méthodes

- `getDuration()`
  Retourne `double` pour la durée d'un temporisateur.
- `setDuration(double duration)`
  Positionne la durée `TriTimerDuration` sur `duration`.
- `equals(TriTimerDuration duration)`
  Compare `duration` avec la durée `TriTimerDuration` pour déterminer si elles sont égales. Retourne `true` si et seulement si les deux durées sont les mêmes, et `false` dans le cas contraire.

6.3.2.15 Type TriFunctionIdType

Le type **`TriFunctionIdType`** est mappé sur l'interface suivante:

```
// TRI IDL TriFunctionIdType
package org.etsi.ttcn.tri;
public interface TriFunctionId {
    public String toString();
    public String getFunctionName();
    public boolean equals(TriFunctionId fun);
}
```

6.3.2.15.1 Méthodes

- `toString()`
  Retourne la représentation sous forme de chaîne de la fonction définie dans la spécification TTCN-3.
- `getFunctionName()`
  Retourne l'identificateur de fonction défini dans la spécification TTCN-3.
- `equals(TriFunctionId fun)`
  Compare `fun` avec l'identificateur `TriFunctionId` pour déterminer si ces deux fonctions sont égales. Retourne `true` si et seulement si les deux fonctions ont le même identificateur, et `false` dans le cas contraire.

6.3.2.16 Type TriTestCaseIdType

Le type **`TriTestCaseIdType`** est mappé sur l'interface suivante:

```
// TRI IDL TriTestCaseIdType
package org.etsi.ttcn.tri;
public interface TriTestCaseId {
    public String toString();
    public String getTestCaseName();
    public boolean equals(TriTestCaseId tc);
}
```

6.3.2.16.1 Méthodes

- `toString()`
  Retourne la représentation sous forme de chaîne du test élémentaire défini dans la spécification TTCN-3.
- `getTestCaseName()`
  Retourne l'identificateur du test élémentaire défini dans la spécification TTCN-3.
- `equals(TriTestCaseId tc)`
  Compare `tc` avec l'identificateur `TriTestCaseId` pour déterminer s'ils sont égaux. Retourne `true` si et seulement si les deux tests élémentaires ont le même identificateur, et `false` dans le cas contraire.

6.3.2.17 Type TriActionTemplateType

Obsolète.

6.3.2.18 Type TriStatusType

Le type **`TriStatusType`** est mappé sur l'interface suivante:

```
// TriStatusType
package org.etsi.ttcn.tri;
public interface TriStatus {
    public final static int TRI_OK = 0;
    public final static int TRI_ERROR = -1;
    public String toString();
    public int getStatus();
```

```
    public void setStatus(int status);
    public boolean equals(TriStatus status);
}
```

6.3.2.18.1 Méthodes

- `toString()`
  Retourne la représentation sous forme de chaîne de l'état.
- `getStatus()`
  Retourne l'état `TriStatus`.
- `setStatus(int status)`
  Positionne l'état `TriStatus`.
- `equals(TriStatus status)`
  Compare l'état `status` avec l'état `TriStatus` pour déterminer s'ils sont égaux. Retourne `true` si et seulement si ces deux états sont identiques, et `false` dans le cas contraire.

6.4 Constantes

Dans ce mappage vers le langage Java, des constances ont été spécifiées. Toutes les constantes sont définies comme étant `public final static` et sont accessibles depuis chaque objet de chaque paquetage. Les constantes définies dans le présent paragraphe ne sont pas définies par rapport au paragraphe relatif au langage IDL mais elles découlent de la spécification des types IDL TRI marqués comme étant natifs.

Les constantes suivantes peuvent être utilisées pour déterminer le mode de transmission des paramètres TTCN-3 (voir aussi le § 6.3.2.10).

- org.etsi.ttcn.tri.TriParameterPassingMode.TRI_IN;
- org.etsi.ttcn.tri.TriParameterPassingMode.TRI_INOUT;
- org.etsi.ttcn.tri.TriParameterPassingMode.TRI_OUT.

Les valeurs des instances de ces constantes doivent refléter le mode de transmission des paramètres définis dans les signatures de procédure TTCN-3.

Pour la valeur de paramètre distincte `null`, la valeur de paramètre codée doit être mise à la valeur `null` Java.

Les constantes suivantes doivent être utilisées pour indiquer la réussite locale d'une méthode (voir aussi le § 6.3.2.18):

- org.etsi.ttcn.tri.TriStatus.TRI_OK;
- org.etsi.ttcn.tri.TriStatus.TRI_ERROR.

6.5 Mappage d'interfaces

La définition IDL TRI définit deux interfaces: l'interface **`triCommunication`** et l'interface **`triPlatform`**. Les opérations sont définies pour un sens particulier de cette interface, c'est-à-dire que certaines opérations ne peuvent être appelées que par l'exécutable TTCN-3 (TE) sur l'adaptateur du système sous test (SA) alors que d'autres opérations ne peuvent être appelées que par l'entité SA sur l'entité TE. Cette situation se traduit par la subdivision des interfaces IDL TRI en deux sous-interfaces, chacune suivie du suffixe de l'entité appelée.

Tableau 4/Z.144 – Sous-interfaces

Entité appelante/ entité appelée	TE	SA	PA
TE	-	`TriCommunicationSA`	`triPlatformPA`
SA	`TriCommunicationTE`	-	-
PA	`TriPlatformTE`	-	-

Toutes les méthodes définies dans ces interfaces devraient agir comme indiqué au § 5.

6.5.1 Mode de transmission des Paramètres Out et inout

Les types IDL suivants sont utilisés dans le mode de transmission des Paramètres `Out` ou `inout`:

- TriParameter.
- TriParameterList.

- TriBoolean.
- TriTimerDuration.

Dans le cas où ils sont utilisés dans le mode de transmission des Paramètres Out ou `inout`, les objets de la classe considérée seront transmis avec l'appel de la méthode. L'entité appelée aura alors accès à des méthodes permettant de fixer les valeurs de retour.

6.5.2 Interface triCommunication

L'interface **`triCommunication`** est subdivisée en deux sous-interfaces: l'interface `triCommunicationSA`, qui définit les appels émanant de l'entité TE à destination de l'entité SA, et l'interface `triCommunicationTE`, qui définit les appels émanant de l'entité SA à destination de l'entité TE.

6.5.2.1 Interface triCommunicationSA

L'interface `triCommunicationSA` est mappée sur l'interface suivante:

```
// TriCommunication
// TE -> SA
package org.etsi.ttcn.tri;
public interface TriCommunicationSA {
    // Reset Operation
    // Réf: TRI-Définition 5.5.1
    TriStatus triSAReset();

    // Opérations de traitement de connexion
    // Réf: TRI-Définition 5.5.2.1
    public TriStatus triExecuteTestCase(TriTestCaseId
        testCaseId,TriPortIdList tsiPorts);
    // Réf: TRI-Définition 5.5.2.2
    public TriStatus triMap(TriPortId compPortId, TriPortId tsiPortId);
    // Réf: TRI-Définition 5.5.2.3
    public TriStatus triUnmap(TriPortId compPortId, TriPortId tsiPortId);

    // Opérations de communication en mode message
    // Réf: TRI-Définition 5.5.3.1
    public TriStatus triSend(TriComponentId componentId, TriPortId tsiPortId,
            TriAddress sutAddress, TriMessage sendMessage);
    // Réf: TRI-Définition 5.5.3.2
    public TriStatus triSendBC(TriComponentId componentId, TriPortId tsiPortId,
            TriMessage sendMessage);
    // Réf: TRI-Définition 5.5.3.3
    public TriStatus triSendMC(TriComponentId componentId, TriPortId tsiPortId,
            TriAddressList addresses, TriMessage sendMessage);

    // Opérations de communication en mode procédure
    // Réf: TRI-Définition 5.5.4.1
    public TriStatus triCall(TriComponentId componentId,
            TriPortId tsiPortId, TriAddress sutAddress,
            TriSignatureId signatureId, TriParameterList parameterList);
    // Réf: TRI-Définition 5.5.4.2
    public TriStatus triCallBC(TriComponentId componentId,
            TriPortId tsiPortId,
            TriSignatureId signatureId, TriParameterList parameterList);
    // Réf: TRI-Définition 5.5.4.3
    public TriStatus triCallMC(TriComponentId componentId,
            TriPortId tsiPortId, TriAddressList sutAddresses,
            TriSignatureId signatureId, TriParameterList parameterList);

    // Réf: TRI-Définition 5.5.4.4
    public TriStatus triReply(TriComponentId componentId,
            TriPortId tsiPortId, TriAddress sutAddress,
            TriSignatureId signatureId, TriParameterList parameterList,
            TriParameter returnValue);
    // Réf: TRI-Définition 5.5.4.5
    public TriStatus triReplyBC(TriComponentId componentId,
            TriPortId tsiPortId,
            TriSignatureId signatureId, TriParameterList parameterList,
            TriParameter returnValue);
    // Réf: TRI-Définition 5.5.4.6
    public TriStatus triReplyMC(TriComponentId componentId,
            TriPortId tsiPortId, TriAddressList sutAddresses,
            TriSignatureId signatureId, TriParameterList parameterList,
            TriParameter returnValue);

    // Réf: TRI-Définition 5.5.4.7
    public TriStatus triRaise(TriComponentId componentId, TriPortId tsiPortId,
```

```
            TriAddress sutAddress,
            TriSignatureId signatureId,
            TriException exc);
    // Réf: TRI-Définition 5.5.4.8
    public TriStatus triRaiseBC(TriComponentId componentId, TriPortId tsiPortId,
            TriSignatureId signatureId,
            TriException exc);
    // Réf: TRI-Définition 5.5.4.9
    public TriStatus triRaiseMC(TriComponentId componentId, TriPortId tsiPortId,
            TriAddresses sutAddresses,
            TriSignatureId signatureId,
            TriException exc);

    // Opérations diverses
    // Réf: TRI-Définition 5.5.5.1
    public TriStatus triSutActionInformal(String description);

}
```

6.5.2.2 Interface triCommunicationTE

L'interface `triCommunicationTE` est mappée sur l'interface suivante:

```
// TriCommunication
// SA -> TE
package org.etsi.ttcn.tri;
public interface TriCommunicationTE {
    // Opérations de communication en mode message
    // Réf: TRI-Définition 5.5.3.4
    public void triEnqueueMsg(TriPortId tsiPortId,
            TriAddress sutAddress, TriComponentId componentId,
            TriMessage receivedMessage);

    // Opérations de communication en mode procédure
    // Réf: TRI-Définition 5.5.4.10
    public void triEnqueueCall(TriPortId tsiPortId,
            TriAddress SUTaddress, TriComponentId componentId,
            TriSignatureId signatureId, TriParameterList parameterList );

    // Réf: TRI-Définition 5.5.4.11
    public void triEnqueueReply(TriPortId tsiPortId, TriAddress sutAddress,
            TriComponentId componentId, TriSignatureId signatureId,
            TriParameterList parameterList, TriParameter returnValue);

    // Réf: TRI-Définition 5.5.4.12
    public void triEnqueueException(TriPortId tsiPortId,
            TriAddress sutAddress, TriComponentId componentId,
        TriSignatureId signatureId, TriException exc);
}
```

6.5.3 Interface triPlatform

L'interface **`triPlatform`** est subdivisée en deux sous-interfaces: l'interface `triPlatformPA`, qui définit les appels émanant de l'entité TE à destination de l'entité PA, et l'interface `triPlatformTE`, qui définit les appels émanant de l'entité PA à destination de l'entité TE.

6.5.3.1 Interface TriPlatformPA

L'interface `triPlatformPA` est mappée sur l'interface suivante:

```
// TriPlatform
// TE -> PA
package org.etsi.ttcn.tri;
public interface TriPlatformPA {
    // Réf: TRI-Définition 5.6.1
    public TriStatus triPAReset();

    // Opérations de temporisation
    // Réf: TRI-Définition 5.6.2.1
    public TriStatus triStartTimer(TriTimerId timerId,
        TriTimerDuration timerDuration);

    // Réf: TRI-Définition 5.6.2.2
    public TriStatus triStopTimer(TriTimerId timerId);

    // Réf: TRI-Définition 5.6.2.3
    public TriStatus triReadTimer(TriTimerId timerId,
        TriTimerDuration elapsedTime);
```

```
    // Réf: TRI-Définition 5.6.2.4
    public TriStatus triTimerRunning(TriTimerId timerId,
        TriBoolean running);

    // Opérations diverses

    // Réf: TRI-Définition 5.6.3.1
    public TriStatus triExternalFunction(TriFunctionId functionId,
     TriParameterList parameterList, TriParameter returnValue);
}
```

6.5.3.2 Interface TriPlatformTE

L'interface `triPlatformTE` est mappée sur l'interface Java suivante:

```
// TriPlatform
// PA -> TE
package org.etsi.ttcn.tri;
public interface TriPlatformTE {
    // Réf: TRI-Définition 5.6.2.5
    public void triTimeout(TriTimerId timerId);
}
```

6.6 Paramètres facultatifs

Comme indiqué au paragraphe 5.4, une valeur réservée doit être utilisée pour indiquer l'absence d'un paramètre facultatif. Pour le mappage vers le langage Java, la valeur `null` Java doit être utilisée pour indiquer l'absence d'une valeur facultative. Par exemple, si dans l'opération `triSend`, le paramètre d'adresse doit être omis, l'invocation de l'opération doit se présenter comme suit: `triSend(componentId, tsiPortId, null, sendMessage)`.

6.7 Initialisation de l'interface TRI

Toutes les méthodes sont non statiques, c'est-à-dire que les opérations ne peuvent être appelées que pour des objets. La présente Recommandation ne définissant pas les stratégies d'implémentation concrète des entités TE, SA et PA, le mécanisme selon lequel les entités TE, SA ou PA sont informées du traitement appliqué aux différents objets ne relève pas de la présente Recommandation.

Les fournisseurs d'utilitaires doivent fournir aux développeurs d'entités SA et PA des méthodes d'enregistrement des entités TE, SA et PA auprès de leur partenaire de communication respectif.

6.8 Traitement des erreurs

En dehors du traitement des erreurs défini au § 5.2, aucun autre traitement des erreurs n'est défini dans le présent mappage vers le langage Java. En particulier, aucun mécanisme de traitement des exceptions n'est défini.

7 Mappage vers le langage ANSI-C

7.1 Introduction

Le présent paragraphe définit le mappage vers le langage ANSI-C TRI pour les types de données abstraits spécifiés au § 5.3. Pour les types IDL de base, le mappage est conforme aux Recommandations OMG.

7.2 Noms et portées

Les identificateurs de paramètre du langage C doivent commencer par une minuscule, l'élément ou les éléments suivants qui forment l'identificateur de paramètre devant commencer par une majuscule. Par exemple, le paramètre IDL `SUTaddress` correspond à `sutAddress` en langage C.

Les identificateurs des types de données abstraits du langage C omettent l'élément `Type` de droite utilisé dans la définition IDL. Par exemple, le type IDL `TriPortIdType` correspond à `TriPortId` en langage C.

Les anciennes spécifications du langage C limitaient l'unicité des identificateurs aux 8 caractères de plus fort poids. Toutefois, les spécifications ANSI-C récentes ont étendu la limite d'unicité des identificateurs aux 31 caractères de plus fort poids. Hormis ce point, aucun problème de nommage ou de portée n'a été identifié dans ce mappage.

7.2.1 Mappage des types de données abstraits (ADT)

Type ADT TRI	Représentation ANSI-C	Notes et commentaires
TriAddress	BinaryString	
TriAddressList	typedef struct TriAddressList { TriAddress** addrList; long int length; } TriAddressList;	NOTE – Aucune valeur spéciale ne marque la fin de addrList[]. Le champ length doit être utilisé pour traverser cette matrice correctement.
TriComponentId	typedef struct TriComponentId { BinaryString compInst; String compName; QualifiedName compType; } TriComponentId;	NOTE – compInst signifie "instance de composant".
TriComponentIdList	typedef struct TriComponentIdList { TriComponentId** compIdList; long int length; } TriComponentIdList;	NOTE – Aucune valeur spéciale ne marque la fin de compIdList[]. Le champ length doit être utilisé pour traverser cette séquence tabulaire correctement.
TriException	BinaryString	
TriFunctionId	QualifiedName	
TriMessage	BinaryString	
TriParameterList	typedef struct TriParameterList { TriParameter** parList; long int length; } TriParameterList;	NOTE – Aucune valeur spéciale ne marque la fin de parList. Le champ length doit être utilisé pour traverser cette matrice correctement.
TriParameter	typedef struct TriParameter { BinaryString par; TriParameterPassingMode mode; } TriParameter;	
TriParameterPassingMode	typedef enum { TRI_IN = 0, TRI_INOUT = 1, TRI_OUT = 2 } TriParameterPassingMode;	NOTE – Les valeurs des instances de ce type doivent refléter le mode de transmission des paramètres défini dans les signatures de procédure TTCN-3 correspondantes.
TriPortIdList	typedef struct TriPortIdList { TriPortId** portIdList; long int length; } TriPortIdList;	NOTE – Aucune valeur spéciale ne marque la fin de portIdList []. Le champ length doit être utilisé pour traverser cette matrice correctement.
TriPortId	typedef struct TriPortId { TriComponentId compInst; char* portName; long int portIndex; QualifiedName portType; void* aux; } TriPortId;	NOTE 1 – compInst signifie "instance de composant". NOTE 2 – Pour la déclaration d'un singleton (et non d'une matrice), la valeur portIndex doit être de –1. NOTE 3 – Le champ aux vise à permettre d'étendre dans l'avenir la fonctionnalité TRI.

Type ADT TRI	Représentation ANSI-C	Notes et commentaires
TriSignatureId	QualifiedName	
TriStatus	long int #define TRI_ERROR -1 #define TRI_OK 0	NOTE – Toutes les valeurs négatives sont réservées en vue de l'extension future de la fonctionnalité TRI.
TriTestCaseId	QualifiedName	
TriTimerDuration	Double	
TriTimerId	BinaryString	NOTE – L'instruction en attente concernant la sémantique des temporisateurs et des instantanés pourra influer sur la représentation future.

7.2.2 Définitions des types ANSI-C

Type ADT C	Définition du type	Notes et commentaires
BinaryString	typedef struct BinaryString { unsigned char* data; long int bits; void* aux; } BinaryString;	NOTE 1 – data est une chaîne se terminant par une valeur autre que néant. NOTE 2 – bits est le nombre de bits utilisés dans les données. L'utilisation de la valeur –1 pour bits indique une valeur omise. NOTE 3 – Le champ aux vise à permettre d'étendre dans l'avenir la fonctionnalité TRI.
QualifiedName	typedef struct QualifiedName { char* moduleName; char* objectName; void* aux; } QualifiedName;	NOTE 1 – Les champs moduleName et objectName fields sont les identificateurs TTCN-3 stricto sensu. NOTE 2 – Le champ aux vise à permettre d'étendre dans l'avenir la fonctionnalité TRI.

7.2.3 Mappage de types IDL

Type IDL	Représentation ANSI-C	Notes et commentaires
Boolean	unsigned char	Mappage du langage IDL OMG vers le langage C++
String	char*	Mappage du langage IDL OMG vers le langage C++

7.2.4 Mappage des opérations TRI

Représentation IDL	Représentation ANSI-C
TriStatusType triSAReset()	TriStatus triSAReset()
TriStatusType triExecuteTestCase (in TriTestCaseIdType testCaseId, in TriPortIdListType tsiPortList)	TriStatus triExecuteTestCase (const TriTestCaseId* testCaseId, const TriPortIdList* tsiPortList)
TriStatusType triMap (in TriPortIdType compPortId, in TriPortIdType tsiPortId)	TriStatus triMap (const TriPortId* compPortId, const TriPortId* tsiPortId)
TriStatusType triUnmap (in TriPortIdType compPortId, in TriPortIdType tsiPortId)	TriStatus triUnmap (const TriPortId* compPortId, const TriPortId* tsiPortId)
TriStatusType triSend (in TriComponentIdType componentId, in TriPortIdType tsiPortId, in TriAddressType SUTaddress, in TriMessageType sendMessage)	TriStatus triSend (const TriComponentId* componentId, const TriPortId* tsiPortId, const TriAddress* sutAddress, const TriMessage* sendMessage)

Représentation IDL	Représentation ANSI-C
TriStatusType triSendBC (in TriComponentIdType componentId, in TriPortIdType tsiPortId, in TriMessageType sendMessage)	TriStatus triSendBC (const TriComponentId* componentId, const TriPortId* tsiPortId, const TriMessage* sendMessage)
TriStatusType triSendMC (in TriComponentIdType componentId, in TriPortIdType tsiPortId, in TriAddressListType SUTaddresses, in TriMessageType sendMessage)	TriStatus triSendMC (const TriComponentId* componentId, const TriPortId* tsiPortId, const TriAddressList* sutAddresses, const TriMessage* sendMessage)
void triEnqueueMsg (in TriPortIdType tsiPortId, in TriAddressType SUTaddress, in TriComponentIdType componentId, in TriMessageType receivedMessage)	void triEnqueueMsg (const TriPortId* tsiPortId, const TriAddress* sutAddress, const TriComponentId* componentId, const TriMessage* receivedMessage)
TriStatusType triCall (in TriComponentIdType componentId, in TriPortIdType tsiPortId, in TriAddressType SUTaddress, in TriSignatureIdType signatureId, in TriParameterListType parameterList)	TriStatus triCall (const TriComponentId* componentId, const TriPortId* tsiPortId, const TriAddress* sutAddress, const TriSignatureId* signatureId, const TriParameterList* parameterList)
TriStatusType triCallBC (in TriComponentIdType componentId, in TriPortIdType tsiPortId, in TriSignatureIdType signatureId, in TriParameterListType parameterList)	TriStatus triCallBC (const TriComponentId* componentId, const TriPortId* tsiPortId, const TriSignatureId* signatureId, const TriParameterList* parameterList)
TriStatusType triCallMC (in TriComponentIdType componentId, in TriPortIdType tsiPortId, in TriAddressListType SUTaddresses, in TriSignatureIdType signatureId, in TriParameterListType parameterList)	TriStatus triCallMC (const TriComponentId* componentId, const TriPortId* tsiPortId, const TriAddressList* sutAddresses, const TriSignatureId* signatureId, const TriParameterList* parameterList)
TriStatusType triReply (in TriComponentIdType componentId, in TriPortIdType tsiPortId, in TriAddressType SUTaddress, in TriSignatureIdType signatureId, in TriParameterListType parameterList, in TriParameterType returnValue)	TriStatus triReply (const TriComponentId* componentId, const TriPortId* tsiPortId, const TriAddress* sutAddress, const TriSignatureId* signatureId, const TriParameterList* parameterList, const TriParameter* returnValue)
TriStatusType triReplyBC (in TriComponentIdType componentId, in TriPortIdType tsiPortId, in TriSignatureIdType signatureId, in TriParameterListType parameterList, in TriParameterType returnValue)	TriStatus triReplyBC (const TriComponentId* componentId, const TriPortId* tsiPortId, const TriSignatureId* signatureId, const TriParameterList* parameterList, const TriParameter* returnValue)
TriStatusType triReplyMC (in TriComponentIdType componentId, in TriPortIdType tsiPortId, in TriAddressListType SUTaddresses, in TriSignatureIdType signatureId, in TriParameterListType parameterList, in TriParameterType returnValue)	TriStatus triReplyMC (const TriComponentId* componentId, const TriPortId* tsiPortId, const TriAddressList* sutAddresses, const TriSignatureId* signatureId, const TriParameterList* parameterList, const TriParameter* returnValue)

Représentation IDL	Représentation ANSI-C
TriStatusType triRaise (in TriComponentIdType componentId, in TriPortIdType tsiPortId, in TriAddressType SUTaddress, in TriSignatureIdType signatureId, in TriExceptionType exc)	TriStatus triRaise (const TriComponentId* componentId, const TriPortId* tsiPortId, const TriAddress* sutAddress, const TriSignatureId* signatureId, const TriException* exception)
TriStatusType triRaiseBC (in TriComponentIdType componentId, in TriPortIdType tsiPortId, in TriSignatureIdType signatureId, in TriExceptionType exc)	TriStatus triRaiseBC (const TriComponentId* componentId, const TriPortId* tsiPortId, const TriSignatureId* signatureId, const TriException* exception)
TriStatusType triRaiseMC (in TriComponentIdType componentId, in TriPortIdType tsiPortId, in TriAddressListType SUTaddresses, in TriSignatureIdType signatureId, in TriExceptionType exc)	TriStatus triRaiseMC (const TriComponentId* componentId, const TriPortId* tsiPortId, const TriAddressList* sutAddresses, const TriSignatureId* signatureId, const TriException* exception)
void triEnqueueCall (in TriPortIdType tsiPortId, in TriAddressType SUTaddress, in TriComponentId componentId, in TriSignatureIdType signatureId, in TriParameterListType parameterList)	void triEnqueueCall (const TriPortId* tsiPortId, const TriAddress* sutAddress, const TriComponentId* componentId, const TriSignatureId* signatureId, const TriParameterList* parameterList)
void triEnqueueReply (in TriPortIdType tsiPortId, in TriAddressType SUTaddress, in TriComponentIdType componentId, in TriSignatureIdType signatureId, in TriParameterListType parameterList, in TriParameterType returnValue)	void triEnqueueReply (const TriPortId* tsiPortId, const TriAddress* sutAddress, const TriComponentId* componentId, const TriSignatureId* signatureId, const TriParameterList* parameterList, const TriParameter* returnValue)
void triEnqueueException (in TriPortIdType tsiPortId, in TriAddressType SUTaddress, in TriComponentIdType componentId, in TriSignatureIdType signatureId, in TriExceptionType exc)	void triEnqueueException (const TriPortId* tsiPortId, const TriAddress* sutAddress, const TriComponentId* componentId, const TriSignatureId* signatureId, const TriException* exception)
TriStatusType triSUTActionInformal (in string description)	TriStatus triSUTActionInformal (const char* description)
TriStatusType triPAReset()	TriStatus triPAReset()
TriStatusType triStartTimer (in TriTimerIdType timerId, in TriTimerDurationType timerDuration)	TriStatus triStartTimer (const TriTimerId* timerId, TriTimerDuration timerDuration)
TriStatusType triStopTimer (in TriTimerIdType timerId)	TriStatus triStopTimer (const TriTimerId* timerId)
TriStatusType triReadTimer (in TriTimerIdType timerId, out TriTimerDurationType elapsedTime)	TriStatus triReadTimer (const TriTimerId* timerId, TriTimerDuration* elapsedTime)

Représentation IDL	Représentation ANSI-C
TriStatusType triTimerRunning (in TriTimerIdType timerId, out boolean running)	TriStatus triTimerRunning (const TriTimerId* timerId, unsigned char* running)
void triTimeout (in TriTimerIdType timerId)	void triTimeout (const TriTimerId* timerId)
TriStatusType triExternalFunction (in TriFunctionIdType functionId, inout TriParameterListType parameterList, out TriParameterType returnValue)	TriStatus triExternalFunction (const TriFunctionId* functionId, TriParameterList* parameterList, TriParameter* returnValue)

7.3 Gestion de mémoire

Obsolète.

7.4 Traitement des erreurs

Aucun traitement des erreurs n'a été défini pour ce mappage.

8 Scénarios d'utilisation

Le présent paragraphe expose des scénarios d'utilisation qui devraient aider les utilisateurs de l'interface TRI ainsi que les fournisseurs d'utilitaires mettant en œuvre cette interface à comprendre la sémantique des opérations définies dans la présente Recommandation.

Trois scénarios sont définis sous forme de diagrammes de séquences de messages (MSC, *message sequence chart*). Chaque scénario correspond à un fragment de code TTCN-3 qui utilise des fonctions de communication TTCN-3 à destination du système SUT ainsi que des fonctions de traitement de la temporisation. Les diagrammes MSC montrent les interactions entre les entités TE, SA et PA ainsi qu'avec le système SUT.

Il est à noter que les fragments de code TTCN-3 ne sont pas complets, étant donné que leur principal objet est d'utiliser le comportement dynamique. Tous les scénarios présentés utilisent un préambule MSC commun pour les opérations TRI, représenté sur la Figure 2.

Il est à signaler que les diagrammes MSC présentés dans le présent paragraphe utilisent des messages appariés pour modaliser chaque opération TRI. Le message MSC triMap suivi du message triMapOK indique, par exemple, que l'opération TRI triMap a été invoquée par l'entité TE et que la valeur de retour provenant de l'entité SA signifie que l'opération a été exécutée avec succès. Les appels d'opérations TRI, qui sont représentés au moyen de types abstraits et de leurs valeurs, ne sont donnés qu'à titre d'exemple. La représentation concrète de ces paramètres dans un langage cible donné est définie dans les mappages vers les différents langages.

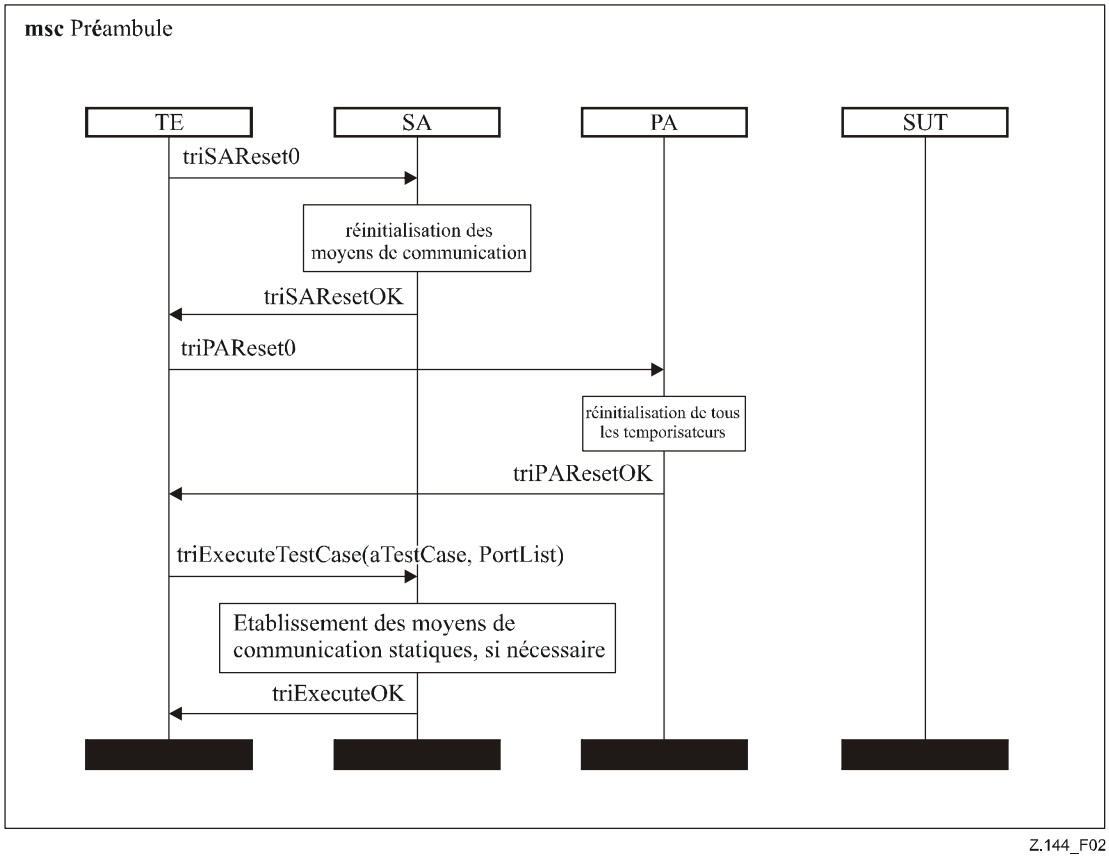

Figure 2/Z.144 – Préambule MSC commun

8.1 Premier scénario

Le premier scénario montre un certain nombre d'opérations de temporisation TTCN-3, à savoir les opérations de déclenchement de temporisateur (start timer) et de temporisateur en cours (timer running), d'opérations de communication en mode message, à savoir les opérations d'émission (send) et de réception (receive) ainsi que d'opérations de traitement de connexion, à savoir les opérations de mappage (map) et de démappage (unmap).

8.1.1 Fragment de code TTCN-3

```
module triScenario1
{
 external function MyFunction();

 type port PortTypeMsg message { inout integer }

 type component MyComponent {
    port PortTypeMsg MyPort;
    timer MyTimer
 }

 type component MyTSI {
    port PortTypeMsg PCO1;
 }

 testcase scenario1() runs on MyComponent system MyTSI
 {
    MyPort.clear;
    MyPort.start;
    MyTimer.start(2);

    map(MyComponent: MyPort, system: PCO1);
    MyPort.send (integer : 5);
    if (MyTimer.running)
    {
     MyPort.receive(integer:7);
    }
    else
    {
```

```
     MyFunction();
    }
    unmap(MyComponent: MyPort, system:PCO1);
    MyPort.stop;
 }


 control {
    execute( scenario1() );
 }

}
```

8.1.2 Diagramme de séquences de messages

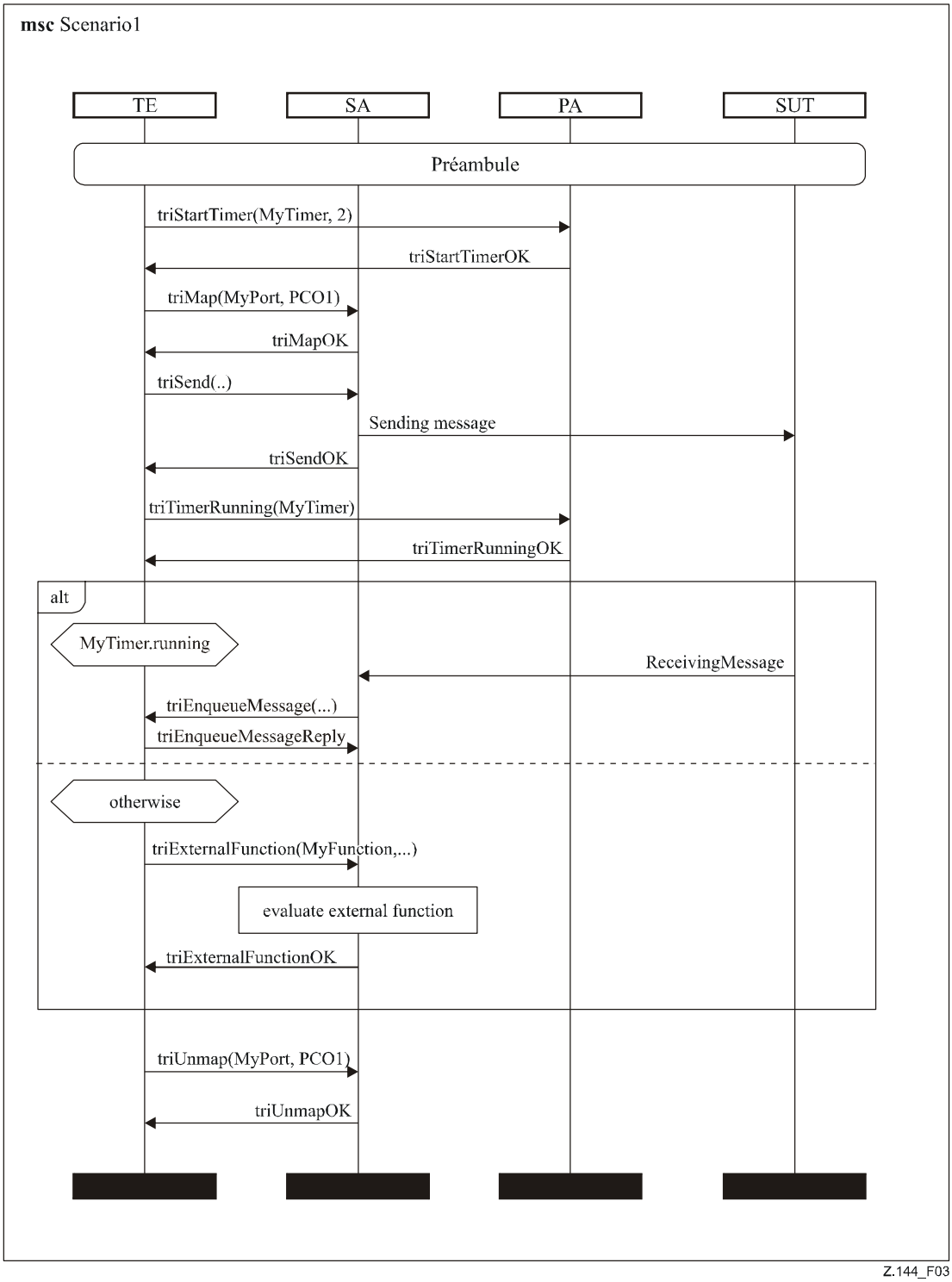

Figure 3/Z.144 – Scénario d'utilisation 1

8.2 Deuxième scénario

Le deuxième exemple représente un scénario analogue qui utilise également des opérations de communication en mode procédure avec temporisation qui sont lancées par le composant de test `MyComponent`. Dans cet exemple, le composant de test `MyComponent` est censé être utilisé comme composant de test principal (MTC).

8.2.1 Fragment de code TTCN-3

```
module triScenario2
{

 signature MyProc ( in float par1, inout float par2)
    exception(MyExceptionType);

 type record MyExceptionType { FieldType1 par1, FieldType2 par2 }

 type port PortTypeProc procedure { out MyProc }

 type component MyComponent {
    port PortTypeProc MyPort;
    timer MyTimer = 7
 }

 testcase scenario2() runs on MyComponent
 {
    var float MyVar;

    MyPort.clear;
    MyPort.start;
    MyTimer.start;

 MyVar := MyTimer.read;

 if (MyVar>5.0) {
     MyPort.call (MyProc:{MyVar, 5.7}, 5);
     alt {
     [] MyPort.getreply(MyProc:{-,MyVar*5}) {}
     [] MyPort.catch (MyProc, MyExceptionType:* ) {}
     [] MyPort.catch (timeout) {}
     }
    }
MyTimer.stop;
    MyPort.stop;
 }

 control {
    execute( scenario2() );
 }

}
```

8.2.2 Diagramme de séquences de messages

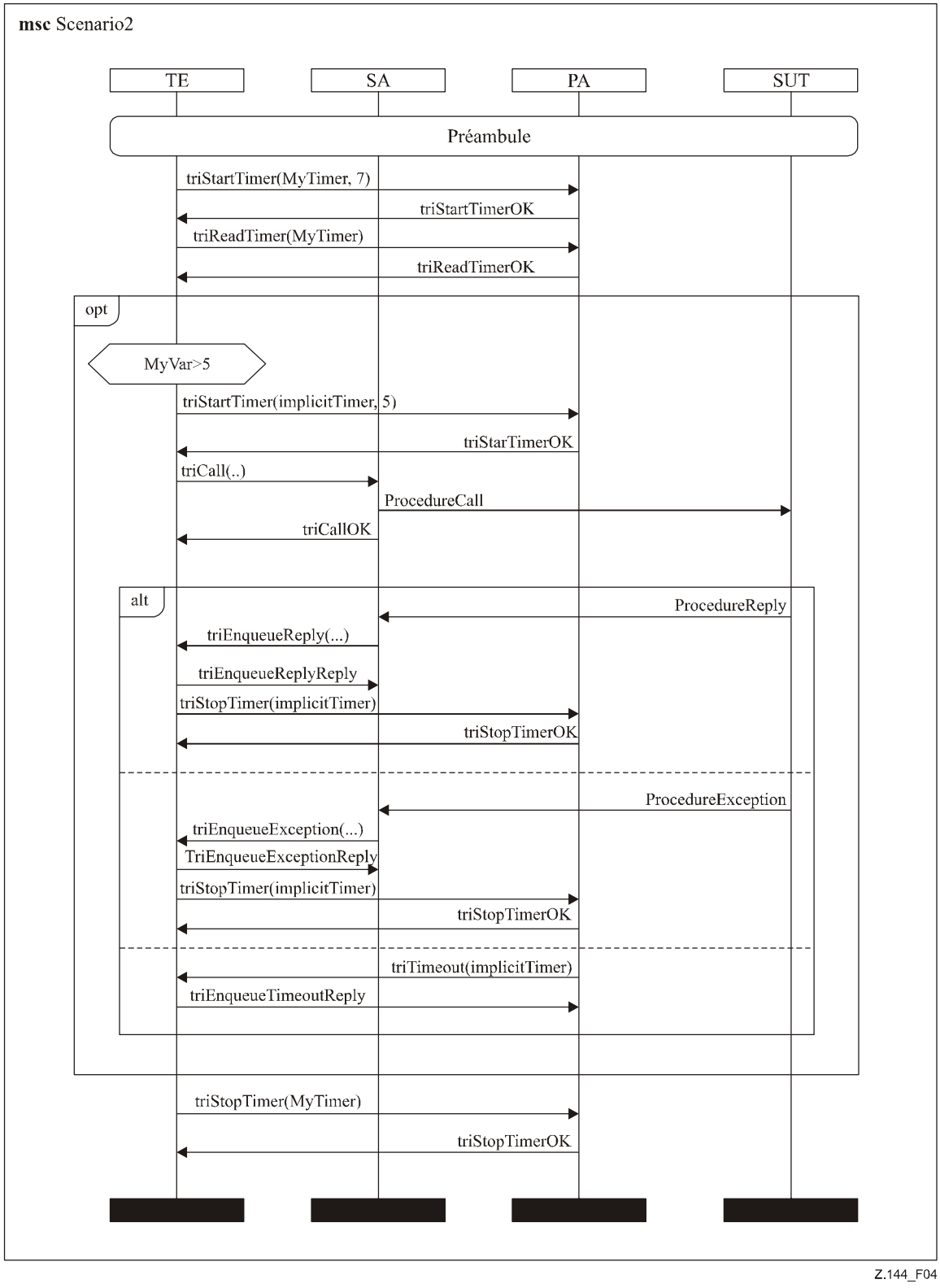

Figure 4/Z.144 – Scénario d'utilisation 2

8.3 Troisième scénario

Le scénario d'utilisation 3 représente la réception d'un appel de procédure ainsi qu'une réponse et la propagation d'une exception sur la base de cet appel reçu. Ici encore, le composant de test `MyComponent` est censé être utilisé en tant que composant MTC. `FieldType1`, `FieldType2`, `p1` et `p2` sont censés être définis ailleurs.

8.3.1 Fragment de code TTCN-3

```
module triScenario3
```

```
module triScenario3
{
 signature MyProc ( in float par1, inout float par2)
    exception(MyExceptionType);

 type record MyExceptionType { FieldType1 par1, FieldType2 par2 }

 type port PortTypeProc procedure { in MyProc }

 type component MyComponent {
    port PortTypeProc MyPort;
    timer MyTimer = 3
 }

 testcase scenario3(integer x) runs on MyComponent
 {
    MyPort.start;
    MyTimer.start;
    alt
    {
     [] MyPort.getcall(MyProc:{5.0, 6.0})
     {
     MyTimer.stop;
     }
     [x>5] MyTimer.timeout
     {
     MyPort.reply(MyProc:{-, 30.0});
     }
     [x<=5] MyTimer.timeout
     {
     MyPort.raise(MyProc, MyExceptionType:{p1, p2} );
     }
    }
    MyPort.stop;
 }

 control {
    execute( scenario3(4) );
 }
}
```

8.3.2 Diagramme de séquences de messages

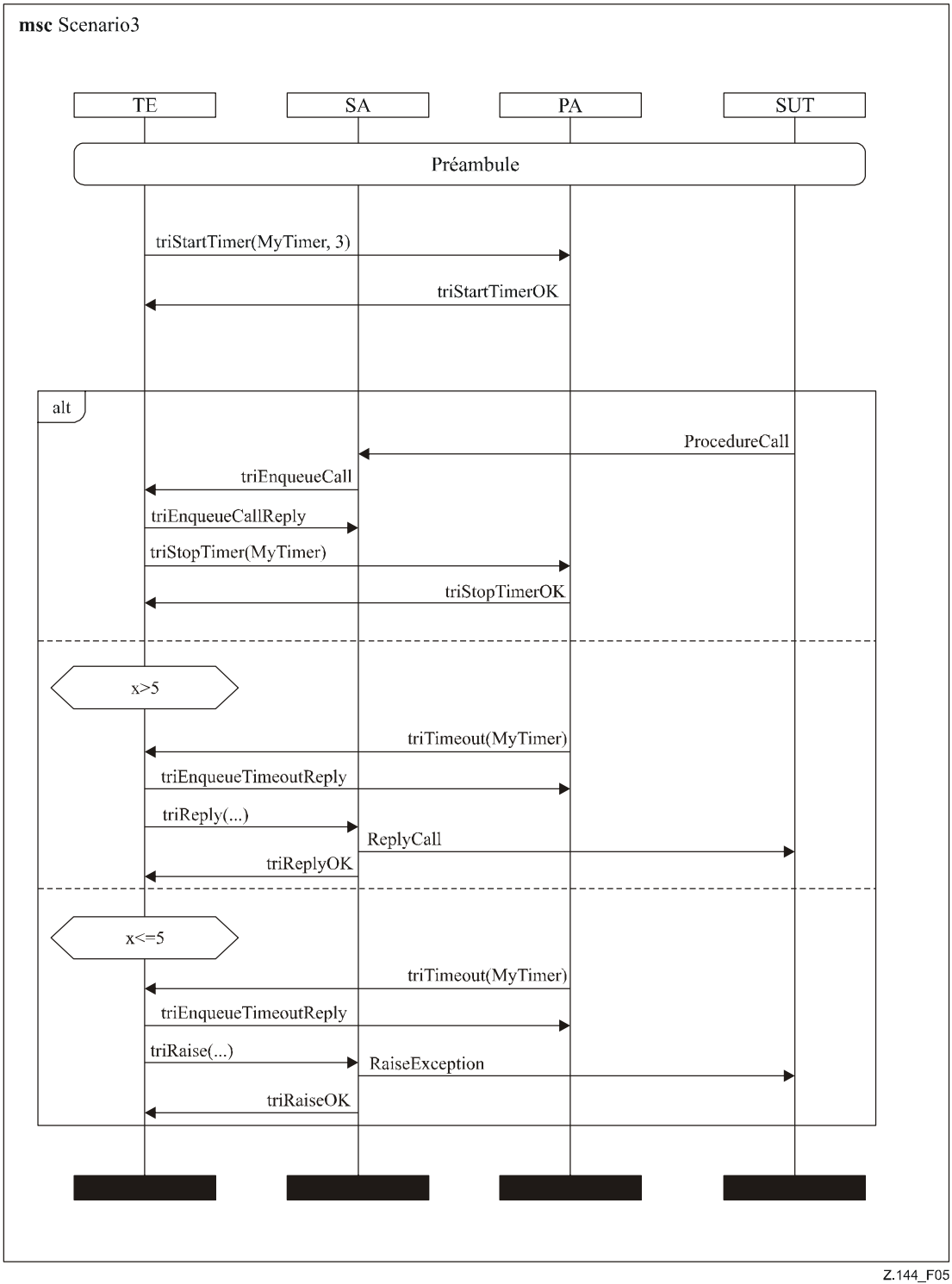

Figure 5/Z.144 – Scénario d'utilisation 3

Annexe A
(normative)

Récapitulation du langage de définition d'interface

La présente annexe récapitule les opérations TRI définies dans le § 5 dans le langage de définition d'interface (IDL).

```
// *********************************************************************
// Définition de l'interface d'exécution TTCN-3

// *********************************************************************

module triInterface
{

 //
 // *********************************************************************
 // Types
 // *********************************************************************
 //

 // Connexion
 native TriPortIdType;
 typedef sequence<TriPortIdType> TriPortIdListType;
 native TriComponentIdType;
 typedef sequence<TriComponentIdType> TriComponentIdListType;

 // Communication
 native TriMessageType;
 native TriAddressType;
 typedef sequence<TriAddressType> TriAddressListType;
 native TriSignatureIdType;
 native TriParameterType;
 typedef sequence<TriParameterType> TriParameterListType;
 native TriExceptionType;

 // Temporisation
 native TriTimerIdType;
 native TriTimerDurationType;

 // Divers
 native TriFunctionIdType;
 native TriTestCaseIdType;
 native TriStatusType;

 //
 // *********************************************************************
 // Interfaces
 // *********************************************************************
 //

 //
 // *********************************************************************
 // L'interface de communication (Réf.; Définition-TRI: 5.5)
 // *********************************************************************
 //
 interface triCommunication
 {

    // Opération de réinitialisation

    // Réf: TRI-Définition 5.5.1
    TriStatusType triSAReset();


    // Opérations de traitement de connexion

    // Réf: TRI-Définition 5.5.2.1
    TriStatusType triExecuteTestCase(in TriTestCaseIdType testCaseId,
    in TriPortIdListType tsiPortList);

    // Réf: TRI-Définition 5.5.2.2
    TriStatusType triMap(in TriPortIdType compPortId, in TriPortIdType tsiPortId);

    // Réf: TRI-Définition 5.5.2.3
    TriStatusType triUnmap(in TriPortIdType compPortId, in TriPortIdType tsiPortId);
```

```
// Opérations de communication en mode message

// Réf: TRI-Définition 5.5.3.1
TriStatusType triSend(in TriComponentIdType componentId, in TriPortIdType tsiPortId,
in TriAddressType SUTaddress, in TriMessageType sendMessage);
// Réf: TRI-Définition 5.5.3.2
TriStatusType triSendBC(in TriComponentIdType componentId, in TriPortIdType tsiPortId,
in TriMessageType sendMessage);
// Réf: TRI-Définition 5.5.3.3
TriStatusType triSendMC(in TriComponentIdType componentId, in TriPortIdType tsiPortId,
in TriAddressListType SUTaddresses, in TriMessageType sendMessage);

// Réf: TRI-Définition 5.5.3.4
void triEnqueueMsg(in TriPortIdType tsiPortId , in TriAddressType SUTaddress,
 in TriComponentIdType componentId, in TriMessageType receivedMessage);


// Opérations de communication en mode procédure

// Réf: TRI-Définition 5.5.4.1
TriStatusType triCall(in TriComponentIdType componentId, in TriPortIdType tsiPortId,
in TriAddressType SUTaddress, in TriSignatureIdType signatureId,
in TriParameterListType parameterList);
// Réf: TRI-Définition 5.5.4.2
TriStatusType triCallBC(in TriComponentIdType componentId, in TriPortIdType tsiPortId,
in TriSignatureIdType signatureId,
in TriParameterListType parameterList);
// Réf: TRI-Définition 5.5.4.3
TriStatusType triCallMC(in TriComponentIdType componentId, in TriPortIdType tsiPortId,
in TriAddressListType SUTaddresses, in TriSignatureIdType signatureId,
in TriParameterListType parameterList);

// Réf: TRI-Définition 5.5.4.4
TriStatusType triReply(in TriComponentIdType componentId, in TriPortIdType tsiPortId,
in TriAddressType SUTaddress, in TriSignatureIdType signatureId,
in TriParameterListType parameterList, in TriParameterType returnValue );
// Réf: TRI-Définition 5.5.4.5
TriStatusType triReplyBC(in TriComponentIdType componentId, in TriPortIdType tsiPortId,
in TriSignatureIdType signatureId,
in TriParameterListType parameterList, in TriParameterType returnValue );
// Réf: TRI-Définition 5.5.4.6
TriStatusType triReplyMC(in TriComponentIdType componentId, in TriPortIdType tsiPortId,
in TriAddressListType SUTaddresses, in TriSignatureIdType signatureId,
in TriParameterListType parameterList, in TriParameterType returnValue );

// Réf: TRI-Définition 5.5.4.7
TriStatusType triRaise(in TriComponentIdType componentId, in TriPortIdType tsiPortId,
in TriAddressType SUTaddress, in TriSignatureIdType signatureId,
in TriExceptionType exc);
// Réf: TRI-Définition 5.5.4.8
TriStatusType triRaiseBC(in TriComponentIdType componentId, in TriPortIdType tsiPortId,
in TriSignatureIdType signatureId,
in TriExceptionType exc);
// Réf: TRI-Définition 5.5.4.9
TriStatusType triRaiseMC(in TriComponentIdType componentId, in TriPortIdType tsiPortId,
in TriAddressListType SUTaddresses, in TriSignatureIdType signatureId,
in TriExceptionType exc);

// Réf: TRI-Définition 5.5.4.10
void triEnqueueCall(in TriPortIdType tsiPortId, in TriAddressType SUTaddress,
in TriComponentIdType componentId, in TriSignatureIdType signatureId,
in TriParameterListType parameterList );

// Réf: TRI-Définition 5.5.4.11
void triEnqueueReply(in TriPortIdType tsiPortId, in TriAddressType SUTaddress,
in TriComponentIdType componentId, in TriSignatureIdType signatureId,
in TriParameterListType parameterList, in TriParameterType returnValue );

// Réf: TRI-Définition 5.5.4.12
void triEnqueueException(in TriPortIdType tsiPortId, in TriAddressType SUTaddress,
in TriComponentIdType componentId, in TriSignatureIdType signatureId,
in TriExceptionType exc);
```

```
    // Opérations diverses

    // Réf: TRI-Définition 5.5.5.1
    TriStatusType triSUTactionInformal(in string description);

};
 //
 // *****************************************************************************
 // L'interface de plate-forme (Réf: TRI-Définition: 5.6)
 // *****************************************************************************
 //
 interface triPlatform
 {

    // Opération de réinitialisation

    // Réf: TRI-Définition 5.6.1
    TriStatusType triPAReset();


    // Opérations de traitement de temporisation

    // Réf: TRI-Définition 5.6.2.1
    TriStatusType triStartTimer(in TriTimerIdType timerId,
    in TriTimerDurationType timerDuration);

    // Réf: TRI-Définition 5.6.2.2
    TriStatusType triStopTimer(in TriTimerIdType timerId);

    // Réf: TRI-Définition 5.6.2.3
    TriStatusType triReadTimer(in TriTimerIdType timerId,
    out TriTimerDurationType elapsedTime);

    // Réf: TRI-Définition 5.6.2.4
    TriStatusType triTimerRunning(in TriTimerIdType timerId, out boolean running);

    // Réf: TRI-Définition 5.6.2.5
    void triTimeout(in TriTimerIdType timerId);


    // Opérations diverses

    // Réf: TRI-Définition 5.6.3.1
    TriStatusType triExternalFunction(in TriFunctionIdType functionId,
     inout TriParameterListType parameterList,
     out TriParameterType returnValue);

 };
};
```

BIBLIOGRAPHIE

– OMG CORBA (V2.2): *The Common Object Request Broker: Architecture and Specification*, (*Courtier commun de requête sur des objets: architecture et spécifications*), Section 3, février 1998.

– INTOOL CGI/NPL038 (V2.2): *Generic Compiler/Interpreter interface*; GCI Interface Specification, (*Interface de compilation/interprétation générique; spécification de l'interface GCI, outils d'infrastructure*), décembre 1996.

SÉRIES DES RECOMMANDATIONS UIT-T

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

Imprimé en Suisse
Genève, 2007