MASTER EN SCIENCES INFORMATIQUES

Protocoles de communication
Étude des outils de vérification de conformité d'une implémentation

Bovy, Philippe; Crasset, Serge

Award date:
1986

Awarding institution:
Universite de Namur

Facultés Notre-Dame de la Paix, Namur

Institut d'informatique

Année académique 1985-1986

Protocoles de communication :

Etude des outils de vérification de conformité d'une implémentation.

Philippe BOVY Serge CRASSET

Mémoire présenté en vue de
l'obtention du titre de licencié et maître
en informatique.

AVANT-PROPOS

Nous tenons tout d'abord à exprimer notre profonde reconnaissance à Monsieur Van Bastelaer qui a accepté d'assurer la direction de ce mémoire.

Que Mademoiselle Scoyer et Monsieur Celiktin se trouvent ici tout particulièrement assurés de notre profonde gratitude pour leur soutien constant et leurs conseils judicieux.

Nous adressons nos plus vifs remerciements à Messieurs Goossens et Lampo de la firme Siemens-Bruxelles ainsi qu'à Monsieur Eckardt et les membres du service DST/DF 143 de la firme Siemens-Munich pour leur accueuil chaleureux et leur attention durant notre séjour au sein de leur département.

Enfin, nous voudrions remercier tous ceux qui, professeurs, parents ou amis ont contribué, à quelque titre que ce soit, à notre formation et à la réalisation de ce mémoire.

TABLE DES MATIERES

4. ETUDE DES OUTILS DE TEST POUR LES PROTOCOLES DE HAUT NIVEAU

5. COMPARAISON DES DIFFERENTES ARCHITECTURES DECENTRALISEES

6. TESTS DES PROTOCOLES DE BAS NIVEAUX

7. LES TESTS

8. CONCLUSIONS

1. INTRODUCTION

Il est évident qu'à côté de l'informatique localisée, constituée d'installations indépendantes, la place de l'informatique répartie, c'est-à-dire des ensembles de systèmes informatiques distribués, sera de plus en plus importante. La coopération entre les équipements et systèmes interconnectés est régie par des règles, appelées protocoles, que chaque équipement doit respecter pour pouvoir collaborer avec les autres.

Cet objectif d'interconnexion entre équipements hétérogènes ne sera atteint que par une normalisation internationale des protocoles qui constitue, par ailleurs, un processus très lent. En effet, alors que le comité de l'ISO (International Standards Organization) chargé de la production de ces standards a tenu sa première conférence en1978, il a fallu attendre 1981 avant de voir publier les premiers projets de standards, en l'occurrence le DIS_7498 (Draft International Standard), qui forment ce que l'on appelle plus couramment le modèle de référence à sept couches. De plus, ce n'est seulement qu'en octobre 1984 que la version définitive (l'ISO_7498) a été approuvée par l'ensemble des membres de l'ISO. Actuellement, on peut noter qu'une préoccupation générale se manifeste pour ce sujet. Cela se traduit par de grands efforts dans le développement et la mise au point de nouveaux standards internationaux.

Cependant, toute cette activité risque d'être inefficiente et le terme "standard" pourrait perdre toute sa signification si les implémentations de protocoles ne sont pas élaborées conformément aux standards. Ainsi l'ISO se concentre sur la production des standards et non pas sur les produits qui pourraient en être déduits. Ce "désintéressement" de l'ISO vis à vis de la conformité a fait en sorte que le problème de conformité n'a pas obtenu, dès son origine, l'attention qu'il méritait.

introduction

Ce n'est que depuis 1980 qu'est apparu le désir de traiter ce problème à part entière, au même titre que celui de la standardisation. Ainsi, dès cette date et dans différents pays, des recherches ont été entamées et sont poursuivies en ce moment dans ce domaine.

Ces recherches se concentrent principalement sur les deux sujets suivants : la définition de la conformité et les méthodes permettant d'établir la conformité à un standard. De nombreux utilisateurs réclament en effet des produits conformes à ces standards, tandis que la plupart des fournisseurs aimeraient pouvoir garantir la qualité des produits qu'ils leur offrent. C'est pourquoi, on constate un intérêt croissant pour la mise en place de centres susceptibles de vérifier la conformité d'une implémentation par rapport à un standard. De tels centres, utiles tant aux futurs utilisateurs qu'aux réalisateurs de protocoles, devraient permettre d'arriver à un emploi efficace des normes édictées par les organismes internationaux de standardisation. Bien que les tests d'une implémentation ne puissent démontrer l'absence complète d'erreurs, l'utilisation d'une méthodologie certifiée permettra d'atteindre un haut degré de fiabilité dans l'interconnexion des systèmes.

Il est bon de faire remarquer que les tests de conformité ne suffisent pas avant la mise en oeuvre d'une implémentation sur un site. Les implémentations doivent aussi être testées dans le but de vérifier si elles satisfont aux exigences de leurs futurs utilisateurs. Ainsi, l'étendue des options offertes par l'implémentation, ses performances (rapidité dans l'établissement d'une connexion avec un partenaire, vitesse d'acheminement des données à celui-ci, etc ...) et sa robustesse (résistance aux erreurs) doivent être déterminées au moyen de mesures appropriées. La combinaison de tests de conformité et de ces mesures constituent l'évaluation d'une implémentation. Toutefois, dans ce travail, seuls les tests de conformité seront abordés.

Nous commencerons par rappeler brièvement les principes de la normalisation des sept couches proposée par l'ISO (chapitre 2) et nous situerons, dans le contexte du développement d'un système de communication, le problème de vérification de conformité d'une implémentation (chapitre 3).

Le chapitre 4 comportera une étude des différentes approches (centralisées et décentralisées) possibles pour la vérification de conformité d'une implémentation d'un protocole de haut niveau (couches 4, 5, 6 et 7) et détaillera les méthodes de test développées par certains organismes pionniers dans la recherche (NBS, NPL, Bull, GMD et ADI).

Le chapitre 5 contiendra une comparaison de ces diverses méthodes ainsi qu'une proposition d'architecture.

Quant au chapitre 6, il sera consacré aux architectures de test permettant d'établir la conformité d'une implémentation d'un protocole de bas niveau (couches 2 et 3).

Le chapitre 7 permettra au lecteur de se rendre compte de la manière dont les scénarios de test sont obtenus sur de telles architectures.

2. PRESENTATION RAPIDE DU MODELE ISO/OSI

2.1. Introduction à l'interconnexion des systèmes ouverts

2.1.1 Vue d'ensemble

Dans le but d'interconnecter les systèmes informatiques fournis par différents constructeurs, l'**ISO (International Organization for Standardization)** a défini un ensemble de standards internationaux pour permettre à ces différents systèmes de collaborer dans le but de réaliser le transfert d'informations. Cet ensemble de standards constitue le modèle de référence pour l'interconnexion de systèmes ouverts, appelé **OSI (Open System Interconnection)**.

Par **systèmes ouverts**, on entend ainsi les systèmes utilisant de tels standards, étant donné que chacun de ces systèmes est "ouvert" vis-à-vis des autres pour l'échange d'informations dans la poursuite de tâches communes distribuées. Aussi, l'ouverture des systèmes repose non pas sur une implémentation particulière ou sur des moyens d'interconnexion spéciaux, mais bien sur la reconnaissance mutuelle et la mise en oeuvre de ces standards internationaux.

Le modèle OSI, proposé par ISO, est accepté par la plupart des constructeurs et autres organisations internationales dont les activités sont orientées vers l'interconnexion des différents systèmes, à savoir l'**ECMA (European Computer Manufacturers Association)**, le **CCITT (Comité Consultatif International pour la Télégraphie et la Téléphonie)** et l'**ANSI (American National Standards Institute)**.

Ce modèle est également appelé "**modèle des 7 couches**" ; dans ce qui suit, nous allons en analyser les principales caractéristiques.

2.1.2. Concepts de base d'une architecture en couches

Dans le but de définir le comportement externe d'un système ouvert, le modèle de référence de l'ISO présente une vue logique des systèmes interconnectés qui est appelée "architecture de l'interconnexion des systèmes ouverts". La technique de structuration de base appliquée dans cette architecture est la division en couches. Selon cette approche une structure complexe est partitionnée en un nombre de couches fonctionnelles indépendantes.

Selon cette technique, chaque système est vu comme étant composé d'un ensemble de sous-systèmes, eux-mêmes constitués d'une ou plusieurs entités.

Les sous-systèmes de même rang forment la **couche-(N)** de l'architecture. Les entités existent au niveau de chaque couche, et les entités de systèmes différents figurant dans une même couche sont appelées **entités paires**.

Appelons par la suite **entité-(N)** une entité se trouvant dans la couche-(N) et entité-(N+1) une entité se trouvant dans la couche supérieure. La couche la plus haute n'a pas de couche-(N+1) au-dessus d'elle ; de même, la couche la plus basse n'a pas de couche-(N-1) en-dessous d'elle.

A l'exception de la couche la plus haute, chaque couche-(N) fournit aux entités de la couche-(N+1) un service de niveau-(N), noté **service-(N)**. Les services d'une couche-(N) sont fournis à la couche supérieure en réalisant un ensemble de fonctions définies dans la couche-(N) et en utilisant les services disponibles de la couche-(N-1). Une entité peut disposer des services d'une (ou plusieurs) entité(s) de la couche inférieure au moyen de points d'accès au service, notés SAP (service acces point), comme le montre la figure 2.1.

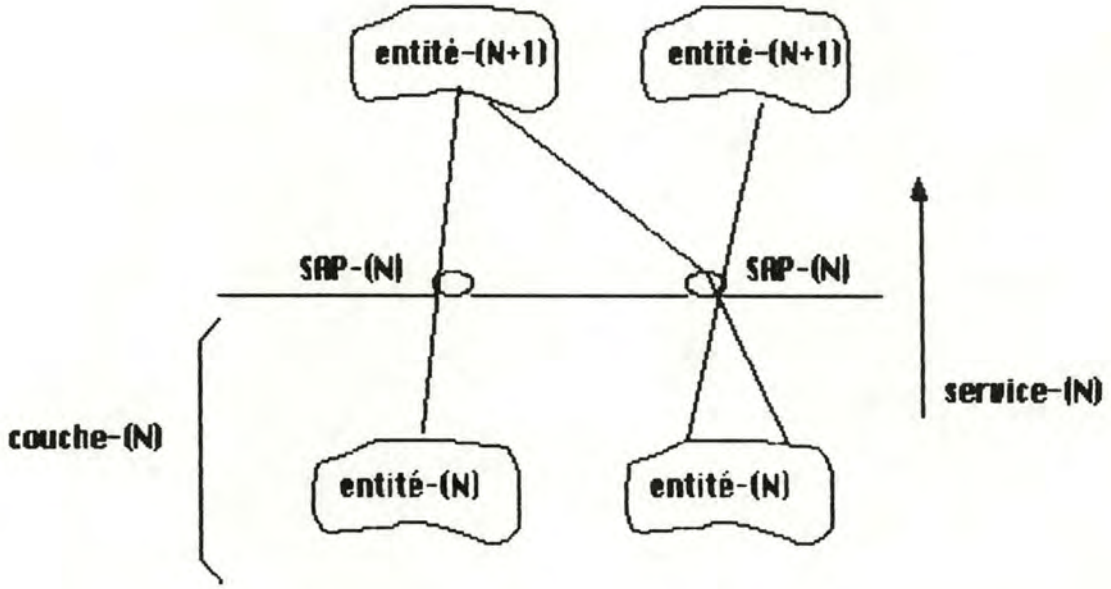


Figure 2.1 : entité et point d'accès au service.

Un SAP est donc le moyen d'accès par lequel deux entités appartenant à des couches adjacentes communiquent. Notons **SAP-(N)** le SAP qui permet à une entité-(N+1) de disposer du service fourni par une entité-(N).

Deux entités appartenant à des couches adjacentes communiquent entre elles en respectant des règles précises matérialisées par un ensemble de primitives appelé **interface**. Pour une entité-(N), on distingue :

- son interface vers le haut (**interface-(N)**) via laquelle elle fournit ses services à l'entité-(N+1) ;
- son interface vers le bas (**interface-(N-1)**) via laquelle elle accède aux services de l'entité-(N-1).

La figure 2.2 représente les interfaces associées à une entité-(N).

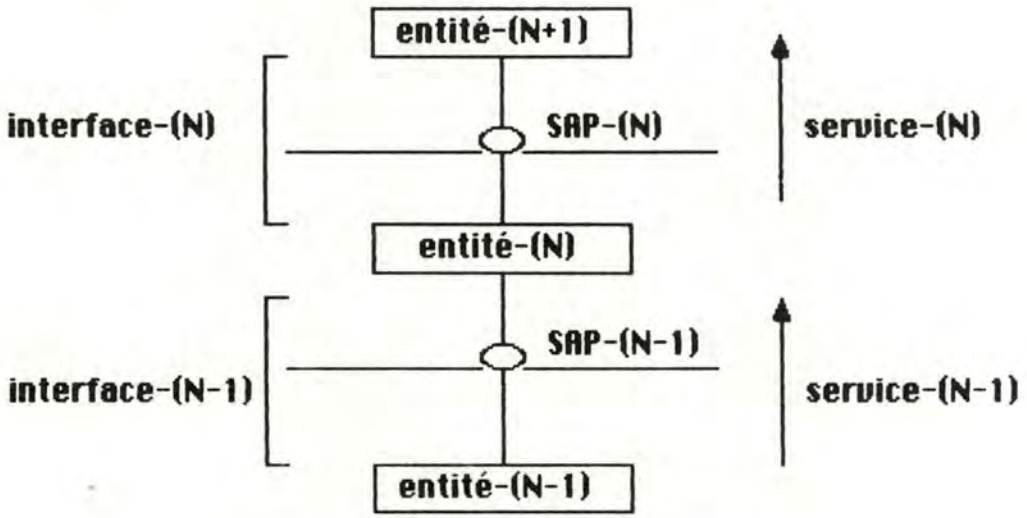


Figure 2.2 : entité, service et interface.

Les informations que l'entité-(N+1) désire faire parvenir à son entité paire sont communiquées à l'entité-(N) au moyen d'unités de données de service, notées **SDU-(N)** (service data unit).

Comme le montre la figure 2.3, la coopération entre les entités-(N) paires d'une même couche est régie par un **protocole-(N)**. Un protocole-(N) est un ensemble de règles et de formats (sémantiques et syntaxiques) selon lesquels des informations de contrôle du protocole-(N) et les unités de données du service-(N) sont échangées entre des entités-(N) dans l'accomplissement des fonctions propres à la couche-(N). L'unité de communication entre deux entités paires est appelé "**élément de protocole**" (PDU, Protocol Data Unit) . Celui-ci contient généralement deux parties, à savoir les données à transmettre et les informations de contrôle. Un **protocole de bout-en-bout** est un protocole qui permet un transfert de données, transparent (en ce qui concerne le nombre de noeuds intermédiaires, réseaux ou sous-réseaux impliqués) pour les systèmes terminaux qui utilisent ce protocole. Les protocoles definis pour les couches 4,5,6 et 7 sont des protocoles de bout-en-bout.

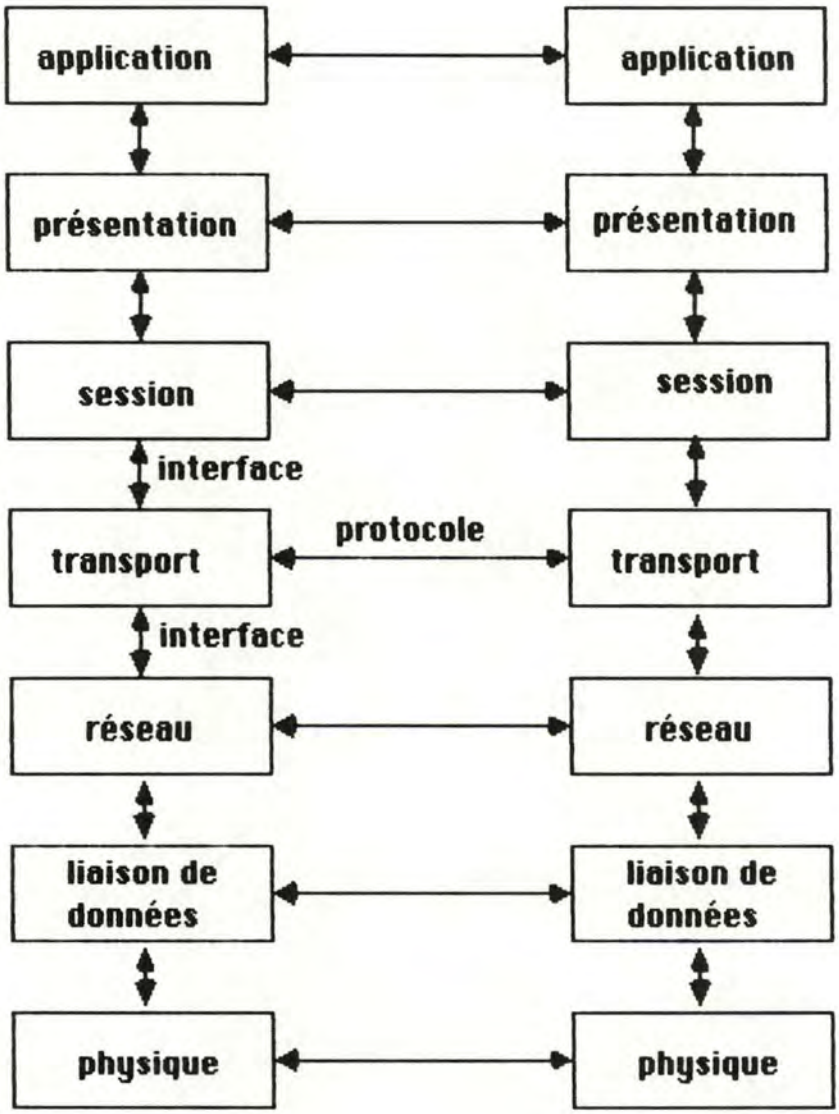


Figure 2.3 : les sept couches de l'ISO.

2.2. Description sommaire des différentes couches

Le modèle de référence OSI possède une architecture constituée de sept couches et est représenté à la figure 2.3. Nous allons décrire brièvement les fonctionnalités de chacune de ces couches, en commençant par la couche la plus haute de l'architecture.

2.2.1. La couche application

En tant que couche la plus élevée, la couche application fournit des services aux utilisateurs de l'environnement ISO/OSI, et non plus à une couche supérieure. Tous les paramètres dont la détermination relève de l'utilisateur, lors de l'établissement de chaque communication, sont notifiés à l'environnement de l'ISO/OSI via la couche d'application. Cette couche fonctionne selon un protocole de bout-en-bout. (Il en est de même pour les couches inférieures jusque et y compris la couche transport)

2.2.2. La couche présentation

Le but de la couche présentation est de délivrer les informations aux entités d'application de façon à ce que leur signification soit préservée tout en apportant une solution (grâce notamment à une transformation de données ou à un formatage de celles-ci) aux problèmes créés par les différences de syntaxe.

2.2.3. La couche session

Le but de la couche session est de fournir les moyens nécessaires à des entités de présentation qui coopèrent pour organiser et synchroniser leur dialogue et pour contrôler leur échange d'informations. A cette fin, la couche session offre des services permettant d'établir une connexion de session entre deux entités de présentation.

Pour assurer le transfert de données entre entités de présentation, la connexion de session utilise les services de la couche transport.

2.2.4. La couche transport

La couche transport a pour rôle de fournir un service de transport en association avec les services sous-jacents assurés par les couches inférieures. Le service de transport assure un transfert transparent de données entre entités de session. Il décharge les entités de session de tous détails de mise en oeuvre du transfert de données entre elles, de façon fiable et économique. Ce service doit également optimiser l'emploi des services de communication disponibles de la couche réseau afin d'obtenir, de la manière la plus économique, les performances demandées pour chaque connexion entre entités de session.

2.2.5. La couche réseau

La couche réseau fournit les procédures et moyens fonctionnels nécessaires à l'échange d'unités de données du service de réseau entre deux entités de transport, y compris dans le cas où des noeuds intermédiaires sont impliqués. Ce service est fourni grâce à l'établissement, le maintien, la libération de connexions réseau et au transfert de données sur ces connexions. Elle décharge les entités de transport de tout souci de routage et de commutation.

2.2.6. La couche liaison de données

Les procédures de contrôle des liaisons de données détectent et corrigent, dans la mesure du possible, les erreurs pouvant se produire pendant la transmission physique. Le but de la couche liaison de données est de fournir des procédures et moyens fonctionnels nécessaires à l'établissement, le maintien et la libération d'une ou plusieurs liaisons de données exploitées lors du transfert de données entre entités de réseau.

2.2.7. La couche physique

La couche physique fournit les procédures ainsi que les moyens mécaniques, électriques et fonctionnels nécessaires à l'établissement, au maintien et à la libération des connexions physiques entre entités de liaison.

Le lecteur désirant un complément d'informations sur la définition de ces couches pourra consulter [TAN] et [RFM].

3. PRESENTATION GLOBALE DES METHODES ET OUTILS D'AIDE AU DEVELOPPEMENT DE SYSTEMES DE COMMUNICATION

3.1. Concepts de base

L'objectif de ce chapitre sera d'une part de montrer quel pourrait être un environnement global de développement de protocoles et d'autre part de bien situer dans son ensemble ce qui sera développé dans le cadre de ce mémoire, à savoir les techniques de réalisation de tests et de génération automatique de séquences de test. Pour ce faire, il convient de bien distinguer les termes suivants :

- **Outil** : ce terme sera employé sous son acceptation commune, à savoir un instrument dont on se sert pour réaliser un travail manuel ou automatisé ;
- **Méthode** : une méthode définit la manière dont on doit utiliser un outil pour arriver à un résultat donné ;
- **Technique** : ensemble d'outils et méthodes ;
- **Architecture** : la définition d'une architecture consiste en une description fonctionnelle de ses différents composants ainsi qu'en une analyse des relations existant entre ces différents composants. Un composant peut être défini comme un outil (logiciel ou matériel) réalisant une fonction déterminée au sein de l'architecture.

3.2. Présentation des différentes phases de développement d'un protocole

L'architecture décrite à la figure 3.1 illustre un enchainement logique de différents outils pouvant être utilisés dans le cadre du développement et des tests d'un protocole. On peut distinguer dans celle-ci trois grandes phases :

- la **description et validation formelles du protocole** qui regroupe différents outils visant à vérifier, via une modélisation du protocole, si les spécifications de celui-ci sont cohérentes ;
- la **génération d'une implémentation** qui regroupe un ensemble d'outils visant à faciliter et accélérer le processus de mise en oeuvre d'un protocole ;
- la réalisation de tests qui pourra comprendre des outils de **gestion automatique de tests** ainsi que des outils de **génération automatique de séquences de test**.

L'ensemble de ces outils peuvent être intégrés en une architecture globale d'aide au développement de protocoles. Notons qu'il importe qu'une telle architecture soit suffisamment modulaire et paramètrable pour pouvoir s'adapter très rapidement à l'apparition de nouvelles spécifications. Compte tenu notamment de l'absence de normes bien établies pour les protocoles des couches supérieures (et justement en prévision de leur arrivée), ce critère de modularité est d'autant plus justifié.

Présentation globale des méthodes et outils

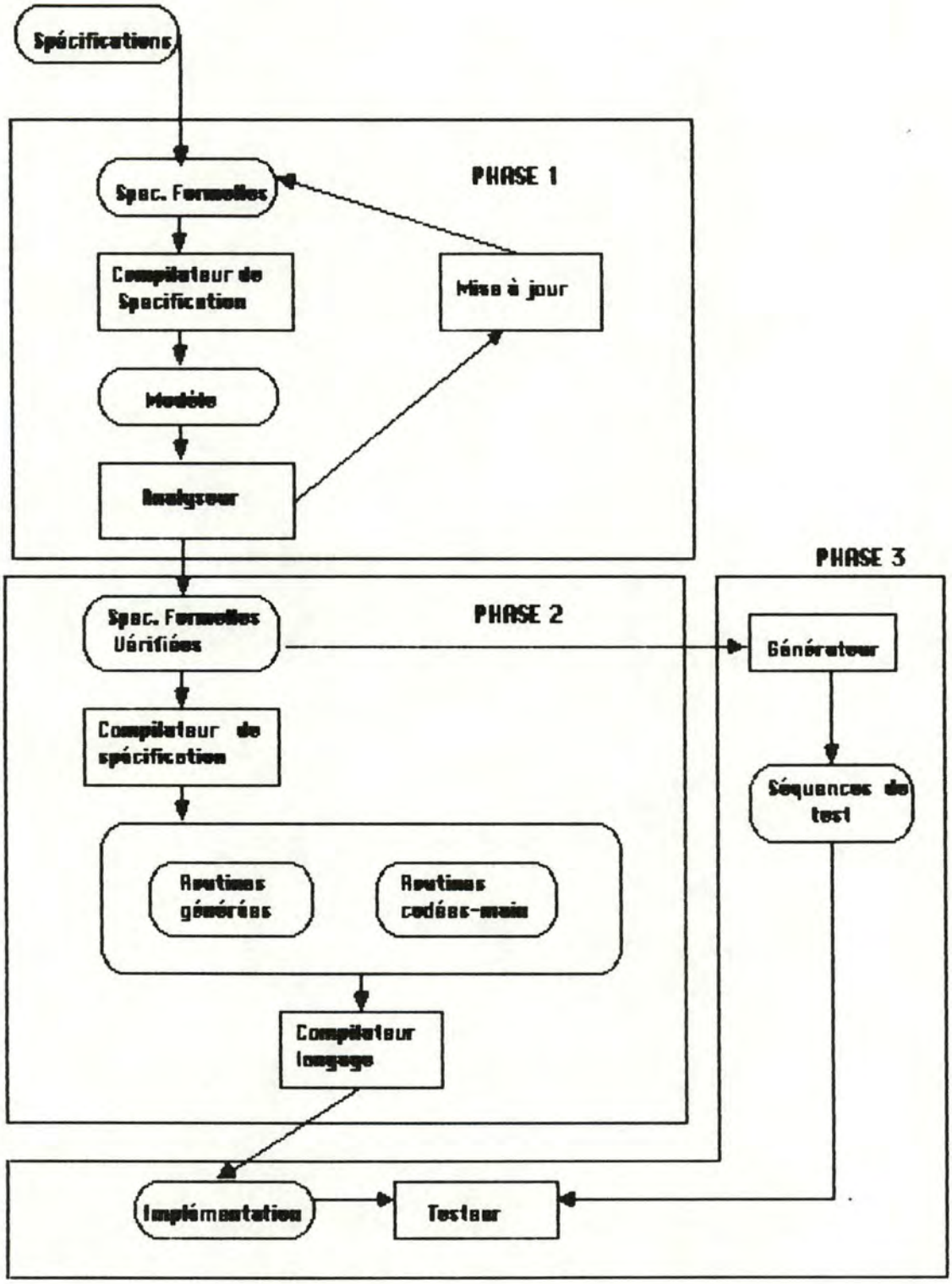


Figure 3.1 : environnement global d'aide au développement de protocoles.

3.3. Description sommaire de quelques outils

3.3.1. Outils de description et validation formelles

Les techniques de **description** (description du comportement réel du système) et de **spécification** (description du comportement attendu du système) **formelles** présentent vis-à-vis des techniques dites "informelles" basées sur une description en langue naturelle, deux avantages indéniables :

- elles permettent de décrire de manière non ambiguë le comportement d'un système ;
- elles permettent une automatisation des processus d'analyse du protocole.

Ces deux avantages font qu'un nombre croissant de langages de description et de spécification apparaissent sur le marché.

Nombre d'entre eux sont basés sur le **modèle d'automate à états finis (FSM : Finite State Machine)** qui semble particulièrement bien adapté pour décrire le comportement de systèmes de télécommunication. Ce modèle est défini en termes d'états (attente d'un message), de transitions (exécution d'un traitement), d'inputs (messages entrants) et d'outputs (messages sortants). En fait, il est constitué d'un ensemble fini d'états reliés par des transitions. La survenance d'un événement provoque le passage du système de l'état courant à un nouvel état, via une transition (exécution de traitements et génération d'outputs).

Parmi les langages proposés, trois sont définis par des organismes internationaux de normalisation :

- **SDL** (*Functional Spécification and Description Language*) est une méthode de représentation formalisée proposée par le CCITT ;
- **FDT** (*Formal Description Technique*), **ESTELLE** qui devraient être utilisés (notamment par le NBS) pour décrire les protocoles ISO.

Il existe également une autre famille de langages qui se base sur la logique temporelle (ex : **LOTOS**).

Quant à la **validation de protocoles** formalisés (aussi appelée vérification de protocoles), un grand nombre d'outils sont actuellement disponibles ; outils évidemment dépendants de la méthode de formalisation utilisée.

Par exemple, la technique la plus directe pour valider un protocole entre deux machines dont le comportement est décrit selon la méthode FSM consiste en une **exploration d'états**. Selon cette technique, un graphe d'accessibilité est construit ; graphe dont les noeuds représentent les états accessibles et dont les arcs correspondent aux transitions. On se réfère ensuite à la théorie des graphes pour déterminer les propriétés du graphe construit et un lien est établi entre les propriétés de ce graphe et celles du protocole. Ainsi, par exemple, si l'on démontre que le graphe est fini (c'est-à-dire qu'il comporte un nombre fini d'arcs), on est assuré que le protocole vérifiera la propriété de limitation définie ci-après. Des théorèmes semblables existent pour d'autres propriétés d'un protocole que voudrait démontrer.

Malgré la diversité des techniques de validation, on peut dégager une constante à savoir que, quels que soient les outils et méthodes utilisés, ils devront nécessairement établir certaines propriétés cruciales du protocole. Ces propriétés définies en termes du modèle FSM sont les suivantes :

- complétude : chaque état possible et chaque événement pouvant survenir sont modélisés ;
- absence d'impasse (deadlock) : il ne peut exister de sous-ensemble d'états tel qu'une fois atteint le système ne progresse plus et qu'il n'existe aucune transition permettant d'en sortir ;
- absence de cycle (livelock) : il ne peut exister d'input ou de séquence d'inputs qui se traduisent par un traitement infiniment répété sans qu'il n'y ait de progression du système ;
- terminaison : quelle que soit la séquence d'inputs fournie, le système doit évoluer vers un état final déterminé ;
- limitation : il ne peut exister aucun input ou séquence d'inputs qui soit mis éternellement en attente de traitement. Autrement dit, il doit être impossible de créer une file infinie de tâches.

Comme il ne sera plus question de description et validation formelle de protocole par la suite, le lecteur qui souhaite obtenir davantage d'informations sur le sujet pourra notamment consulter : Billiard (1983), Gouda (1985), Rafiq et Ansart (1983), Sunshine (1983) et [Z10].

3.3.2. Outils de génération automatique d'implémentations

Une fois les spécifications vérifiées, on peut passer à l'implémentation de celles-ci. Afin, d'une part, de faciliter la réalisation de cette phase et d'autre part, de diminuer le risque d'erreur, des techniques de génération automatique ont été et sont mises au point. Celles-ci reposent essentiellement sur un compilateur qui, sur base de spécifications formelles, produit une implémentation (ou plus exactement une partie de cette implémentation) dans un langage de haut niveau. Une telle tâche est rendue fort ardue par le fait qu'une partie de l'implémentation repose nécessairement sur diverses décisions notamment

- la manière dont les couches sont implémentées : à titre d'exemple, elles peuvent l'être comme faisant partie du système d'exploitation et dès lors être accédées par des appels systèmes ; alternativement, elles peuvent être implémentées comme des processus utilisateurs coopérants ou comme des processus utilisateurs séparés ou encore comme une combinaison de ces méthodes ;

- la manière dont est traitée l'interaction entre couches : à savoir soit via un accès à un segment de mémoire partagé avec d'autres couches soit via un mécanisme quelconque de communication inter-processus (par exemple via le mécanisme des "pipes" sur environnement UNIX, ou le mécanisme des "queues" que l'on rencontre dans de nombreux systèmes d'exploitation) ;

- la manière dont sont structurées les données.

Ces diverses décisions nécessitent des choix qui sont inéluctablement dépendants du cadre et de l'environnement dans lesquels l'application est développée.

Cependant, une bonne séparation de ce qui dépend de l'implémentation et de ce qui n'en dépend pas, ainsi qu'une bonne modularisation de l'architecture de l'implémentation rendent possible la génération automatique d'une partie significative du codage de l'implémentation. Viendront s'adjoindre à cela un ensemble de primitives "codées-main" qui traduisent les choix opérés pour la partie dépendante de l'implémentation.

A titre d'exemple, le NBS (National Bureau of Standards) a développé une implémentation du protocole de transport classe 4 comportant 4500 lignes de code générées automatiquement et 5700 lignes "codées-main" ce qui représente approximativement 40% d'automatisation. Cette expérience a également montré que la partie générée n'était pas plus importante que si l'opération était réalisée manuellement.

Ces implémentations générées automatiquement (ou semi-automatiquement) présentent un haut degré de fiabilité quant à leur respect des spécifications ; c'est la raison pour laquelle elles sont parfois utilisées comme constituant d'un système de test comme on le verra dans le cadre de l'étude des architectures de test.

Ce sujet n'étant plus abordé par la suite, le lecteur pourra obtenir davantage d'informations en consultant les références suivantes : Blumer et Tenney (1982), Bochman (1982), Ansart et Chari (1983).

3.3.3. Outils de test de conformité

Préalablement à l'étude des techniques de tests de conformité, il est essentiel d'essayer de définir ce que signifie la conformité d'une implémentation vis-à-vis d'un standard.

Actuellement, toute définition reste subjective, chaque organisation définissant ses propres critères d'interprétation de ce que constitue la conformité. Ce problème trouve son origine dans la manière dont sont définis les standards [cfr RAYNER (1983 a)]. Ainsi, la spécification d'un standard pourra :

- présenter certaines ambiguïtés : dès lors, sur quelles interprétations baser la définition de la conformité ? ;

- mentionner diverses options : pour être conforme, une implémentation doit-elle proposer l'ensemble des options et ainsi pouvoir répondre à toutes demandes ? Une implémentation peut-elle être conforme si elle ne présente aucune option ? ;
- se référer à d'autres standards : la conformité à un protocole impose-t-elle d'être conforme à l'ensemble des autres protocoles référencés dans ce protocole ?.

La subjectivité qu'entraînent ces divers aspects ne saurait être que nocive quant à la crédibilité des tests qui sont menés. En effet, il est inconcevable que deux organisations arrivent à des conclusions différentes lors du test d'une même implémentation.

Néanmoins, la situation actuelle pourrait être améliorée :

- en essayant d'éliminer les risques d'ambiguïté dans la définition des standards notamment via l'édition de normes formalisées ;
- en standardisant les techniques de tests de conformité ;
- en associant à chacun des standards un ensemble reconnu de séquences de tests à utiliser pour vérifier la conformité.

Dans l'attente d'une telle situation, on dira qu'une implémentation est conforme à une spécification de protocole si elle répond fidèlement aux principes fondamentaux du protocole et aux options sélectionnées pour les ambiguïtés mentionnées ci-dessus.

En partant de cette définition, diverses méthodes et architectures utilisées pour la réalisation de tests de conformité seront présentées dans les chapitres 4 et 6. En particulier, les architectures suivantes seront étudiées :

- architecture de type "**encodage/décodage**" où une implémentation "améliorée" permettant des demandes de service plus précises interagit avec l' équipement à tester ;
- architecture de type "**générateur d'éléments de protocoles**" dans laquelle aucune implémentation de protocole n'est mise en correspondance avec l'implémentation à tester. Seul un générateur susceptible de construire et analyser les éléments de protocole est requis;
- architecture de type "**implémentation de référence**" où une implémentation type du protocole interagit avec un équipement à tester en mettant en jeu le service offert ;

Une attention toute particulière sera accordée au développement d'une architecture qui soit, autant que faire se peut, indépendante du protocole à tester. Une étude comparative de ces différentes architectures essayant d'isoler les avantages de chacune d'entre elles sera proposée au chapitre 5.

3.3.4. Outils de génération automatique de séquence de test

Une des sous-tâches principales à la réalisation de test pour une implémentation de protocole consiste en la génération efficace de **séquences de test**. De telles séquences peuvent être définies comme étant un enchainement d'événements-stimuli pour l'implémentation à tester . Ces séquences, qui seront fournies au testeur, peuvent être produites soit manuellement soit automatiquement sur base des spécifications formelles.

La bonne réalisation de cette phase peut avoir une énorme influence sur la qualité des tests qui seront menés. En effet, il est inutile de disposer d'une très bonne architecture si les tests qui sont menés sur celle-ci sont mal définis.

Etant donné toute l'importance que peut revêtir le choix, l'ordonnancement,... et donc la génération automatique de séquences de test, ce thème sera abordé de manière plus approfondie dans le cadre du chapitre 7.

3.4. Graphe d'intégration des différents outils

Les différents outils qui viennent d'être analysés peuvent être organisés pour définir l'architecture globale de développement de protocoles, représentée la figure 3.1. Tout en décrivant cette figure, cette section donne un résumé de ce qui a été dit précédemment sur chacun des outils pouvant être utilisés.

La première phase consiste en la vérification des spécifications du protocole. On crée tout d'abord des spécifications formelles à partir des spécifications édictées par les organismes internationaux de standardisation (ISO, CCITT, ECMA). Ces spécifications formelles, interprétables tant par l'homme que par la machine, constituent l'input d'un compilateur de spécifications.

Ce compilateur produit un modèle (d'automates à états finis, par exemple) décrivant les mécanismes fondamentaux dérivés des spécifications.

Ce modèle est alors transmis à un analyseur dont le rôle est de déterminer (via une exploration d'états par exemple) si le protocole décrit par les spécifications formelles respecte bien les cinq critères de cohérence, à savoir la complétude, l'absence d'impasse, l'absence de cycle, la terminaison et la limitation. Si des erreurs sont détectées, les spécifications formelles sont modifiées en conséquence et vérifiées de nouveau. Ce cycle est répété jusqu'à obtention de spécifications correctes qui servent d'input à la phase 2.

La seconde phase consiste en la génération d'une implémentation. Un compilateur de spécifications recevant les spécifications vérifiées produit, dans un langage de haut niveau, un ensemble de routines représentant la partie de l'implémentation indépendante de l'environnement. On y ajoute alors des routines codées-main traduisant la dépendance vis-à-vis de l'environnement. Le tout est compilé pour donner une version correcte (du point de vue de la compilation) de l'implémentation, version servant d'input à la phase 3.

La phase 3 consiste en la réalisation de tests de conformité. L'élaboration de séquences de tests est réalisée en ayant recours à un générateur qui, sur base des spécifications formelles vérifiées, est susceptible de produire de telles séquences. Celles-ci, ainsi que l'implémentation correctement compilée sont utilisées par le testeur qui réalise les tests de conformité.

4. ETUDE DES OUTILS DE TEST POUR LES PROTOCOLES DE HAUT NIVEAU

4.1.Introduction

4.1.1. Les deux approches

Comme déjà signalé au niveau de l'analyse globale des outils, il ne suffit pas de disposer de protocoles standards formalisés pour réaliser l'interconnexion, encore faut-il disposer d'implémentation conformes de ces protocoles. Pour la mise en oeuvre de telles implémentations, il faudra nécessairement recourir à un ensemble d'outils et méthodes (techniques de test) permettant de vérifier la conformité d'une réalisation d'un protocole.

Comme le montre le modèle de référence ISO/OSI, dans une architecture en couches, chaque protocole possède :

- une interface vers la couche supérieure via laquelle lui sont fournies les demandes de service ;
- une interface vers la couche inférieure via laquelle il réalise ses demandes de service ;
- une "interface" logique d'où lui parviennent les éléments de protocole provenant de l'entité paire.

L'objectif consistera à manipuler ces interfaces par une sélection adéquate des événements et séquences d'événements.

En toute généralité, les techniques de test devront être capables d'établir si une implémentation de protocole :

- répond correctement aux demandes valides de service ;
- répond correctement aux éléments de protocole valides ;
- rejette les demandes invalides de service ;
- traite correctement les erreurs de protocole ;
- manipule correctement les timers ;

Dans cette énumération, il convient de remarquer l'importance des tests correspondant à des cas d'erreurs. Ainsi, l'évaluation du comportement d'une implémentation suite à la réception d'éléments de protocole invalides (tant pour leur contenu que pour leur séquence) s'avère cruciale puisque l'on estime à 40% la part du codage consacré à la vérification de tels cas.

Pour la réalisation des tests de conformité, deux approches peuvent être analysées :

- l'approche **centralisée** (intra-système) pour laquelle l'ensemble des composants de l'architecture de test sont construits et exploités sur le site même où l'implémentation à tester est réalisée ;
- l'approche **décentralisée** (distribuée ou inter-systèmes) où la plupart des composants de l'architecture de test sont développés sur un site distant de celui où se trouve l'implémentation à tester ; site duquel on accède à l'entité à tester via un réseau approprié.

4.1.2. Les centres de test

Avant d'essayer de décrire et d'analyser les techniques de test, il convient de bien prendre en considération les faits suivants :

- les protocoles ISO/OSI sont généralement complexes et la réalisation d'implémentations de ces protocoles s'avère être une tâche non triviale ;
- la réalisation d'un système de test suffisamment rigoureux ainsi que le choix de séquences de test adéquates est au moins aussi complexe que la production d'une implémentation ;
- il n'existe actuellement aucune séquence de test définie de manière standard.

De telles considérations font que les tests de produits ISO deviennent une tâche très difficile qui dépasse la portée de la plupart des utilisateurs.

En terme d'investissements requis, tant techniques que financiers, cette tâche s'avère également hors de portée des fournisseurs du moins des plus petits. C'est pourquoi, la création d'organisations (**centres de test**) spécialisées dans la réalisation de tels tests semble actuellement constituer la solution présentant le meilleur rapport coût-efficacité.

Une telle approche est souvent appelée "**Third party testing**" en ce sens qu'elle regroupe :

- des firmes productrices et fournisseurs de produits intéresséEs dans la réalisation de tests en vue de faciliter le développement et de démontrer que leur produit répond bien aux normes ;
- des utilisateurs/acheteurs de produits ISO intéressés dans la réalisation de tests en vue de la sélection d'un produit (tests d'acceptation) ;
- des organismes (centres de test) indépendants fournissant les services de test.

Outre la réalisation de tests de conformité, de tels centres pourraient fournir :

- un service d'évaluation : afin de vérifier si un produit rencontre les exigences d'un utilisateur notamment concernant les options fournies, les performances (temps de réponse par exemple), la robustesse (aptitude de recouvrement aux situations d'erreurs),...
- un service d'arbitration : étant donné que les test ne peuvent détecter que la présence d'erreurs et non leur absence, il convient qu'un tel centre puisse régler les problèmes pouvant survenir lorsque deux implémentations testées et approuvées par le centre ne peuvent s'interconnecter correctement.

Pour être réellement utilisable et efficace, ces centres devront répondre aux critères suivants :

- crédibilité : le centre lui-même doit être conforme aux protocoles qu'il permet de tester ;
- efficacité des tests menés (exhaustivité des tests menés) : le protocole testé doit présenter un haut degré de fiabilité ;

- applicabilité : pour interconnecter réellement deux systèmes, il faut que chacun possède des protocoles normalisés pour toutes les couches de l'architecture. C'est pourquoi, les techniques de tests doivent être applicables aux protocoles correspondant à tous les niveaux de l'architecture ;
- impartialité et confidentialité ;
- accessibilité : le centre devra être accessible via un réseau ;
- absence de contrainte : le centre ne peut imposer de lourdes contraintes sur l'équipement qu'il teste sous peine de réduire sa généralité et donc son domaine d'application.

En conclusion, à long terme, l'objectif d'une telle démarche serait d'aboutir à la création d'un ou de centres :

- susceptibles de réaliser des tests de conformité pour les protocoles ISO et
- auxquels serait délivrée l'autorisation d'émettre des "certificats d'attestation de conformité" reconnus au niveau international.

4.2. Approche centralisée

On parle de tests centralisés, également appelés "tests intra-système", lorsque le système de test est construit et utilisé sur le même système qu'est développée l'implémentation à tester (notée **IAT**).

Dans le cadre des tests centralisés, deux techniques de tests permettent d'évaluer l'IAT selon deux optiques différentes : l'approche "white box" et l'approche "black box".

4.2.1. Les tests white box

Dans l'approche white box, la structure interne de l'IAT est connue. La technique consiste à repérer les chemins d'exécution de l'algorithme représentant l'IAT et de trouver les données de tests adéquates pour exécuter ces chemins. Un chemin d'exécution est une séquence d'instructions comprenant une instruction d'entrée et une instruction de sortie. La mise en oeuvre de cette technique nécessite cinq étapes :

a. identifier l'ensemble des chemins d'exécution ;

b. pour tout chemin identifié, calculer un prédicat-chemin, c'est-à-dire une condition qui lui est associée et qui doit être satisfaite, avant l'instruction d'entrée, afin que ce chemin soit parcouru ;

c. pour chaque prédicat-chemin, déterminer un jeu de valeurs initiales pour les variables, au point d'entrée du chemin ;

d. pour tout jeu de valeurs, lui associer un jeu de valeurs représentant les résultats attendus, après exécution du chemin ;

e. conduire le test, c'est-à-dire appliquer chacun des jeux de valeurs initiales et comparer le résultat obtenu avec le résultat attendu.

Cependant, en général, il est difficile ou parfois même impossible de définir et de construire l'ensemble des chemins qui sont nécessaires pour couvrir les principales caractéristiques du programme (étape 1 ci-dessus).
Par conséquent, un objectif plus réaliste des tests white box est de rechercher un ensemble minimal de chemins de telle façon que chaque instruction de l'algorithme soit au moins une fois couverte par un de ces chemins et ensuite d'appliquer à chacun de ceux-ci les étapes 2,3,4 et 5 décrites ci-dessus.

Les tests white box sont surtout utiles pour le responsable de l'implémentation lors de la mise au point du programme, avant d'intégrer le module testé avec le reste du système.

4.2.2 Les tests black box

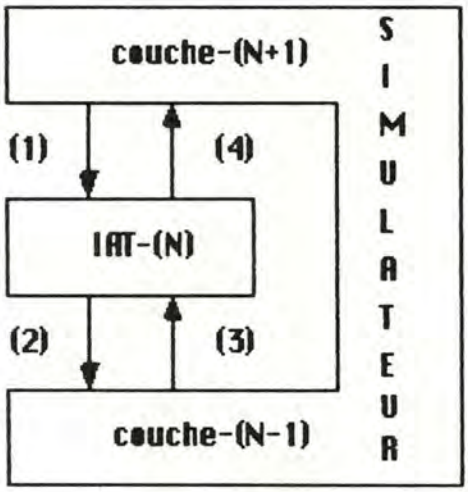
La méthode black box est utilisée dans le but de réaliser des tests basés sur les spécifications.

Dans cette méthode, l'implémentation à tester de niveau-(N) (notée IAT-(N)) est présentée comme une machine abstraite qui n'est accessible que par ses interfaces externes. Il s'agit donc d'isoler le module à tester, représentant l'IAT-(N), de lui injecter tous les types de stimuli et de vérifier son comportement. Comme input, l'IAT accepte les demandes de service-(N) en provenance de la couche-(N+1) et les indications de service-(N-1) envoyés par la couche-(N-1).

Comme output, elle produit des indications de service-(N) à la couche-(N+1) et des demandes de service-(N-1) à la couche-(N-1).

Les jeux de tests sont construits à partir des spécifications de l'IAT-(N), en termes de primitives de service-(N) (demandes) et de service-(N-1) (indications). Ils sont ensuite exécutés sur l'IAT-(N) et l'on observe le comportement de cette implémentation en notant les primitives de service émises à l'intention de la couche-(N) (indications) et de la couche-(N-1) (demandes).

Le comportement observé est comparé au comportement défini dans les spécifications. Un outil permettant de réaliser des tests black box est représenté à la figure 4.1.



(1) demande de service-(N)

(2) demande de service-(N-1)

(3) indication de service-(N-1)

(4) indication de service-(N)

IAT-(N) : Implémentation à tester de niveau N

Figure 4.1 : principes des tests black box.

Le simulateur fournit les services-(N-1) à l'IAT-(N) et se comporte en tant qu'utilisateur des services-(N) de l'IAT-(N). Si l'IAT est de niveau-(N), le simulateur contient donc à la fois la couche-(N-1) et la couche-(N+1).

Bien que l'architecture centralisée donne un premier aperçu du comportement d'un protocole, on préfèrera souvent travailler de manière décentralisée.

Ceci est dû au fait que l'approche centralisée réclame la possibilité d'accès aux deux interfaces de l'IAT (interface vers le haut et interface vers le bas). Or, cette exigence est rarement satisfaite.

Par conséquent, afin d'éviter toute manipulation difficile en vue d'isoler l'IAT, il est utile d'essayer de l'atteindre via un autre système, en utilisant le service-(N-1). (Ainsi, on ne manipulera que l'interface vers le haut, les couches inférieures à la couche testée étant supposées présentes et correctes). Ceci constitue les fondements de l'approche décentralisée.

4.2.3. Proposition d'architecture

Introduction

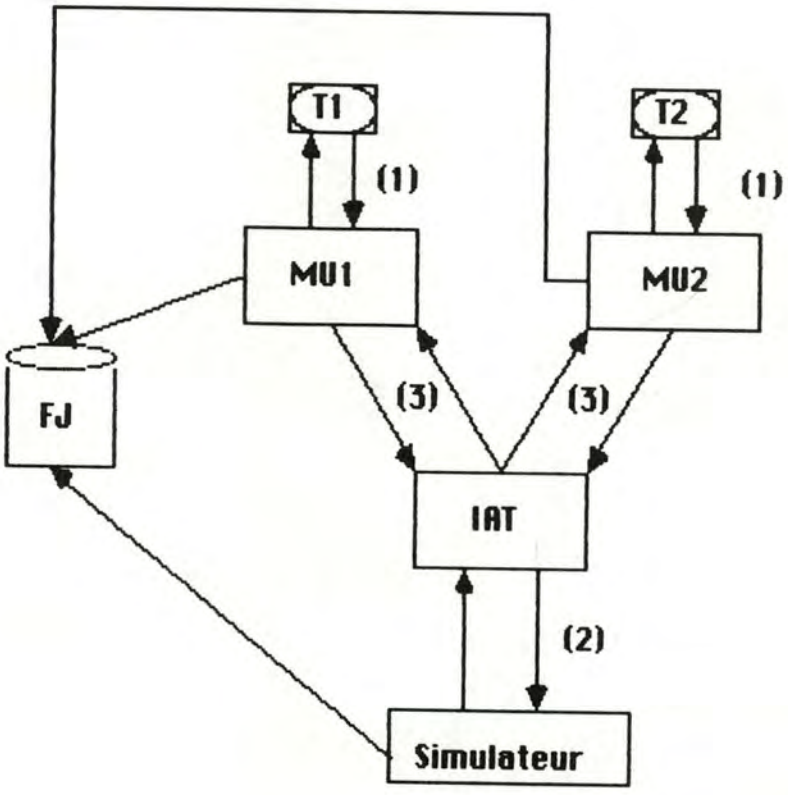
Après avoir énoncé les principes généraux de l'approche centralisée, nous allons maintenant proposer une architecture de ce type.

La présentation prendra la couche réseau (X25/3) comme exemple d'implémentation à tester. L'annexe 1 reprend un résumé succinct des informations nécessaires et suffisantes à la compréhension de ce qui suit. Notons cependant que l'architecture proposée est généralisable en ce sens que ses composants et la fonctionalité de ceux-ci restent les mêmes quelle que soit la couche que l'on désire tester.

Nous définirons tout d'abord une architecture de base qui sera complètée au fur et à mesure qu'apparaîtront de nouveaux besoins.

Les tests interactifs

Un premier objectif que l'on peut assigner aux tests de conformité pour la couche réseau est de vérifier si elle fournit correctement les services pour lesquels elle a été conçue et ce, dans un environnement sans erreur. Ceci signifie qu'à ce niveau, on introduit aucune erreur tant du point de vue utilisateur (couche 4) que du point de vue serveur (couche 2). La figure 4.2 illustre une architecture pouvant être utilisée pour la réalisation de tels tests. Celle-ci comporte trois composants qui sont les modules utilisateurs, l'implémentation à tester (IAT) et un simulateur.



Ti : Terminaux de contrôle
MUi : Modules utilisateurs
IAT : Implémentation à tester
FJ : Fichier journal

→ : messages échangés

Informations stockées :

(1) : les commandes de l'opérateur
(2) : les demandes de service de l'IAT destinées au simulateur
(3) : les réponses fournies par l'IAT aux MU

Figure 4.2 : architecture de base pour les tests interactifs.

Les **modules utilisateurs** ont pour rôle :

- de fournir des demandes de service à l'implémentation à tester afin d'évaluer son comportement. Pour ce faire, ils prennent en charge les commandes introduites par la personne chargée de la réalisation des tests (que l'on appellera par la suite opérateur) via son terminal. Ils transforment celles-ci en demandes de service et transmettent ces demandes de service à la couche 3 via son interface vers le haut ;
- de traiter les réponses fournies par l' IAT. Pour ce faire, ils recueillent les réponses qui leur sont fournies par la couche 3, les transforment en messages pour l'opérateur et les impriment au terminal de l'opérateur.

Les transformations de commandes-utilisateur en demandes de service (et vice-versa) sont dûes aux différences de syntaxe pouvant exister entre elles. De plus, certains paramètres des primitives d'interface pourront se voir assigner par l'opérateur une valeur par défaut, ce qui dispensera de mentionner ces paramètres explicitement au niveau du langage de commande. L'objectif est de simplifier au maximum le langage de commande et de le rendre plus facilement utilisable.

L'**implémentation à tester** - couche 3 dans l'exemple - est le composant qui va être étudié grâce à son interface vers le haut et son interface vers le bas.

Le **simulateur** agit en lieu et place des couches inférieures à l'IAT. Recevant des demandes de service de l'IAT, il doit y répondre sans introduire d'erreur dans les données ni de délai dans la transmission. Deux approches sont envisageables quant aux traitements à réaliser suite à la réception d'une demande de service par la couche 2 qui est ici simulée.

Une première approche considère que la demande de service a été acheminée vers l'entité paire et que celle-ci y a répondu. C'est cette réponse qui est retournée à l'IAT. Ainsi, si l'IAT émet une demande de service consistant en la transmission d'un paquet d'appel (ouverture de connexion), la réponse fournie sera un paquet d'acceptation de connexion. Cette première approche est faible car elle ne permet pas de tester la réception par l'IAT d'événements provenant de l'entité paire, tel qu'un paquet d'appel entrant.

La seconde approche constitue une solution à cet inconvénient. Celle-ci consiste à faire jouer un double rôle à l'IAT en ce sens qu'elle est également considérée comme l'entité paire.

Ainsi, si l'IAT émet une demande de service consistant en la transmission d'un paquet d'appel, la réponse fournie sera la transmission vers cette même IAT d'un paquet d'appel entrant. Dans la suite, il ne sera plus question que de cette seconde approche.

Les modules utilisateurs et le simulateur pourront sauver sur un support externe (disque, bande magnétique) un ensemble d'informations pertinentes pour l'analyse ultérieure des résultats des tests réalisés. Les informations à sauver seront les commandes émises par l'opérateur, les demandes de service émises par l'IAT et destinées au simulateur et les réponses fournies par l'IAT aux modules utilisateurs. Autrement dit, les informations stockées sont les commandes de l'opérateur et les outputs de l'IAT. Ces informations sont reprises à la figure 4.2.

Ce qui suit montre la manière dont un test peut être mené sur une telle architecture.

Terminal A	Terminal B
connect (b)	- - - - - - - -
- - - - - - - -	CONNECT INDICATION FROM A
	accept_con
CONNECTION ACCEPTED	- - - - - - - -
send_data "text of message"	- - - - - - - -
- - - - - - - -	DATA INDICATION
- - - - - - - -	accept_data
- - - - - - - -	TEXT : "text of message"
disconnect (b)	DISCONNECT INDICATION

Cet exemple permet de vérifier si une connexion réseau peut être établie, si des données peuvent être échangées sur celle-ci et si la connexion peut être fermée. Les lettres minuscules représentent ce qui est introduit par l'opérateur et les lettres majuscules l'output des modules utilisateurs. Si l'on considère la commande connect(b), celle-ci est analysée par le module utilisateur associé au terminal A et résulte en une demande d'établissement de connexion-réseau transmise à l'IAT.

Si le comportement de l'IAT est celui attendu, un paquet d'appel doit être transmis vers le simulateur. Celui-ci y répondra par l'émission d'un paquet d'appel entrant. Analysé par l'IAT, ce paquet doit normalement résulter en une indication de demande d'établissement de connexion transmise vers le niveau supérieur (couche 4).

Cette indication sera interceptée par le module utilisateur associé au terminal B qui imprimera une réponse sur ce terminal (CONNECT INDICATION FROM A). La même méthode peut être utilisée pour le traitement des autres commandes (la commande "accept_data provoque l'impression du message arrivé TEXT : "text of message"). En opérant de cette manière, lorsque l'utilisateur détecte une erreur à son terminal, il peut réaliser une étude plus approfondie des causes de cette erreur en consultant le fichier créé sur support externe.

Il est à noter que le langage de commandes utilisé dans cet exemple est réduit à sa plus simple expression. Seuls quelques paramètres doivent être mentionnés explicitement dans les commandes, les autres ayant une valeur par défaut. Ainsi, la commande associée à une demande d'établissement de connexion a la syntaxe suivante : CONNECT (<adresse appelé>) alors que la primitive associée peut avoir la syntaxe suivante : CONNECT (<adresse appelant> <adresse appelé> <qualité du service demandé> <message associé> <longueur du message associé> <utilisation du service de données express> <identifiant connexion établie>) . Cette simplification résulte en une facilité d'utilisation mais limite les tests pouvant être demandés puisqu'aucune manipulation des paramètres implicites (c'est-à-dire auxquels on a associé une valeur par défaut) n'est possible.

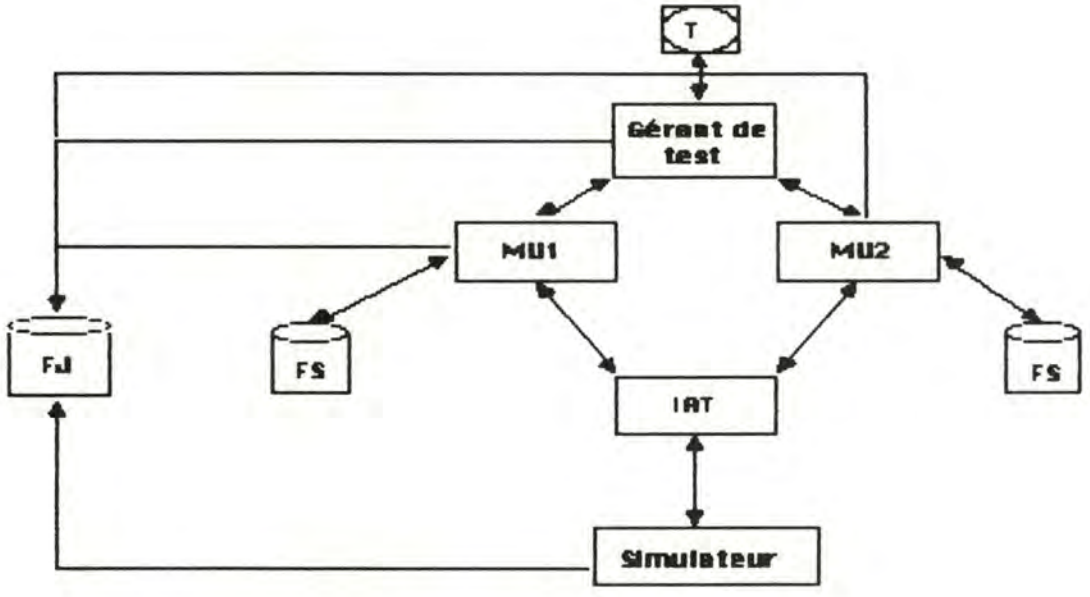
Les tests réalisés manuellement selon la méthode venant d'être décrite, sont limités quant à leur étendue. En effet, il devient vite fastidieux et coûteux en temps d'opérer de la sorte. L'étape suivante dans notre étude d'une architecture centralisée -les tests automatisés- constitue dès lors une extension logique à cette méthode.

L'automatisation des tests

Certains tests nécessaires à la vérification de conformité peuvent être très longs. D'autres peuvent être répétitifs en ce sens qu'ils utilisent les mêmes demandes de service mais avec des paramètres différents. Il est également possible d'envisager que certaines combinaisons de demandes de service se retrouvent dans de nombreuses situations (c'est notamment le cas pour les demandes de service de la phase d'établissement d'une connexion). L'ensemble de ces exigences et les inconvénients de l'approche interactive font qu'il est approprié de recourir à une automatisation des tests.

La figure 4.3 illustre les modifications à apporter à l'architecture de base (de la figure 4.2) pour pouvoir automatiser les tests. Les terminaux utilisés pour l'introduction des demandes de service dans l'approche interactive sont remplacés par des fichiers comprenant des **scénarios** de test. Ces scénarios sont constitués d'un ensemble de commandes représentant les primitives d'interface entre l'IAT et un module utilisateur. On trouve également dans cette architecture un composant appelé **gérant de test** auquel est associé un terminal de contrôle. Celui-ci constitue une interface entre l'opérateur et les modules utilisateurs. C'est via cette interface que l'on offrira la possibilité de choisir les scénarios à exécuter, de lancer leur exécution sur un module utilisateur donné, de contrôler et de stopper l'exécution de tests.

Chaque test est défini par deux scénarios complémentaires qui seront traités chacun par un module utilisateur. Ces deux scénarios sont dits complémentaires en ce sens que l'on trouve dans l'un, l'ensemble des primitives d'interface à transmettre d'un module utilisateur à l'IAT et dans l'autre, les réponses qui en découlent pour le partenaire et qui seront transmises entre un second module utilisateur et l'IAT (considérée dans ce cas comme entité paire).



T : Terminal de contrôle
MUi : Module utilisateur
IAT : Implémentation à tester
FJ : Fichier journal
FS : Fichiers scénarios
<-> : messages échangés

Figure 4.3 : architecture pour les tests automatisés (environnement sans erreur).

Les deux scénarios figurant ci-dessous constituent un exemple de test dans lequel on désire créer une connexion, échanger quelques données et fermer la connexion. Pour simplifier l'exposé, aucun paramètre n'y est spécifié. En réalité, de nombreuses variantes sont envisageables ; les deux extrêmes étant de ne mentionner aucun paramètre (facilité d'emploi mais limitation dans la manipulation de ces paramètres) ou de mentionner l'ensemble des paramètres (emploi moins facile mais actions possibles sur les paramètres plus complètes). Dans le cas où un paramètre n'est pas fixé explicitement, une valeur par défaut lui est associée.

Scénario A	Scénario B
T : CONNECT REQUEST	R : CONNECT INDICATION
R : CONNECT CONFIRM	T : CONNECT RESPONSE
T : DATA REQUEST	R : DATA INDICATION
T : DISCONNECT REQUEST	R : DISCONNECT INDICATION
R : DISCONNECT CONFIRMATION	T : DISCONNECT RESPONSE
EXIT	EXIT

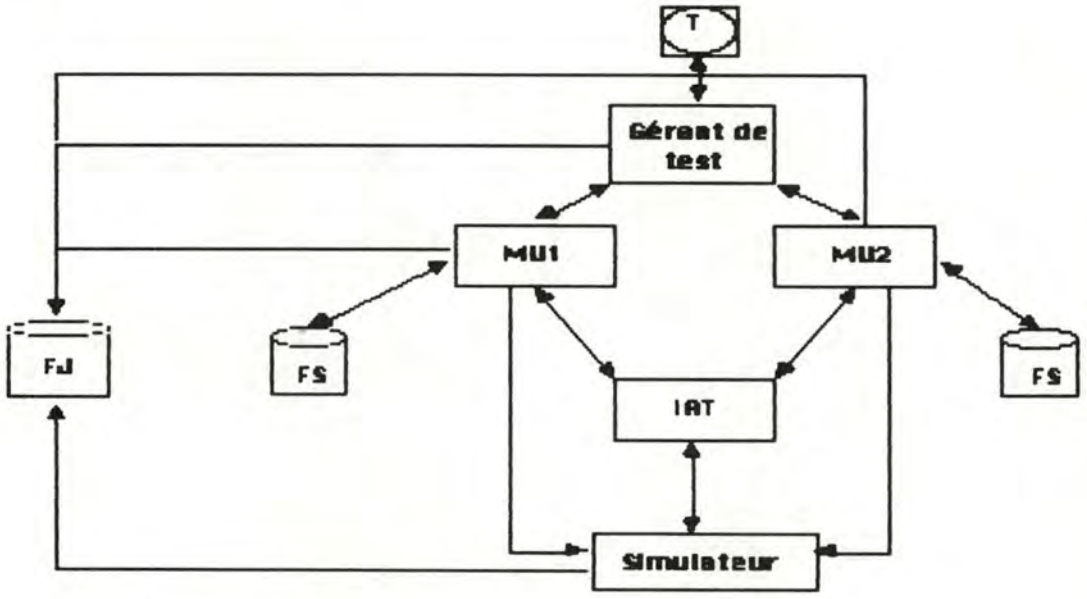
Les scénarios choisis par l'opérateur parmi un ensemble de scénarios possibles sont traités en parallèle, le traitement consistant en une analyse commande par commande. Si la commande analysée est précédée de T:(Transmettre), le traitement consiste à créer la primitive d'interface correspondante et à l'envoyer vers l'IAT. Une fois cette opération terminée, on passe à l'analyse de la commande suivante. Dans le cas d'une commande précédée de R : (Recevoir), rien n'est généré mais on attend la reception de la primitive correspondante de l'IAT. Si la bonne primitive est reçue mais avec de mauvais paramètres, ou si une mauvaise primitive est reçue ou si rien n'est reçu endéans un certain temps (timer), c'est qu'une erreur s'est produite et le test est clôturé (tout en ayant pris soin de sauver un ensemble d' informations comme dans l'approche interactive). Si la bonne primitive est reçue dans les délais impartis, on passe à l'analyse de la commande suivante. On opère de la sorte jusqu'à atteindre la commande EXIT dans les deux scénarios, commande qui marque la fin du test.

Introduction d'erreurs dans l'environnement de l'IAT

Jusqu'à présent, nous n'avons cherché à évaluer le comportement de l'IAT que dans le cas d'un environnement sans erreur.

Ceci constitue une vue un peu simpliste de la réalité. En effet, dans la pratique, deux types d'erreurs peuvent survenir à savoir des erreurs dans les demandes de service transmises à l'IAT (erreur dans le(s) paramètre(s) ou dans la séquence des demandes de service) ou des erreurs dans les services fournis à l'IAT.

En ce qui concerne la transmission vers l'IAT de demandes de services erronées, aucune modification de l'architecture 4.3 n'est nécessaire pour les prendre en considération. En effet, il suffit pour cela de construire des fichiers scénarios comportant de telles demandes. On pourra ainsi aisément introduire des demandes de service avec de mauvais paramètres ou de mauvaises combinaisons de demandes de service.



T : Terminal de contrôle
MUI : Module utilisateur
IAT : Implémentation à tester
FJ : Fichier journal
FS : Fichiers scénarios
 : messages échangés

Figure 4.4 : architecture pour les tests automatisés (environnement avec erreurs).

Pour introduire des erreurs au niveau du service fourni à l'IAT, il est nécessaire que le simulateur soit paramètrable. En effet, pour tester certains cas d'erreur, on doit pouvoir indiquer à ce simulateur d'introduire une erreur déterminée sur une information précise. Les erreurs envisageables sont des délais dans la transmission, des pertes de paquets, des altérations du contenu de paquets, des paquets dupliqués, etc .... Cette paramètrisation du simulateur peut être réalisée selon deux approches. Une première approche consiste à paramètrer le simulateur via le gérant de test. Ceci signifie que l'opérateur introduit à son terminal de contrôle les commandes qu'il désire envoyer au simulateur. Mais cette approche pose de gros problèmes de synchronisation avec l'exécution des scénarios et souffre de la même lourdeur que les tests interactifs (introduction des commandes fastidieuse).

D'où la seconde approche qui consiste à commander le simulateur via les scénarios. Comme le montre l'exemple ci-dessous, on introduit dans ces scénarios un troisième type de commandes qui seront précédées de S : (Simulateur). Le traitement de ce type de commande consistera en la transmission d'une commande vers le simulateur. L'exemple présenté consiste à tester si le timer associé à l'envoi d'un paquet d'appel (CALL REQUEST) et la possibilité de retransmettre en cas d'absence de réponse sont bien implémentés. Pour ce faire, on demande au simulateur d'ignorer quatre CALL REQUEST et donc de n'envoyer de réponse qu'à la réception du cinquième.

Scénario A	Scénario B
S : SUP 4 CALL REQUEST	- - - - - - - -
T : CONNECT REQUEST	R : CONNECT INDICATION
R : CONNECT CONFIRMATION	T : CONNECT RESPONSE

Conclusion

En résumé, l'architecture venant d'être présentée permet de tester, sur un seul site et de manière relativement complète une implémentation de protocole. Ainsi, elle permet d'évaluer le comportement de cette implémentation suite à la réception de demandes de service correctes et erronées, suite à la réception d'éléments de protocole corrects et erronés. Elle permet également de tester si le mécanisme des timers fonctionne correctement.

Cependant, elle présente deux inconvénients spécifiques à l'approche centralisée. Elle nécessite d'avoir accès aux deux interfaces de l'IAT ce qui peut constituer une contrainte dans certains cas. Elle envisage l'IAT indépendamment de l'environnement dans lequel elle devra être intégrée. Il faudra donc, lors de l'intégration, compléter l'analyse réalisée sur l'IAT par un ensemble de tests d'intégration. C'est en fait ces deux inconvénients qui justifient l'utilisation d'architectures décentralisées.

4.3. Approche décentralisée

4.3.1. Principes

Les principes de base de l'approche décentralisée (distribuée, inter-systèmes) sont d'une part d'envisager l'entité à tester comme une boîte noire (approche black box) uniquement accesssible via son interface vers le haut, et d'autre part de considérer le système de test comme étant implémenté sur un site distant du site où est développée l'implémentation à tester. Sur base de ces principes, on peut définir une architecture physique de base pour les protocoles de haut niveau (niveau 4 et supérieurs) qui constituent des protocoles de bout en bout. Cette architecture physique est illustrée à la figure 4.5.

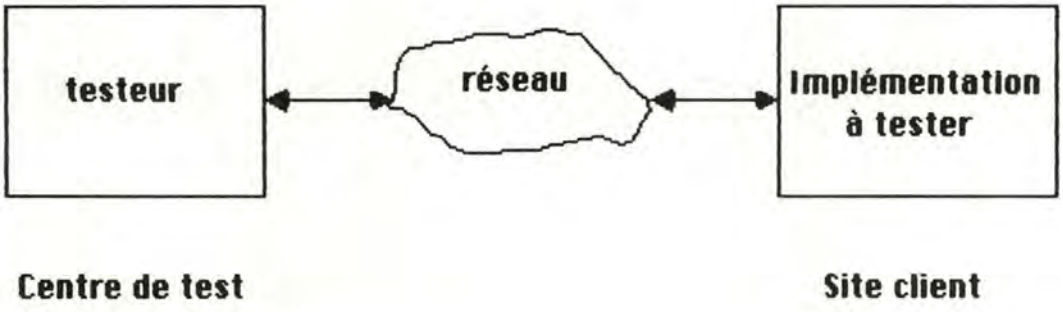


Figure 4.5 : architecture physique de base pour les tests centralisés.

Dans celle-ci, le **testeur** est implémenté sur un site "centre de test" et le **système client** est testé à distance via un moyen de communication tel qu'un réseau public de transmission de données.

Lorsque le protocole à tester n'est pas un protocole de bout en bout, comme c'est le cas pour le protocole X25/NIV3 émis par le CCITT par exemple, l'architecture de la figure 4.5 n'est plus applicable. En effet, si le protocole n'est pas de bout en bout, la partie information de contrôle des éléments de protocole a une signification pour les noeuds intermédiaires du réseau et est donc utilisée par ceux-ci. Dès lors, il devient impossible de tester un cas d'erreur à partir du testeur étant donné que cette erreur sera détectée au niveau des noeuds et ne parviendra jamais au système que l'on désire tester. Ainsi en est-il d'une demande d'établissement de connexion du niveau 3 sans adresse qui sera bloquée dès le noeud d'entrée dans le réseau.

Diverses solutions permettent de remédier à cet inconvénient. Ainsi, le chapitre 6 sera consacré entièrement à l'étude d'architectures permettant de tester les protocoles de bas niveau (niveau 3 et 2).

L'objectif des sections suivantes sera d'étudier divers outils et méthodes proposés pour réaliser des tests de conformité pour les protocoles de haut niveau. Une analyse des architectures logiques décrivant les composants logiciels des différents sites sera proposée.

4.3.2. L'architecture du NBS

4.3.2.1. Introduction

L'"Institute for Computer Science and Technology" (ICST) du "National Bureau of Standards" (NBS, USA) poursuit un programme visant le développement et la mise au point de protocoles standards de communication, appelés "Federal Information Processing Standards" (FIPS). L'objectif de ce programme est de rendre possible la communication entre systèmes distribués. Les FIPS constituent donc la condition nécessaire à cette communication. Ils sont conformes au modèle de référence ISO/OSI (décrit en 2.2)

Parallèlement à cet effort de standardisation, une architecture et des outils de test sont développés depuis 1981, afin de vérifier si les implémentations de protocoles résidant dans les couches définies par les FIPS sont bien conformes aux spécifications données par ces FIPS.

4.3.2.2. Graphe de l'architecture

Hypothèses

- Comme le comportement de l'implémentation à tester (IAT) est indéterminé, l'introduction d'un protocole de gestion de tests entre les points terminaux (c'est-à-dire, d'une part, l'utilisateur de l'implémentation de référence, et d'autre part, l'utilisateur de l'implémentation à tester) s'avère peu fiable. Par conséquent, un moyen de communication parallèle, le plus souvent téléphonique, est utilisé entre les opérateurs sur les deux sites, pour synchroniser l'exécution de ces tests.

- Des fichiers de texte peuvent être échangés entre le centre de test et le site client, sur lequel figure l'implémentation à tester, dans le but de communiquer au site client les tests à réaliser et de recevoir les résultats de ceux-ci. Ces échanges se font au moyen de supports magnétiques ou bien, lorsque le test de l'implémentation à tester a été réalisé et s'est révélé concluant, au moyen d'un système de transfert de fichiers existant dans l'environnement de test.

- Les autres hypothèses concernent les caractéristiques du système sur lequel est développée l'implémentation à tester :

a. possibilité de stockage d'informations sur mémoires auxiliaires,

b. utilisation d'une horloge pour les mécanismes de timer et pour pouvoir dater dans le fichier journal le moment de survenance des événements,

c. présence d'une interface opérateur utilisateur.

Architecture

L'architecture de test a été conçue en tenant compte des contraintes décrites ci-dessus. La figure 4.6 représente les différents composants de cette architecture et les relations qui les unissent. Ces composants seront décrits dans la section suivante.

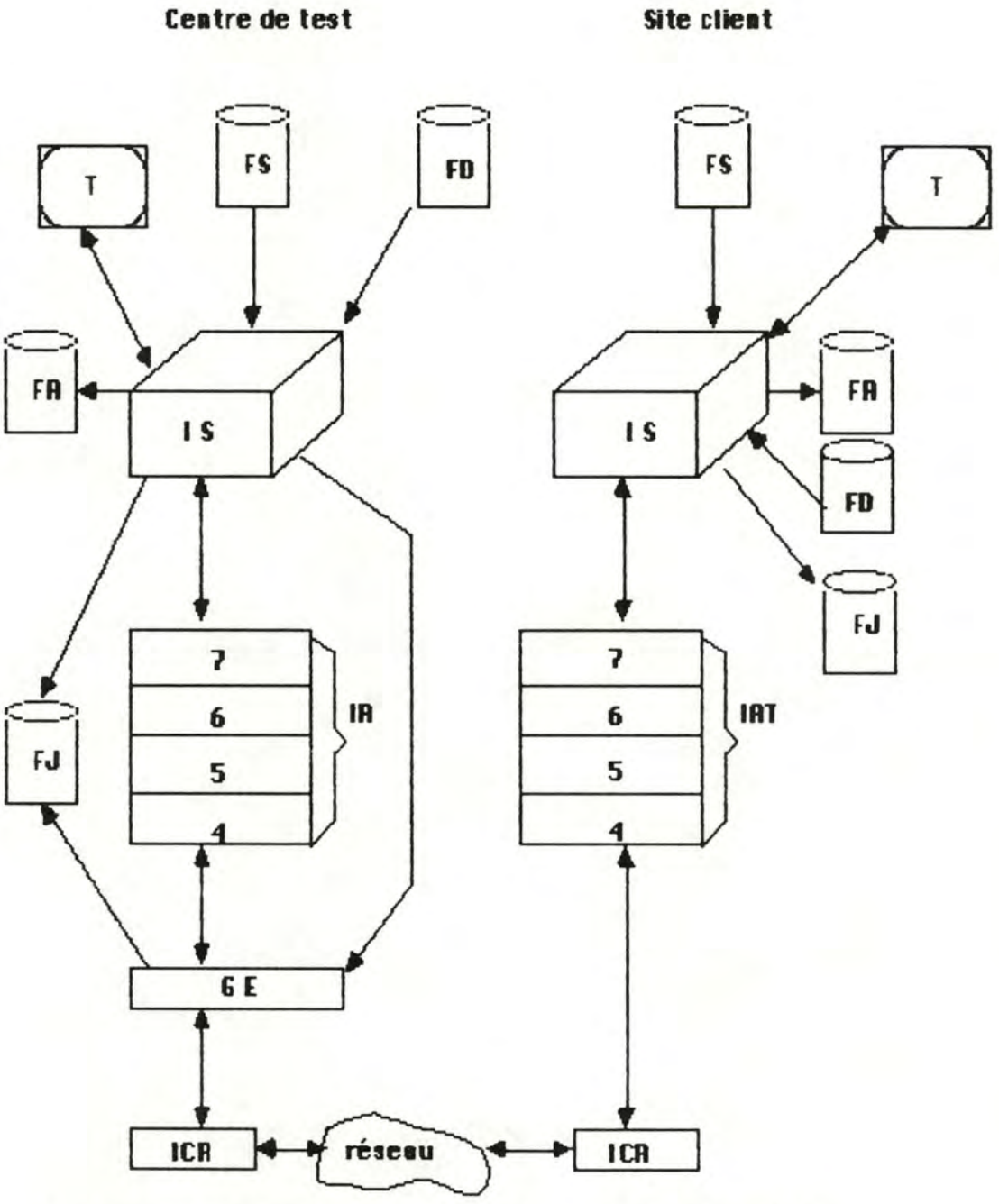


Figure 4.6 : architecture du NBS.

4.3.2.3. Analyse des composants de l'architecture

Composants du centre de test

1. Interpréteur de scénarios (IS) du centre de test

L'IS du centre de test exécute des fichiers scénarios, lesquels contiennent des commandes permettant de guider l'interface supérieure de l'implémentation de référence et le générateur d'exceptions, décrits au point 2 et 3 ci-dessous. Il possède une interface par laquelle l'opérateur dirige la session de tests. Il tient également à jour :

(1) un fichier journal reprenant l'historique de la session de tests, (une **session** de tests est une séquence de commandes relatives au test d'un protocole et prises en charge par l'interpréteur de scénarios après la mise en activité de celui-ci) ;
(2) un fichier des résultats reprenant la liste des tests exécutés ;
(3) des fichiers de données qui contiennent pour chacune des connexions établies les données reçues sur cette connexion.

2. Implémentation de référence (IR)

Le rôle de l'IR est de prendre en charge les requêtes de service demandées par l'interpréteur de scénarios. Une grande partie de cette implémentation peut être générée automatiquement à partir de la description formelle du protocole (voir section 3.3.2). Les scénarios de tests ont d'abord été utilisés sur deux copies communiquant entre elles de cette implémentation avant d'être employés pour tester réellement une nouvelle implémentation. Cela permet au testeur qui va utiliser cette implémentation de référence d'être sûr qu'elle fonctionne correctement (c'est-à-dire qu'elle ne génère que des éléments de protocole valides et qu'elle présente un haut degré de fiabilité quant au respect des spécifications du protocole).

Comme l'architecture est prévue pour tester toute implémentation des couches figurant entre la quatrième et la septième couche du modèle ISO, les implémentations de référence des couches 4,5,6 et 7 sont reprises dans cette architecture. Lors d'un test de conformité, l'implémentation de référence de la couche testée est complétée par les implémentations des couches inférieures à celle testée.

Ainsi, si l'on doit tester la conformité d'une implémentation de la couche 5 (session), les implémentations de référence des couches 5 et 4 (transport) seront utilisées.

3. Générateur d'exceptions (GE)

Le GE est situé entre les couches transport et réseau. Il traite les éléments de protocole à destination ou en provenance de l'implémentation de référence. Il offre la possibilité de générer des erreurs de protocole et de simuler certaines erreurs qui pourraient être induites par la couche-(N-1) et ce, afin de vérifier la réaction de l'implémentation à tester face à de tels événements. C'est en effet le seul moyen de tester le comportement d'une implémentation face aux erreurs, étant donné que l'implémentation de référence est supposée correcte et ne peut donc générer d'erreurs de protocole. Les commandes offertes par le générateur d'exceptions permettent de supprimer, dupliquer, altérer (insérer, enlever, changer les valeurs de certains champs) les éléments de protocole et de modifier l'ordre d'acheminement de ceux-ci. Les commandes pouvant être transmises au générateur d'exceptions lui sont transmises par l'interpréteur de scénarios grâce à un moyen de communication interne (par exemple par un canal inter-processus). Tous les éléments de protocole véhiculés entre la couche transport et la couche réseau peuvent être enregistrés dans le fichier journal par le générateur d'exceptions.

4. Interface vers la couche réseau (ICR)

L' ICR fait partie de la couche 3 du modèle ISO. Son rôle est d'établir une correspondance entre les services demandés par la couche transport et ceux disponibles sur le réseau employé (public ou privé).

Composants du site client

1. Interpréteur de scénarios (IS) du site client

Les fonctions de cet IS constituent un sous-ensemble des fonctions réalisées par l'IS du centre de test. L' IS du site client exécute des fichiers scénarios, commande l'interface supérieure de l'implémentation à tester au moyen des primitives de service et prend en charge l'interface opérateur.

Les commandes du scénario sont complémentaires à celles exécutées en même temps par l'interpréteur de scénarios du centre de test. Un fichier journal sur l'activité de l'interpréteur de scénarios du site client est également tenu à jour, de même que le fichier résultats des tests menés au cours de la session et les fichiers de données.

2. Implémentation à tester (IAT)

L'IAT est l'implémentation d'un protocole défini dans le cadre du modèle de référence ISO et pour laquelle on désire établir la conformité par rapport aux FIPS, Federal Information Processing Standards. Les implémentations des couches inférieures à l'IAT sont supposées testées et correctes. Comme cela a déjà été signalé précédemment, l'IAT peut être l'implémentation d'un protocole des couches 4, 5, 6 ou 7 du modèle ISO/OSI.

3. Interface vers la couche réseau (ICR)

Cette interface vers la couche réseau joue le même rôle que celle décrite dans le centre de test.

Contenu des fichiers utilisés

1. Fichier journal

Toute entrée du fichier contient :

- la mention de l'heure à laquelle s'est terminée une commande confiée à l'interpréteur de scénarios ;
- le nom de cette commande et les paramètres associés ;

2. Fichier des résultats

Ce fichier des résultats fournit un ensemble d'informations concernant le résultat des tests. Toute entrée du fichier :

- contient le moment de terminaison du test ;
- donne la manière suivant laquelle le test s'est terminé : FINISHED (s'il s'est exécuté complètement) ou ABORTED (s'il a dû être interrompu).

4.3.2.4. Méthodes de réalisation des tests

Commandes de scénarios

Un test est engagé à l'aide de l'interface opérateur de l'interpréteur de scénarios et consiste à sélectionner sur le site client (respectivement, le centre de test) un fichier contenant des commandes permettant d'utiliser l'interface vers le haut de l'implémentation à tester (respectivement, de référence) sous-jacente. Les commandes sont exécutées séquentiellement par l'interpréteur de scénarios et peuvent être de trois sortes.

1. commandes de primitives de service

Les commandes de primitives de service sont de quatre types : REQUEST, RESPONSE, INDICATION et CONFIRM.

Les commandes de type "**request**" et "**response**", appelées commandes **actives** demandent à l'interpréteur de scénarios de générer une primitive de service correspondante avec les paramètres donnés (voir le tableau A1 en annexe).

Dans le cas du test d'un protocole de transport, ces commandes sont les suivantes : connect request, connect response, disconnect request, data request, expedited request.

Tant qu'une commande n'a pas pu être exécutée (pour des raisons de contrôle de flux avec l'implémentation, par exemple), la lecture des commandes suivantes du scénario est interrompue. La fin de l'exécution de la commande est signalée dans le fichier journal.

Les commandes de type "**indication**" et "**confirm**", appelées commandes **passives**, signalent à l'interpréteur de scénarios qu'il doit s'attendre à recevoir, de l'implémentation sous-jacente, la primitive correspondante. Si la commande concerne l'établissement ou la terminaison d'une connexion, la lecture du scénario est interrompue jusqu'à la réception de l'événement attendu (voir le tableau A1 en annexe). Dans le cas du test d'un protocole de transport, ces commandes sont les suivantes : connect indication, connect confirm, data indication, expedited indication et disconnect indication.

Le tableau ci-dessous donne l'ensemble des commandes actives et des commandes passives complémentaires, ainsi que les primitives de service correspondantes. Pour plus d'informations sur la fonctionalité de chacune de ces primitives, on peut consulter la deuxième partie de l'annexe 1 consacrée aux primitives du service transport.

Commandes actives	Commandes passives complémentaires
CONNECT REQUEST <--> T_CONNECT.request	CONNECT INDICATION <--> T_CONNECT.indication
CONNECT CONFIRM <--> T_CONNECT.confirm	CONNECT CONFIRM <--> T_CONNECT.confirm
DATA REQUEST <--> T_DATA.request	DATA INDICATION <--> T_DATA.indication
EXPEDITED REQUEST <--> T_EXPEDITED.request	EXPEDITED INDICATION <--> T_EXPEDITED.indication
DISCONNECT REQUEST <--> T_DISCONNECT.request	DISCONNECT INDICATION <--> T_DISCONNECT.indication

Lorsque l'événement qui survient n'est pas celui attendu par l'interpréteur de scénarios, l'erreur est mentionnée dans le fichier journal. Seules les commandes relatives à la phase de transfert de données (data indication, expedited indication) autorise l'interpréteur à lire les commandes suivantes du scénario, pour autant qu'il s'agisse toujours de commandes relatives au transfert de données. En effet, il est difficile de prévoir l'ordre d'arrivée des données lorsque le scénario complémentaire contient à la fois des commandes "data request" et "expedited request" (étant donné que les données envoyées au moyen d'un "expedited request" seront acheminées plus rapidement au destinataire).

C'est la raison pour laquelle on autorise l'interpréteur de scénarios à lire la commande suivante et à la traiter s'il s'agit toujours d'une commande relative au transfert des données. Cela lui permet d'éviter une attente passive des données qui pourraient éventuellement être retardées dans leur transfert.

2. commandes directives à l'interpréteur de scénarios

log : inscrit dans le fichier journal la chaîne de caractères donnée comme argument.

consecutive : invoque le fichier scénario donné en paramètre. Le scénario maître dans lequel figure une commande "consecutive" est suspendu tant que le scénario mentionné dans la commande ne s'est pas terminé.

listen : suspend la lecture du scénario pour une durée fixée, tout en permettant la réception de primitives pouvant être rattachées à des commandes "indication" relatives au transfert de données et qui ne sont pas encore satisfaites.

synchronise : suspend l'exécution du scénario jusque la survenance de l'événement donné en paramètre.

3. commandes destinées au générateur d'exceptions

Ces commandes ne peuvent être mentionnées que dans des scénarios destinés à être exécutés par l'interpréteur de scénarios du centre de test. En effet, il n'est pas prévu d'utiliser un générateur d'exceptions sur le site client.

exemples :

EG INLIST SUPPRESS DT ; demande au générateur d'exceptions d'ignorer le prochain élément de protocole (DT) entrant contenant des données ;

EG OUTLIST REMOVE CR VARIABLE CHEKSUM ; demande au générateur d'exceptions de supprimer le champ "cheksum" du prochain élément de protocole (CR) sortant et représentant une demande de connexion ;

Commandes de gestion des tests

Contrairement aux commandes précédentes, ces commandes ne figurent pas dans les scénarios mais sont envoyées à l'interpréteur de scénarios par l'interface opérateur. Ces commandes sont les suivantes :

next : a pour effet d'initialiser un nouveau test en exécutant le scénario spécifié. Les scénarios complémentaires à exécuter simultanément sont communiqués par les opérateurs du centre de test et du site client aux interpréteurs de scénarios de ces deux sites, au moyen de la commande Next suivie du nom du fichier à exécuter et du numéro de machine de test que l'on associe à ce test. Une machine de test est un ensemble de données qui donne en permanence l'état et les caractéristiques du test associé. Une machine reste allouée à un test jusqu'à la fin (normale ou anormale) de ce test.

abort : termine le test associé au numéro de machine de test donné en paramètre.

end : termine une session de test.

view : affiche à l'écran le fichier dont le nom est donné en paramètre.

Déroulement d'un test sur base d'un exemple

Supposons que l'implémentation à tester soit l'implémentation d'un protocole de transport (niveau 4). Le test suivant consiste à établir une connexion et à la fermer immédiatement. Les scénarios complémentaires correspondants sont init_1 (pour l'initiateur de la connexion transport) et recv_1 (pour le partenaire).

```
Init_1

LOG "test open and close connection after receiving 15 bytes data,
     initiator"
CONNECT REQUEST 1 2
CONNECT CONFIRM
DATA INDICATION 15
DISCONNECT REQUEST
```

```
Recv_1

LOG "test open and close connection after sending 15 bytes data,
     receiver"
CONNECT INDICATION 2 1
CONNECT RESPONSE
DATA REQUEST 15
DISCONNECT INDICATION
```

Ces scénarios devront être exécutés simultanément, l'un sur le site du centre de test, l'autre sur le site client. Les exécutions seront lancées par la commande Next.

Si on analyse le déroulement du scénario init_1, dans le cas où l'exécution de celui-ci a lieu sur le centre de test, on constate que :

- la commande CONNECT REQUEST est lue par l'interpréteur de scénarios ; les nombres 1 et 2 constituent des index dans une table d'adresse. Les entrées 1 et 2 dans cette table donnent respectivement les adresses réelles de destination et d'origine.

Cette table évite au responsable des tests la laborieuse tâche de modifier tous les scénarios afin de tenir compte des adresses réelles du site client. Seule un modification du contenu de la table est nécessaire avant de tester une implémentation sur un nouveau site. L'interpréteur de scénarios invoque la primitive de service correspondante (ici T_CONNECT.request, pour la couche transport) avec comme paramètres les adresses (origine et destination) obtenues dans la table. Les paramètres non spécifiés dans le scénario recevront une valeur par défaut. Si la primitive a été acceptée par l'implémentation de référence, l'interpréteur de scénarios enregistre l'événement dans le journal et lit la commande suivante du scénario.

- la commande CONNECT CONFIRM oblige l'interpréteur de scénarios à attendre la réception d'une primitive T_CONNECT.confirm avant de poursuivre la lecture du scénario. Si l'événement attendu se produit avec les paramètres spécifiés, (bien que dans cette commande aucun paramètre ne soit mentionné) l'interpréteur enregistre dans le fichier journal cet événement ainsi que l'heure de terminaison de celui-ci et la lecture du scénario se poursuit. Sinon le test est abandonné et la mention "ABORTED" est inscrite dans le fichier des résultats.

- la commande DATA INDICATION 15 signale à l'interpréteur l'arrivée de 15 octets de données. L'interpréteur génère la quantité de données spécifiée et ce, selon le même algorithme utilisé par l'émetteur lors de l'envoi de celles-ci. Il comparera ainsi les données qu'il a générées avec celles réellement reçues. Si les données reçues diffèrent des données attendues, le test est abandonné, et il y a écriture dans le fichier journal de l'événement et des deux ensembles de données. En outre, les données reçues sont enregistrées dans le fichier des données associées à la connexion sur laquelle la réception a eu lieu.

- après avoir exécuté la dernière commande (DISCONNECT REQUEST), l'interpréteur de scénario signale la fin du test dans le fichier des resultats, avec la mention "FINISHED".

Pour avoir une idée plus précise de la diversité des scénarios de test, le lecteur pourra consulter l'annexe 2 où sont repris la plupart des scénarios utilisés lors du test de conformité d'une implémentation d'un protocole de niveau 4.

4.3.3. L'architecture du National Physical Laboratory (NPL)

4.3.3.1. Introduction

Depuis avril 1980, le "National Physical Laboratory" (NPL) développe des techniques de test pour les implémentations de protocole. Son objectif est de mettre au point des outils et méthodes de test applicables non seulement aux protocoles existants mais également aux nouveaux protocoles qui devraient être émis par les organismes de standardisation.

Entre-temps, les protocoles déjà en application en Grande-Bretagne servent de base à l'étude. En particulier, les travaux initiaux ont été réalisés sur le "Network Service and Protocol over X25", protocole jouant un rôle d'interface entre les services réseau (niveaux 1-3) et le niveau transport. Les techniques de test développées utilisent comme moyen de communication entre le centre de test et le site client le "British Telecom's Packet Switched Service" (PSS) ou tout autre réseau utilisant le protocole X25 tel qu'il est défini par le CCITT.

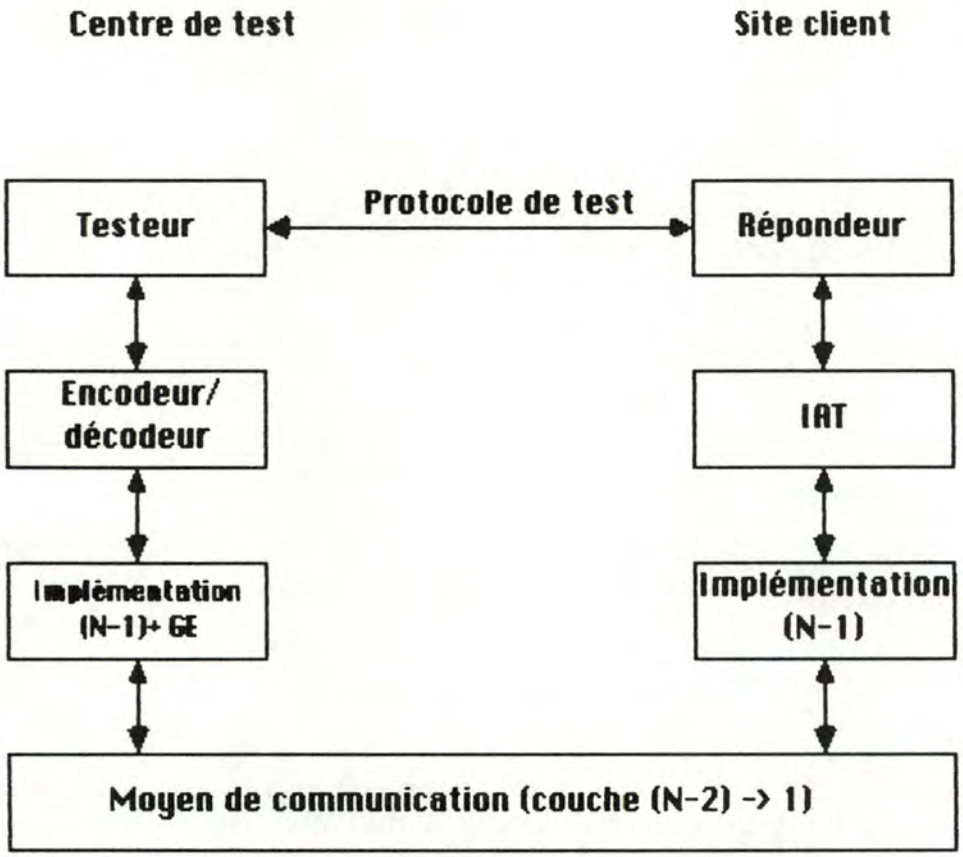
Parallèlement à ces travaux, une étroite collaboration a été établie avec le "National Computing Centre" (NCC, Manchester) pour la mise en place d'un centre de test répondant aux spécifications décrites au point 4.1.2. Ce projet, subventionné par le Ministère de l'Industrie, a abouti au début de l'année 1983 à la création d'un centre offrant des services de test de conformité qui se basent sur les techniques développées au NPL.

4.3.3.2. Graphe de l'architecture

Hypothèse

L'architecture proposée par le NPL est fondée sur le principe que l'on ne peut imposer de contraintes sur le système à tester sous peine de réduire la généralité et le domaine d'application des techniques de test. Ainsi, l'architecture qu'il propose et qui est illustrée à la figure 4.7 ne fait aucune supposition sur le système à tester (possibilités de stockage d'informations ou de datation des événements, par exemple).

Architecture



IAT : implémentation à tester
GE : générateur d'exceptions

Figure 4.7 : architecture du NPL.

4.3.3.3. Analyse des composants de l'architecture

Composants du centre de test

1. **Testeur**

Le NPL a mis au point deux types de testeur : un testeur guidé manuellement et un testeur guidé par tables d'états. Chacune de ces approches sera analysée et un exemple d'utilisation sera donné lors de l'analyse de la méthode de test (point 4.3.3.4.).

Testeur guidé manuellement

Le testeur guidé manuellement exécute des tests qui sont constitués de séquences de commandes, soit introduites par l'opérateur via son terminal soit lues dans un fichier. L'opérateur contrôle, grâce à son terminal, l'exécution des tests en envoyant des demandes de service (demande d'établissement de connexion, par exemple) et en recevant des indications de service (indication d'une demande de connexion en provenance du répondeur). De plus, il peut demander que les commandes soient non plus lues au terminal mais à partir d'un fichier dont il donne le nom. Ce fichier, similaire aux fichiers scénarios de l'approche NBS, contient outre les demandes de service à transmettre, des commandes signifiant au testeur d'attendre la survenance d'un événement donné et des commandes d'impression d'informations au terminal de l'opérateur. Deux autres types de fichier peuvent également être utilisés :

- lorsque l'on désire transférer des données, celles-ci peuvent être prélevées dans un fichier de données prévu à cet effet ;
- en vue d'analyser le résultat des tests, chaque événement et chaque action entreprise par le testeur sont mémorisés par celui-ci dans un fichier résultat.

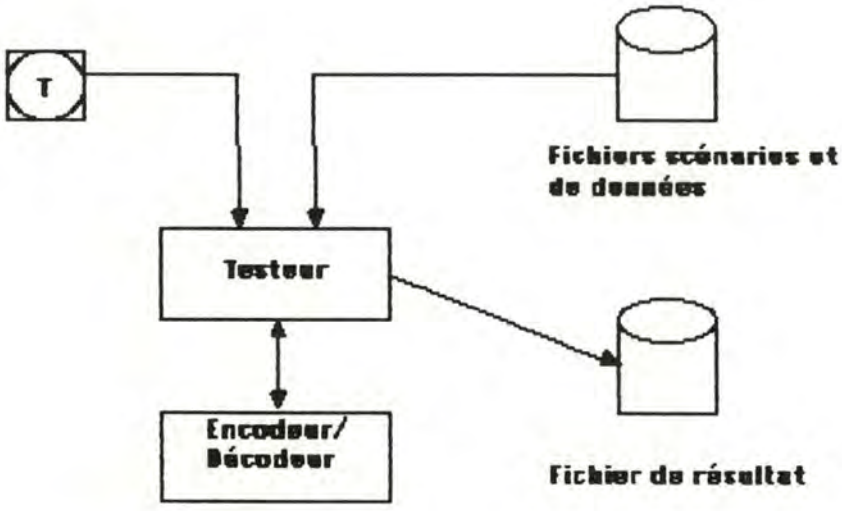


Figure 4.8 : testeur guidé manuellement.

Dans cette approche, l'ensemble des décisions concernant l'exécution du test sont prises par l'opérateur. Ainsi, à la réception de chaque primitive, il devra comparer les paramètres reçus à ceux attendus et décider de la poursuite ou de l'abandon du test en fonction de cette comparaison. Cette manière de procéder ralentit fortement la réalisation des tests .

C'est pourquoi, on réservera cette première approche à la réalisation de quelques tests initiaux généralement très courts et souvent interrompus ("debugging phase") . Une seconde approche utilisant un testeur guidé par tables d'états sera utilisée lors de la réalisation d'un ensemble de tests plus complets visant à établir la conformité de l'implémentation.

Testeur guidé par tables d'états

Lorsqu'un testeur guidé par tables d'états est utilisé, chaque test est défini par la combinaison d'une table d'états et d'une table des paramètres. La table d'états est générée automatiquement à partir de la description du test fournie par l'opérateur dans un langage prévu à cet effet **TDL** (Test Description Language , similaire au SDL décrit par le CCITT). La construction de cette table est réalisée par un processus qui peut être considéré comme un compilateur du langage TDL.

Lors de la description d'un test, l'opérateur définit un certain nombre d'états et associe à chacun de ces états un ensemble d'un ou plusieurs triplet(s) (<événement> <action(s)> <transition>). La survenance d'un événement repris dans cet ensemble provoque l'exécution des actions précisées et le passage au nouvel état mentionné dans la transition. La survenance d'un événement non repris provoque l'exécution d'actions par défaut (consistant généralement en l'abandon du test) et le passage à un état par défaut.

Les états du test ne sont pas nécessairement les mêmes que les états du protocole mais marquent les différentes étapes de ce test. Les événements sont de trois types : les messages arrivant de l'encodeur/décodeur, les commandes de l'opérateur, les événements internes tel que l'expiration d'un timer. Les actions sont de quatre types : les messages à envoyer à l'encodeur/décodeur, les messages à envoyer à l'opérateur, les actions internes tel que l'initialisation d'un timer et les impressions d'informations dans le fichier de résultats . Un exemple de table d'états est donné dans l'analyse de la méthode de test.

Le testeur est, comme le montre la figure 4.9, composé :

- d'un module traverseur permettant d'évoluer au sein de la table d'états en fonction de la survenance des événements ;
- d'un gérant d'événements ;
- d'un gérant d'actions.

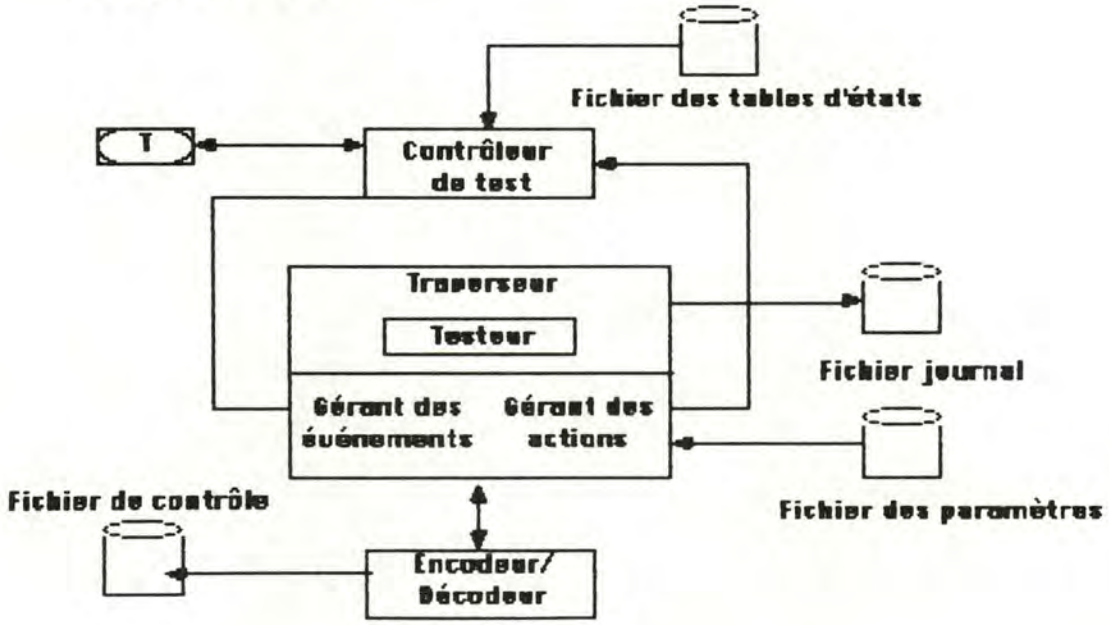


Figure 4.9 : testeur guidé par tables d'états.

Dans cette approche, le rôle de l'opérateur se limite à spécifier la table d'états et la table des paramètres qu'il désire utiliser. Une fois le test lancé, il reçoit divers messages l'informant sur l'évolution du test mais le seul contrôle qu'il puisse exercer est d'abandonner le test. L'interaction entre l'opérateur et le testeur est régie par le contrôleur de test.

Comme dans le cas du testeur guidé manuellement, chaque événement et chaque action entreprise par le testeur sont enregistrés dans un fichier de résultats. De plus, les informations traversant l'encodeur/décodeur sont également enregistrées dans un fichier (fichier de contrôle). Un programme d'analyse est disponible pour traiter ce fichier et déterminer précisement ce qui est arrivé lors de l'exécution du test.

Une amélioration peut être apportée à cette architecture en remplaçant le contrôleur de test par un coordinateur de test susceptible de réaliser une sélection dynamique des tests à exécuter. Cette sélection serait réalisée sur base des résultats des tests précédents et en prenant en compte un objectif de test (par exemple : test de conformité , mesure de performance,...). Cette amélioration est toujours à l'état de projet et constitue une perspective de recherche au NPL.

2. Encodeur/Décodeur

L'encodeur/décodeur a deux fonctions :

- générer des éléments de protocole en fonction des demandes qui lui sont fournies par son niveau utilisateur et
- transformer les éléments de protocole qu'il reçoit de l'entité paire en indications pour son utilisateur.

Par rapport à une implémentation de référence, l'encodeur/décodeur remplit ces deux fonctions en permettant en plus :

- au niveau utilisateur de demander la génération d'éléments de protocole valides mais dans un contexte invalide (par exemple, demande d'envoi de données sur une connexion non ouverte) ;
- au niveau utilisateur de demander la génération d'éléments de protocole invalides (par exemple, demande de connexion sans adresse).

De plus, il peut :

- indiquer au niveau utilisateur la réception d'éléments de protocole hors contexte ;
- indiquer au niveau utilisateur la réception d'éléments de protocole syntaxiquement invalides (alors qu'une implémentation de référence les aurait rejetés).

Grâce à ces améliorations, l'encodeur/décodeur permet de tester une implémentation aussi bien dans les cas normaux que dans les cas d'erreurs.

3. Implémentation de niveau-(N-1)

Comme pour l'encodeur/décodeur, l'implémentation du protocole-(N-1) du centre de test n'est pas une implémentation de référence étant donné qu'elle doit présenter vis-à-vis de celle-ci une amélioration nécessaire pour pouvoir tester certaines situations. Cette amélioration permet à l'encodeur/décodeur de demander la génération de cas d'exception susceptibles de se produire à ce niveau-(N-1). Ainsi, si l'encodeur/décodeur se situe au niveau 4, il pourra demander au niveau 3 de générer des erreurs tel qu'un reset ou un disconnect.

Composants du site client

1. Répondeur

Le répondeur constitue un outil simple et au comportement prévisible dont le rôle est de gérer l'interface vers le haut de l'implémentation à tester. Sa simplicité est nécessaire puisqu'il devra être inséré avec un minimum d'effort dans une grande variété de systèmes et son caractère déterministe fait que l'implémentation à tester constitue le seul composant de l'architecture dont le comportement est incertain.

De plus, le répondeur devra être suffisamment flexible pour permettre de tester un large éventail de situations.

Le répondeur est paramètré de manière à ce qu'à tout moment il ne puisse fournir qu'une seule réponse connue à un événement. Ce couplage d'une réponse à un événement (que l'on appelle le **mode**) peut être modifié dynamiquement pour une connexion donnée lorsque celle-ci a été créée. A l'initialisation, le répondeur se trouve dans un mode par défaut tel que les couples (<événement(primitive reçue)> -> <action(primitive envoyée)>) suivants sont appliqués : (connect -> accept), (accept -> disconnect),

(disconnect -> disconnect), (reset -> disconnect) (<invalid command> -> disconnect) (<autre événement : data,...> -> <aucune action>).

Le contrôle du répondeur peut être réalisé en ayant recours à un protocole non standard entre le testeur et le répondeur qui sera utilisé uniquement pour la réalisation des tests (c'est pourquoi il est parfois appelé **protocole de test**). Le comportement du répondeur est modifié en lui transmettant une commande et les paramètres qui lui sont associés au moyen des primitives de transfert de données offertes par l'IAT (DATA ou EXPEDITED DATA). La commande principale est "changer le mode" qui permet de redéfinir en une seule fois l'ensemble des couples (<événement> -> <action(s)>), c'est-à-dire le mode qui détermine le comportement du répondeur. On retrouve également dans ce protocole diverses commandes permettant de contrôler et d'analyser d'autres aspects du répondeur (commande de consultation des paramètres adresses, qualité de service,... utilisés par le répondeur) ainsi que des commandes permettant une analyse des tests (commande de consultation des différents compteurs associés à chaque événement permettant de déterminer le nombre de survenance de chacun de ces événements). L'exemple présenté dans l'analyse de la méthode de test reprend et commente certaines de ces commandes. Le protocole de test est décrit dans NPL Protocol Standards Group (1981).

Le mode par défaut du répondeur permet de réaliser les tests de base visant à déterminer si l'implémentation à tester répond correctement aux demandes d'établissement de connexion et gère correctement la réception des données. Une fois ces deux possibilités vérifiées, on peut utiliser les services offerts par l'implémentation à tester pour l'acheminement des commandes formant le protocole de test (et notamment pour effectuer un changement de mode ce qui n'était pas le cas avant ces tests puisque les commandes traversaient un composant au comportement indéterminé).

2. L'implémentation à tester (IAT)

L' IAT est une implémentation d'une ou plusieurs couche(s) adjacente(s) que l'on désire tester. Elle est choisie de manière à pouvoir être testée indépendamment des implémentations de protocoles supérieurs c'est-à-dire telle qu'elle ait une interface vers le haut accessible. Si une telle interface n'est pas disponible, l'implémentation est dite "intestable" dans l'environnement décrit.

3. Implémentation de niveau (N-1)

L'implémentation-(N-1) du site client ne présente aucune caractéristique notable bien qu'elle doit évidemment avoir été testée au préalable tout comme l'ensemble des couches de niveau inférieur à l'IAT. On peut donc considérer qu'il s'agit d'une implémentation de référence.

4.3.3.4. Méthode de réalisation des tests

La méthode de test sera décrite en deux parties correspondant aux deux alternatives en ce qui concerne le testeur (testeur guidé manuellement et testeur guidé par tables d'états).

Exemple 1 de déroulement de test : testeur guidé manuellement

Dans cette approche, une séquence de commandes est fournie au testeur qui agit en fonction du type de chacune de ces commandes. Ces séquences sont soit introduites directement au terminal soit prélévées dans un fichier. C'est cette seconde approche qui est la plus souvent utilisée (et qui sera illustrée ici) car elle évite à l'opérateur de devoir mémoriser toute une syntaxe tout en étant de loin plus rapide.

L'opérateur commence par sélectionner un des scénarios qu'il désire exécuter. Le tableau ci-dessous montre un exemple de tel scénario. On y retrouve quatre types de commandes qui devront être séquentiellement traitées par le testeur :

- commandes d'impression au terminal ou dans le fichier résultat. Elles permettent d'une part de suivre l'évolution d'un test et d'autre part d'en faire une analyse.

Ontest, Oftest : impriment le message donné en paramètre au terminal et dans le fichier résultat

Flash : imprime le message donné en paramètre au terminal.

-commandes d'émission de primitives à transmettre à l'encodeur/décodeur et qui auront la syntaxe suivante <primitive à émettre> <paramètres>

-commandes d'attente de primitives (**Hold** <nombre d'événements> <événement(s)>) qui suspend l'analyse du scénario jusqu'à la survenance des événements donnés en paramètre.

-commandes d'enchaînement des scénarios.

File : provoque l'exécution de la séquence de commandes du fichier donné en paramètre;

Return : provoque le retour vers l'entité appelante. Situé dans un fichier appelé par la commande file, "return" provoque le retour au scénario appelant au point suivant immédiatement la commande d'appel. Situé dans un scénario "principal", elle permet de rendre le contrôle à l'opérateur.

Les commandes venant d'être énumérées ne constituent pas une liste complète des possibilités offertes mais permettent à la fois, d'évaluer l'étendue des possibilités du langage et de comprendre l'exemple donné ci-dessous.

```
ONTEST "** Example **"
FILE "Test.open"
FILE "Test.chmode"
      .    .    .
      .    .    .
OFTEST "** Example ends correctly **"
```

Fichier scénario principal

```
Test.open
    FLASH "Normal connect";
    CONNECT 0,  1234, 5678;
    FLASH "Wait for accept";
    HOLD 1, "accept";
    RETURN;
```

```
Test.chmode
    FLASH "Change of mode";
    FLASH "Actual value :";
    FLASH "5,1,6,1,1,1,1,1,6,
    6,1,9,6,6,33,400,1,33,90,
    1,48,57,1,3";
    EXPEDITED "<command>";
    RETURN;
```

Fichiers scénarios

Cet exemple est en fait incomplet puisqu'il consiste uniquement à ouvrir une connexion et à fixer le comportement du répondeur.

L'établissement de la connexion est réalisé en ayant recours au fichier test.open dans lequel on retrouve des commandes d'impression, une commande d'envoi de la primitive CONNECT et une commande d'attente de la réponse ACCEPT. Une fois la connexion établie on communique au répondeur quel doit être son nouveau comportement. Ceci est réalisé dans le fichier test.chmode par l'envoi de données express contenant la commande "changer le mode" et ses paramètres (venant d'être affichés à l'écran). La syntaxe utilisée est relativement complexe mais reste néanmoins sémantiquement équivalente à la suivante :

<type de la commande> <événement i> <action i>

Le test réalisé concerne donc la possibilité d'ouvrir une connexion et d'utiliser le service de transfert de données express.

Exemple 2 de déroulement de test : testeur guidé par tables d'états

Contrairement à la première approche, lorsqu'un testeur guidé par tables d'états est utilisé, l'opérateur ne doit pas fournir une séquence de commandes mais une table d'états et une table des paramètres. La commande de base pour lancer l'exécution d'un test sera donc RUN <fich.table> <fich.param>. Le tableau ci-dessous donne un exemple de table d'états représentant un test. Celle-ci possède la même structure qu'un programme à savoir une partie déclaration et une partie texte. La partie déclaration (ADDR PARAMETERS <paramètres>) énonce les différents paramètres qui doivent être puisés dans la table des paramètres donnée.

```
TEST example;
 ADDR PARAMETERS called address, calling address, recall address;
 INITIAL ACTION connect request called address, calling address;
END

STATE wait-for-accept;
 EVENT accept ra,qsa,eta;
 ACTION compare-parms-accept ra, recall address, qsa, "",eta,
                             "test responder version 1";
 NEXT accept-parms-ok;
END
```

```
STATE accept-parms-ok;
  EVENT normalreturn;
  ACTION <command to change the mode>;
  NEXT check-mode-correctly-set;
  EVENT abnormalreturn;
  ACTION stop test "accept parameters faulty";
  NEXT idle;
END
```

Exemple de table d'états

Le texte comporte une première action à exécuter (INITIAL ACTION) qui dans notre exemple consiste en l'envoi d'une demande d'établissement de connexion. Par défaut, une fois cette commande réalisée, on passe dans le premier état explicitement défini : wait-for-accept. Pour cet état comme pour tout autre, un ensemble d'un ou plusieurs triplet(s) (<événement> <action> <transition>) est défini et constitue la description de l'automate. Dans le premier état, seul un événement (accept ra qsa eta) se voit associer une action explicite (compare . . .). Cette action est dite "interne" au testeur et consiste à comparer les paramètres reçus aux paramètres attendus. Celle-ci produira un événement interne comme résultat qui sera soit "normalreturn" en cas d'équivalence des paramètres soit "abnormalreturn" en cas de différence entre les paramètres. Un grand nombre de telles actions internes sont définies permettant ainsi la réalisation de nombreux contrôles.

Le langage utilisé TDL permet aussi l'utilisation de variables locales qui peuvent être utilisées comme compteurs d'actions ou d'événements. La gestion des timers est également possible grâce aux commandes "set timer <nom> <durée>" (positionnement d'un timer lors d'une action), "stop timer <nom>" (arrêt d'un timer lors d'une action) et "timer <nom>" (expiration d'un timer constituant un événement).

4.3.4. L'architecture de l'Agence De l'Informatique (ADI)

4.3.4.1. Introduction

Le projet pilote français RHIN mis sur pied par l'Agence De l'Informatique (ADI) au début de l'année 1981 s'est fixé comme objectif général d'accélérer l'utilisation effective des standards de télécommunication en France. Pour ce faire, les recherches réalisées ont pour objectif de définir et d'expérimenter un environnement technique nécessaire à l'élaboration (édition), l'évaluation (validation formelle), la qualification (test de conformité), la promotion et la diffusion de protocoles normalisés.

Dans le cadre des tests de conformité, le projet a abouti à la réalisation de deux outils disponibles tous deux depuis fin 1982 :

- *GENEPI* : un générateur d'éléments de protocole ;
- *CERBERE*: un outil d'espionnage du service réseau.

Une architecture globale *STQ* (Système de test et de Qualification) a également été définie mais par manque d'informations, seuls ses principes seront énoncés. La section qui suit décrit successivement les outils Cerbère, Genepi et l'architecture STQ s'écartant ainsi quelque peu de la structure d'analyse utilisée jusqu'à présent.

4.3.4.2. Architecture et analyse des composants

La vérification de conformité à des protocoles de haut niveau est un processus complexe, cette complexité étant dûe d'une part à l'aspect "réparti" de ce processus, et d'autre part aux conditions spécifiques de son environnement (à savoir ne pas intervenir dans le système à tester).

C'est la raison pour laquelle plusieurs méthodes sont développées en parallèle dans le cadre du projet RHIN.

CERBERE

Cerbère est un outil d'espionnage du service réseau et de simulation de défaut de qualité de service de celui-ci. Intercalé entre le réseau et le système à tester, il se comporte vis-à-vis du réseau comme un équipement terminal de traitement de données (ETTD) et vis-à-vis du système à tester en tant qu'équipement de terminaison de circuit de données (ETCD), comme le montre la figure 4.10.

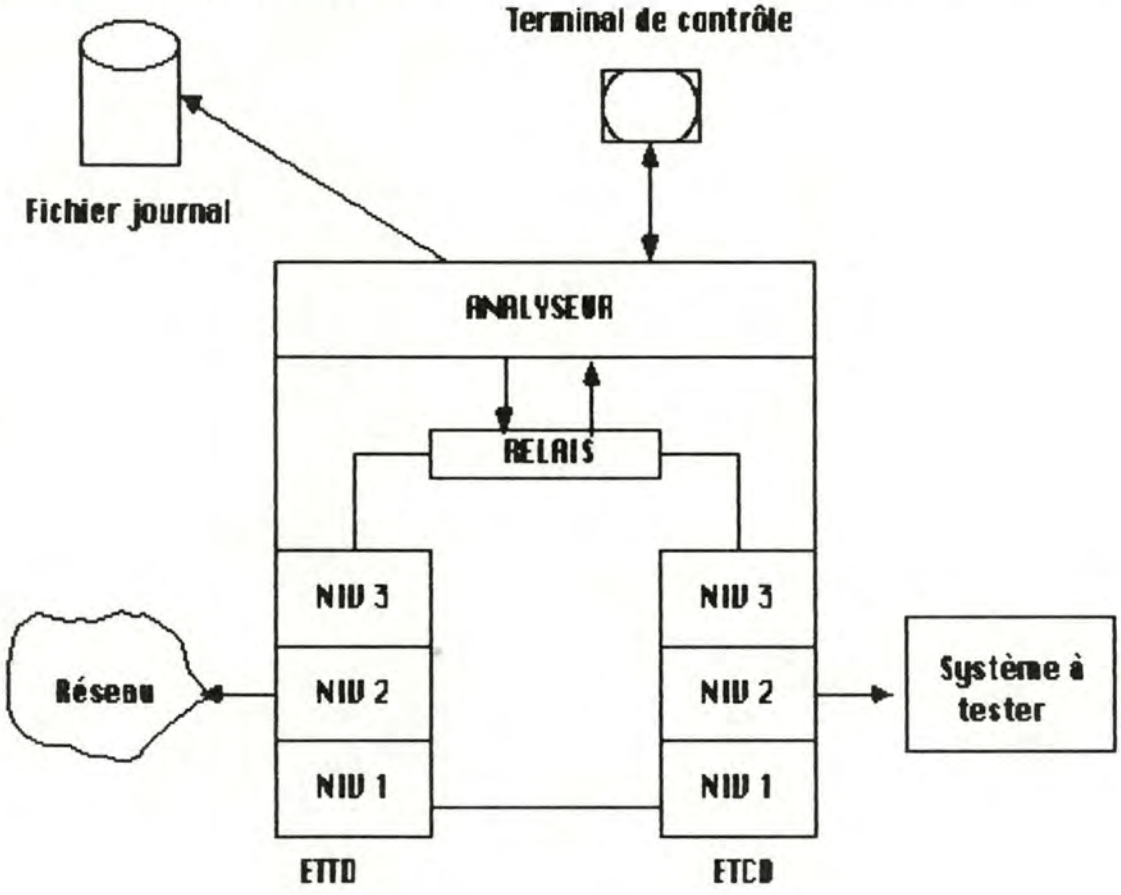


Figure 4.10 : structure interne de l'outil Cerbère.

Cerbère a deux fonctions principales, à savoir la gestion du protocole X25 et l'analyse des paquets de données et leur manipulation.

Gestion du protocole X25 : relais

Cette partie du software comporte un ensemble de fonctions pour assurer la réception et l'envoi de paquets selon les spécifications X25. On y retrouve deux implémentations de X25, l'une fonctionnant en tant qu'ETCD et l'autre en tant qu'ETTD. Cette partie appelée **relais** échange des paquets avec une seconde partie l'**analyseur** via un ensemble de primitives qui correspondent à la définition du service de la couche réseau (mêmes primitives que dans la relation niveau 4/ niveau 3).

Analyse et manipulation des paquets : analyseur

En ce qui concerne les fonctions d'analyse et de manipulation, il appartient à l'opérateur de définir un ensemble de routines réalisant de telles fonctions. Ainsi, il associera à chaque élément de protocole pouvant être reçu une routine d'analyse et de manipulation rédigée dans un langage de haut niveau (PASCAL).

Lorsqu'un élément de protocole est reçu et transmis à l'analyseur, les opérations suivantes sont réalisées :

- décodage de l'élément de protocole et détermination de son type ;
- exécution de la routine d'analyse et de manipulation associée ;
- mise à jour éventuelle de variables internes ;
- stockage et/ou impression éventuelle d'informations.

En vue de faciliter la tâche de l'opérateur dans la réalisation des routines d'analyse, un ensemble de fonctions de base lui sont offertes. Ces fonctions, qui peuvent être utilisées dans les routines d'analyse rédigées par l'opérateur, sont les suivantes :

- fonctions d'affichage permettant la définition de formats d'écran et l'impression de données formatées ;
- fonctions de stockage permettant d'enregistrer des données et informations de contrôle dans des fichiers de l'opérateur ;
- fonctions mathématiques de calculs statistiques tel que des calculs de moyenne, variance et écart-type ;
- fonctions d'étude du réseau permettant une étude statistique de l'utilisation du réseau (nombre de demandes de connexion, taille moyenne des éléments de protocole véhiculés, . . .) ;
- fonctions de simulation de défaut de qualité du service réseau :

Sur demande de l'opérateur ou lors de la détection d'un événement particulier par l'analyseur, Cerbère pourra générer des erreurs dans le service réseau telles que la perte, l'altération, la duplication de données, des réinitialisations, des libérations, . . . ;

- fonctions de datation permettant d'introduire une chronologie dans la survenance des événements.

Mode de fonctionnement

Cerbère permet de gérer 16 connexions au maximum. Chacune de ces connexions peut se trouver dans un des trois états suivants :

- état transparent : tous les paquets traversent l'outil sans qu'il y ait analyse ou manipulation ; logiquement, on peut considérer dans ce cas que Cerbère n'est pas utilisé.

- état parallèle : tous les paquets traversant Cerbère sont analysés et transmis simultanément vers l'autre coté. Les données transmises ne peuvent être manipulées (altérées). Le trafic entre le centre de test et le système à tester est par conséquent indépendant de l'analyse. Cet état est utilisé pour des analyses de longue durée telles que des mesures statistiques ou de performance.

- état série : tous les paquets traversant Cerbère peuvent être analysés et manipulés. Les paquets sont transmis uniquement lorsque leur analyse et/ou leur manipulation est terminée. Si les fonctions réalisées ralentissent le flux des données, aucune donnée ne sera pour autant perdue étant donné que Cerbère utilise les principes du contrôle de flux X25/3 (mécanisme de fenêtre et des accusés de réception). Cet état est utilisé pour les tests de conformité et permet, en particulier, d'introduire des erreurs dans le service réseau.

Ainsi, Cerbère intervient comme outil complémentaire de test au sein d'une architecture. Il peut ou non être inséré dans celle-ci selon les tests que l'on désire mener. Notons également que cet outil est indépendant du centre de test puisqu' aucun mécanisme de transfert d'informations entre ce centre et Cerbère n'est prévu. En fait, Cerbère est utilisé comme outil autonome. De plus, étant indépendant des protocoles à tester, il peut être utilisé quel que soit le protocole à tester. Ainsi, sur base des fonctions élémentaires, l'opérateur pourra bâtir des programmes décodant les éléments de protocoles de niveau supérieur (transport, session, . . .) afin de les analyser, les manipuler, les afficher, ...

Genepi

Genepi est un générateur d'éléments de protocole c'est-à-dire un outil susceptible de construire des éléments de protocole-(N) et d'utiliser le service-(N-1) pour échanger ces éléments avec l'IAT. De plus, Genepi est indépendant du protocole à tester. Il est conçu pour manipuler des objets qui correspondent aux concepts de base d'un protocole : élément de protocole-(N), accès au service service-(N-1), mise en correspondance des éléments de protocole-(N) dans les demandes à envoyer au service-(N-1). Genepi ne connaît pas les formats des éléments de protocole-(N) mais est capable de les apprendre.

Genepi est basé sur le principe qu'un protocole peut être "éclaté" en trois parties :

- le codage et le décodage des éléments de protocole ainsi que leur passage par le service-(N-1) ;
- la structure de contrôle du protocole (qui définit la manière dont sont gérés les passages d'un état à un autre) ;
- la manipulation de variables internes.

Codage - Décodage

Pour chaque élément de protocole, on définira trois types de représentation :

- la représentation externe correspondant aux formats tels qu'ils sont décrits dans la norme ;
- la représentation logique définissant une vue logique de l'opérateur sur ces éléments de protocole (PDU). Celle-ci reprendra pour chaque PDU son nom, le nom de ses paramètres et leur valeur ;
- la représentation interne non connue de l'opérateur et uniquement manipulée au sein du générateur.

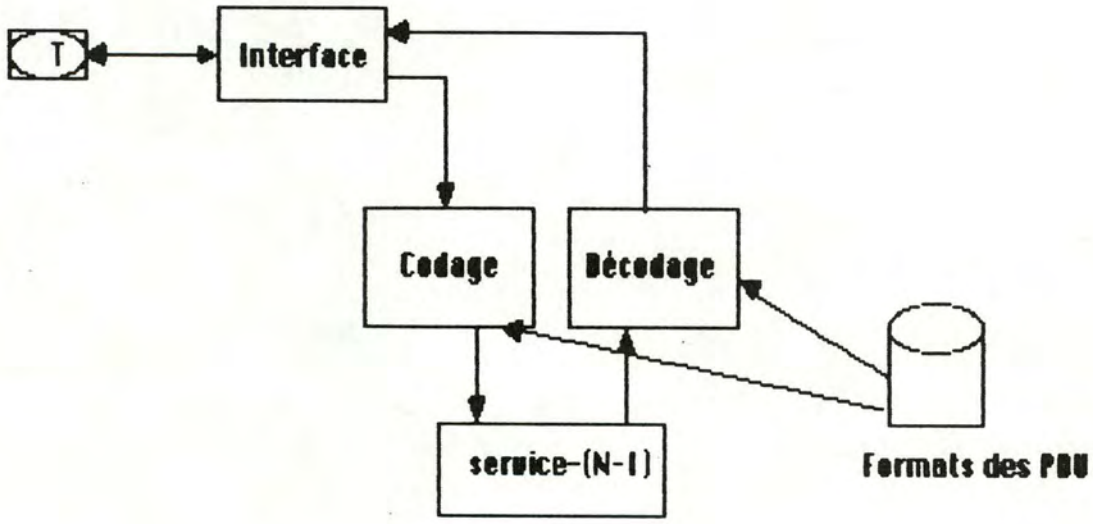


Figure 4.11(a) : structure interne de l'outil Genepi.

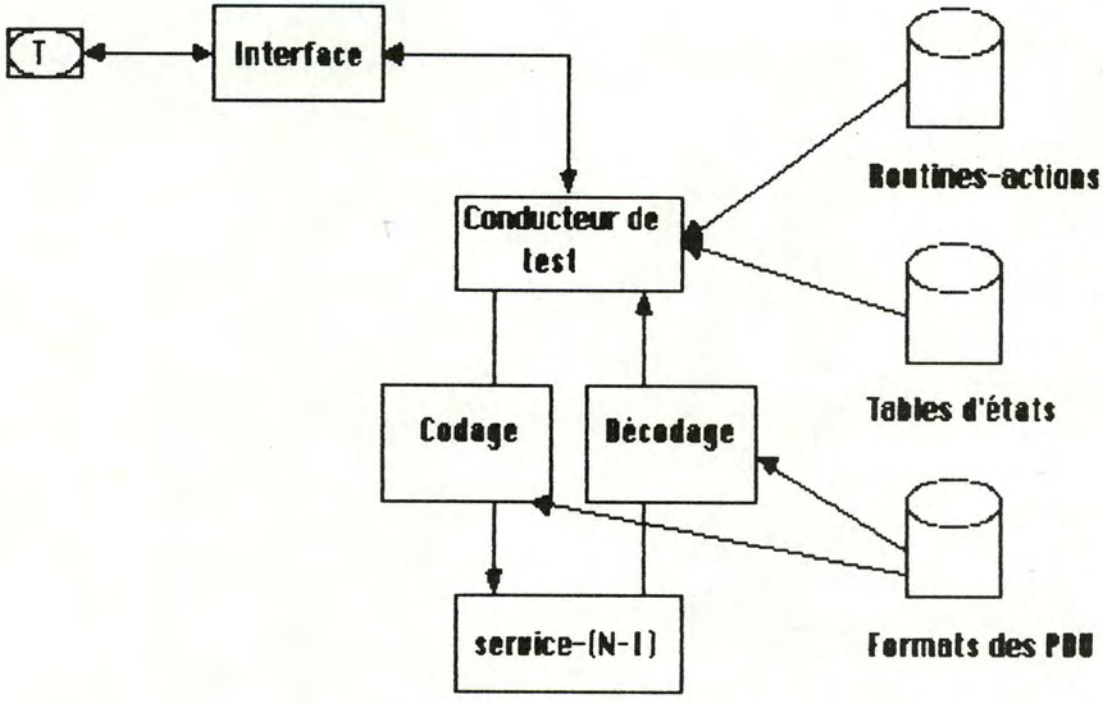


Figure 4.11(b) : structure interne de l'outil Genepi/a.

Lorsque l'opérateur désirera tester un protocole, il définira :

- les formats des éléments de protocole (représentation externe) ;
- la représentation logique ;
- les correspondances existant entre ces deux représentations.

Sur cette base, deux routines internes de codage et décodage seront générées automatiquement.

Structure de contrôle du protocole

Dans une première approche (**Genepi**), tout contrôle est laissé à l'opérateur qui sur base d'un événement (apparition à l'écran de contrôle d'un élément de protocole reçu) détermine quelle action il désire réaliser pour poursuivre le test. Cette approche interactive est très lourde et entraîne de fortes pertes de temps. Elle est illustrée à la figure 4.11(a).

Une seconde approche (**Genepi/A** - figure 4.11(b)) consiste en l'adjonction d'un composant - le conducteur de test - susceptible de guider le test sur base d'une table d'états (automate). Pour ce faire, l'opérateur doit définir la liste et le format des différents événements pouvant survenir ainsi que les transitions inter-états. Les événements possibles sont l'arrivée d'un élément de protocole, l'indication d'un service-(N-1) ou des événements internes tels que l'expiration d'un timer ou une commande de l'opérateur. L'opérateur mentionnera également pour tout événement et pour tout état le nom d'une procédure à exécuter (appelées routines-actions).

Ces routines sont constituées d'un ensemble de commandes permettant de construire des éléments de protocole, d'accéder au service-(N-1), de manipuler des variables locales, de tester les paramètres des événements ou de fixer l'état du protocole.

Manipulation de variables internes

Un ensemble de commandes pouvant être utilisées par l'opérateur lors de la rédaction des routines associées à chaque événement sont offertes par Genepi. Ces commandes permettent :

- d'initialiser des champs d'éléments de protocole avec des valeurs de variables ;
- de réaliser différentes opérations sur les variables ;
- d'affecter une valeur à une variable en utilisant soit des constantes soit des champs d'élément de protocole reçu.

Mode d'utilisation

Comme le montre la figure 4.12, Genepi envoit les éléments de protocole qu'il génère à l'implémentation à tester via un moyen de communication (réseau). Celle-ci réagit en conséquence et fournit certaines indications à son niveau utilisateur : **le répondeur passif**. Ce répondeur est un composant au comportement connu et non ambigu. Quant il reçoit un indication de service de l'IAT, il y répond en émettant une demande de service vers la couche inférieure. Cette demande est transcrite en élément de protocole et est transmise vers le centre de test (via le réseau) où l'on peut observer le comportement de l'IAT. La technique employée est donc classique pour une architecture décentralisée.

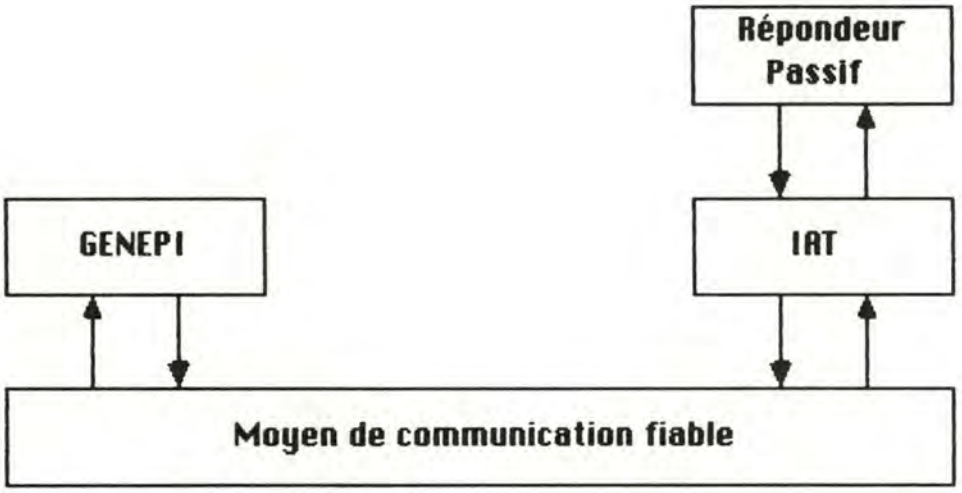


Figure 4.12 : utilisation de Genepi.

Lorqu'il utilise Genepi, l'opérateur définit dans une première phase les formats des éléments de protocole (deux représentations : externe et logique). Il introduit ensuite diverses commandes et observe leur résultat dans une séquence telle que la suivante :

- introduction d'une commande d'accès au service-(N-1) (demande d'ouverture d'une connexion-(N-1)) ;

- analyse de la primitive de service-(N-1) reçue (connexion-(N-1) acceptée et ouverte) ;
- introduction d'une commande permettant de composer des PDU (construction d'un élément de protocole-(N) connect request).

L'opérateur utilise donc Genepi comme un outil d'aide :

- à la construction d'éléments de protocole ;
- au décodage des éléments de protocole reçus ;
- à l'accès au service-(N-1) ;
- à la manipulation et la mémorisation de variables internes.

SYSTEME DE TEST ET DE QUALIFICATION (STQ)

La figure 4.13 représente l'architecture STQ. Par manque d'informations, nous ne détaillerons pas celle-ci. Notons cependant qu'elle devrait être basée sur les principes suivants :

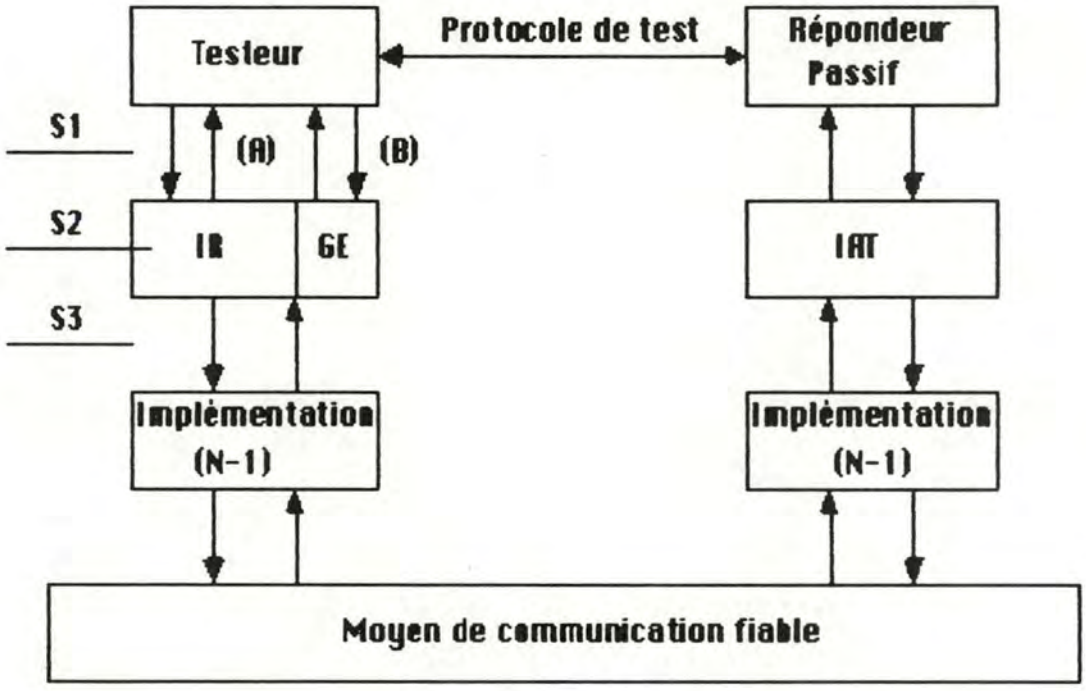
- utilisation d'un protocole de test semblable à celui utilisé dans l'approche NPL pour fixer le comportement du répondeur passif ;

- l'implémentation équivalente à l'IAT serait composée :

  * d'une interface de service standard (A) afin de permettre à la couche (N+1) (testeur) d'accéder au service ;
  * d'une interface additionnelle (B) afin d'accéder aux fonctions de trace et de génération d'erreurs ;
  * des traces situées à trois niveaux permettant d'étudier les événements aux interfaces N+1/N (S1) et N/N-1 (S2) ainsi que les valeurs de variables d'état du protocole (S3);

- utilisation d'une connexion de service pour le transfert de commandes vers le répondeur passif (comme c'est le cas pour l'architecture Bull à la différence près que dans STQ, les deux connexions passent par l'IAT) ;

- adjonction à l'implémentation de référence d'un générateur d'erreurs capable de générer des erreurs de protocole, d'activer les traces, de mettre en jeu toutes les valeurs possibles des paramètres d'éléments de protocole.



IAT : implémentation à tester
GE : générateur d'exception
S1, S2, S3 : sondes
IR : implémentation de référence

Figure 4.13 : proposition d'architecture de l'ADI.

Le système STQ est un système complet de test. Utilisé en conjonction avec Cerbère, il permet de tester tous les états possibles et paramètres des protocoles, Cerbère permettant de simuler les stimuli fournis par la couche réseau. Ce système est dépendant des protocoles par opposition à Genepi qui est indépendant du protocole.

4.3.5. L'architecture de Bull

4.3.5.1. Introduction

Le groupe BULL a conçu un système décentralisé pour tester les implémentations de protocoles ISO/OSI. Ce système, appelé **STP** (System for Testing Protocol) a été élaboré dans le cadre d'un accord conclu entre douze constructeurs européens : ICL, GEC, Plessey, Siemens, Nixdorf, AEG, Olivetti, STET, Philips, Thomson, CGE et Bull. Cet accord établi en mars 1984 vise le développement et la mise au point d'implémentations de protocoles correspondant au modèle ISO/OSI. Ces protocoles devront permettre aux utilisateurs des machines de ces différents constructeurs de communiquer entre eux.

4.3.5.2. Graphe de l'architecture

Hypothèses

- Le modèle OSI est généralement considéré comme une pile d'implémentations de protocole où l'on ne peut accèder qu'à l'implémentation supérieure. Cette situation empêche l'utilisateur d'avoir accès à différents niveaux de communication (session, transport, ...). En ce qui concerne les tests, les informations échangées entre le testeur et le répondeur passent nécessairement par l'implémentation à tester qui, par définition, est un composant à comportement indéterminé. Vu la difficulté d'acheminer les instructions au répondeur, cette situation diminue le contrôle de l'exécution des tests.

Afin de remédier à cet inconvénient, Bull structure les implémentations des différentes couches de protocoles du modèle OSI suivant une disposition que l'on pourrait qualifier de "disposition en escalier" et qui est représentée à la figure 4.14 . La caractéristique principale de cette disposition est qu'un utilisateur peut avoir accès à n'importe quel niveau de communication.

Dans le but de rendre uniforme l'accès à ces différents niveaux, une interface généralisée est construite.

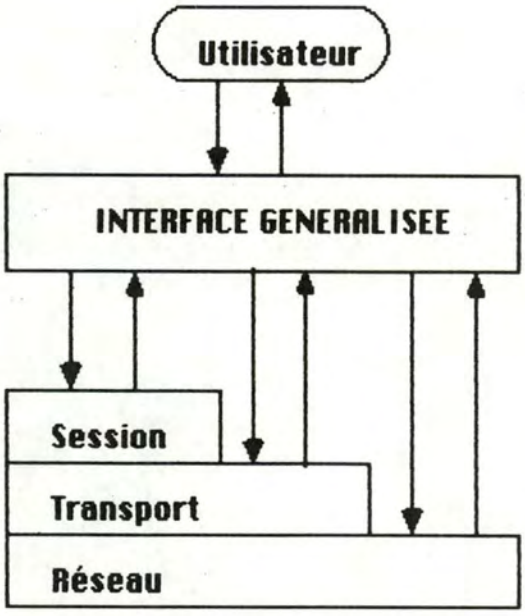
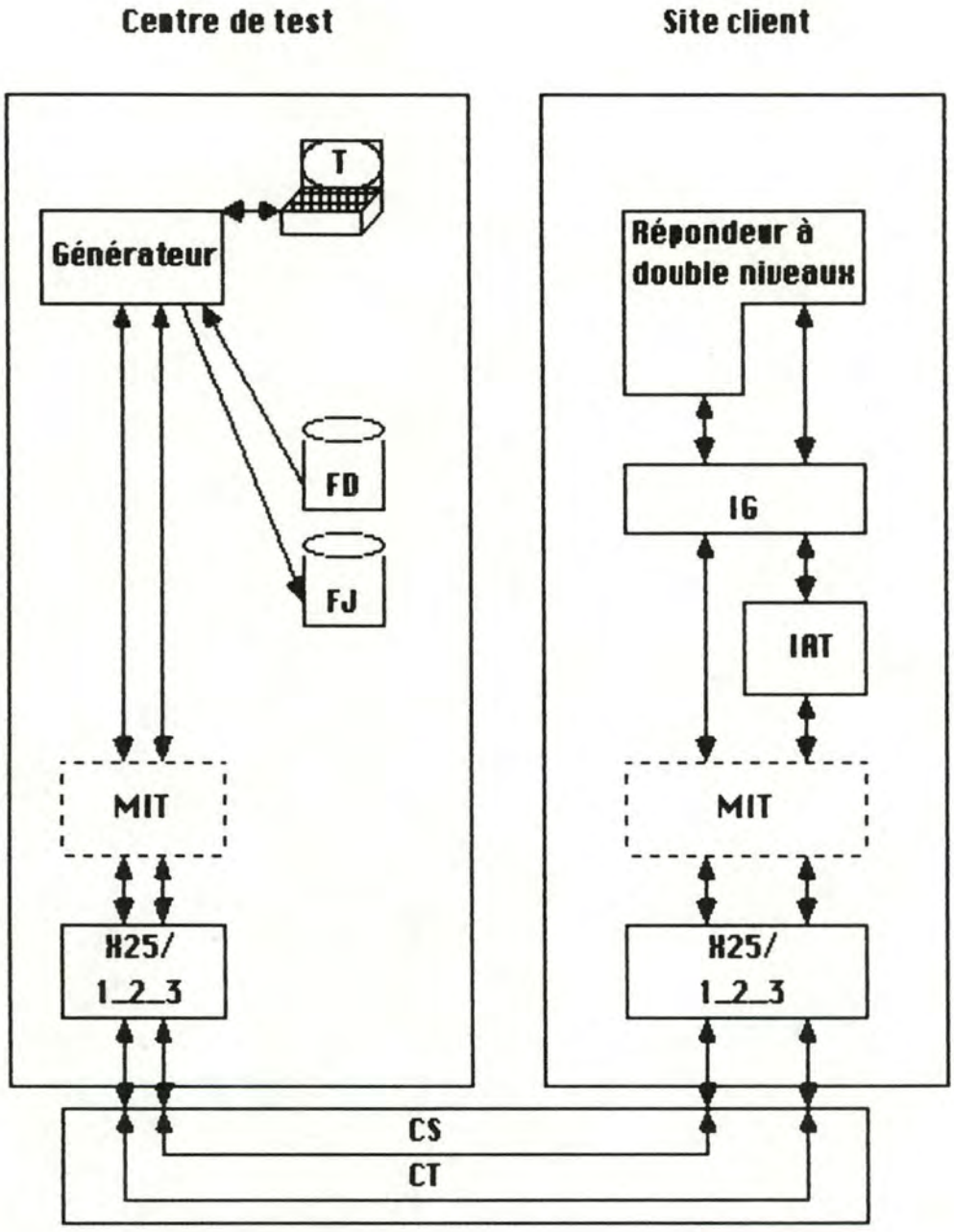


Figure 4.14 : implémentation en escalier.

Architecture

La figure 4.15 représente l'architecture du "System for Testing Protocol de Bull" de Bull.



FD : fichier des données
FJ : fichier journal
MIT : module des implémentations testées

IG : interface généralisée
IAT : implémentation à tester
CS : connexion de service
CT : connexion de test

Figure 4.15 : architecture de Bull.

4.3.5.3. Analyse des composants de l'architecture

Composants du centre de test

Les composants du centre de test doivent permettre d'échanger des éléments de protocole-(N) avec l'implémentation à tester de niveau-(N), via un service-(N-1). Afin de construire les éléments de protocole-(N) à envoyer à l'implémentation à tester-(N), différentes techniques peuvent être utilisées dans le centre de test, à savoir celle de l'implémentation de référence, celle de l'encodeur/décodeur et celle de la substitution. La première méthode a été étudiée lors de l'analyse de l'architecture proposée par le NBS (voir section 4.3.2). La seconde a été vue lors de l'analyse de la proposition du NPL (voir section 4.3.3).

Quant à la technique de substitution, elle est basée sur la construction directe des éléments de protocole par un générateur d'éléments de protocole, en lieu et en place de l'implémentation de référence. C'est la technique adoptée par Bull.

1. **Générateur**

Sur base d'un scénario, le générateur construit les éléments de protocole destinés à l'implémentation à tester, ainsi que les primitives de service devant être communiquées par le répondeur du site client à l'interface supérieure de l'implémentation à tester. Une fois les éléments de protocole et les primitives construits, ils sont transmis à l'implémentation à tester par les services-(N-1) sous-jacents, et respectivement à travers les connexions de service et de test établies.

Les éléments de protocole-(N) en provenance de l'implémentation à tester sont reçus grâce à ces services sur la connexion de test, tandis que les indications de service transmises par l'IAT au répondeur sont acheminées sur la connexion de service.

Les principales fonctions assurées par le générateur de Bull sont :

- l'établissement de deux connexions-(N-1) : la connexion de service (CS) et la connexion de test (CT) ;

- la génération d'éléments de protocole-(N) et l'envoi de ceux-ci à l'implémentation à tester, sur la connexion de test ;

- la génération des primitives de service-(N) à transmettre à l'interface supérieure de l'implémentation à tester, et l'envoi de celles-ci sur la connexion de service ;

- le décodage des éléments de protocole-(N) générés par l'implémentation à tester et reçus sous forme d'indications de service-(N-1) sur la connexion à tester ;

- le décodage des indications de service-(N) générées par l'implémentation à tester sur son interface supérieure. Ces indications de service sont transmises au centre de test, par le répondeur du site client, via la connexion de service ;

- la manipulation des timers ;

- la gestion des fichiers.

2. Module des implémentations testées (MIT)

Lorsque l'implémentation à tester est celle d'une couche supérieure à la couche transport (session (5), présentation (6), application (7)), ce module contient les implémentations vérifiées et correctes des couches inférieures à celle testée et non comprises dans le module X25 (voir ci-dessous). Par exemple, si l'implémentation à tester est celle d'un protocole de session, le MIT contient l'implémentation vérifiée de la couche transport. Lorqu'elles sont nécessaires, ces implémentations sont utilisées par le générateur.

3. X25/1_2_3

Ce module contient les implémentations vérifiées et correctes des couches 1, 2, 3 telles qu'elles sont spécifiées dans l'avis X25 du CCITT et permet au site client d'entrer en communication avec le noeud d'entrée du réseau selon l'interface standard X25. Ce réseau permettra la transmission de données entre le répondeur et le centre de test par l'intermédiaire des connexions de service et des connexions de test.

Composants du site client

1. Répondeur

Suivant le modèle des implémentations en escalier, un répondeur, appelé **"répondeur à double niveaux" (RDN)** est développé sur le site client. Ce répondeur constitue un composant logiciel ayant accès à deux niveaux de communication : le niveau-(N) de l'implémentation à tester et un niveau inférieur déjà testé (par exemple (N-1)). La caractéristique principale d'une telle configuration est d'être capable d'établir des échanges fiables entre le système de test et le répondeur grâce à une connexion ne passant pas par l'implémentation à tester, qui possède un comportement imprévisible. Cette connexion est appelée connexion de service (CS) et se différencie de la connexion de test (CT) exploitée par l'implémentation à tester. Le rôle du répondeur est de gérer ces deux connexions en :

- considérant les données reçues sur la connexion de service comme des demandes de service à transmettre vers l'implémentation à tester-(N);

- transmettant les primitives de service de type indication reçues de l'implémentation à tester au centre de test, au moyen de la connexion de service, en utilisant le service-(N-1) rattaché à celle-ci.

Lorsque l'on entame un test, l'implémentation à tester peut être atteinte par l'intermédiaire de son interface vers le bas et une connexion de test ou via son interface vers le haut et une connexion de service, préalablement établie entre le système de test et le répondeur.

Pour mettre en oeuvre un tel répondeur, la connaissance de l'interface-(N) de l'implémentation à tester-(N) et de l'interface-(N-1) de l'implémentation déjà testée sont requises. Ainsi, si l'implémentation à tester représente un protocole de transport (N=4), le répondeur gèrera l'interface supérieure de cette implémentation et l'interface supérieure d'une implémentation déjà testée de la couche réseau (N-1=3).

2. Interface généralisée (IG)

L'IG est une interface entre le répondeur à double niveaux et les implémentations inférieures. Elle permet au répondeur une utilisation plus aisée des implémentations de niveau inférieur en lui évitant de connaître la façon dont les primitives de service doivent leur être communiquées (soit par file d'attente, soit par pipe/UNIX, soit par fichier intermédiaire).

3. Implémentation à tester (IAT)

L'IAT est l'implémentation d'un protocole défini dans le cadre du modèle OSI et pour laquelle on désire établir la conformité par rapport aux spécifications de ce même protocole.

4. Module des implémentations testées (MIT)

Ce module remplit la même fonction que son homologue du centre de test, mis à part le fait que lorsque ces implémentations sont utiles, elles sont sollicitées par l'implémentation à tester ou le répondeur et non plus par le générateur.

5. X25/1_2_3

Comme il l'a été expliqué ci-dessus, ce module permet au site client d'entrer en communication avec le noeud d'entrée du réseau auquel il est rattaché.

Fichiers utilisés

Les fichiers tenus à jour par le générateur sont :

- le **fichier journal (FJ)** contenant (1) les éléments de protocoles générés et reçus par le générateur et (2) les primitives de service générées pour l'interface supérieure de l'IAT et les indications de service apparues sur cette interface ;

- le **fichier des données (FD)** contenant les données à introduire dans les éléments de protocole et dans les demandes de service à transmettre à l'IAT.

Caractéristiques de la technique STP (System for Testing Protocol)

1. Pas de protocole de test

Dans cette technique, le répondeur à double niveau n'a pas d'intelligence en ce sens qu'il n'effectue aucun traitement sur les données qu'il reçoit de l'implémentation à tester ou de la connexion de service. De même, il ne fait aucune manipulation de l'interface de l'implémentation à tester sans en avoir eu l' "autorisation" de la part du centre de test. Lorsqu'il reçoit une indication de service sur l'interface-(N) de l'implémentation à tester, il l'envoie immédiatement au système de test en utilisant la connexion de service. Lorsqu'il repère sur la connexion de service, une indication de service-(N-1) contenant une demande de service-(N), il la délivre immédiatement à l'implémentation à tester. Ainsi, le système de test contrôle le répondeur via la connexion de service et sans protocole de test spécifique.

2. possibilité de tests bi-directionnels

La technique utilisée permet d'évaluer le comportement de l'implémentation à tester pour les deux sens possibles d'initialisation de la connexion entre le centre de test et le site client. Ainsi, dans le cas où la connexion est demandée par le centre de test, la technique permet de vérifier la réaction de l'implémentation en réponse aux indications de service-(N-1), reçues sur l'interface inférieure, via la connexion de test. Inversément, dans le cas où la connexion est initiée par le site client, elle permet de vérifier le comportement de l'implémentation suite aux demandes de service-(N) reçues sur l'interface supérieure. Ces demandes de service sont communiquées au site client via la connexion de service.

3. Pas de support magnétique requis sur le site client

L'activité de test, dans le site client, est centrée seulement sur le comportement de l'implémentation à tester et il n'est pas nécessaire de garder une trace des échanges entre l'implémentation et le répondeur.

En effet, le contenu de ces échanges peut être totalement enregistré dans le centre de test.

4. Possibilité de contrôler la réaction de l'implémentation aux erreurs

Ces erreurs sont de deux types :

- les erreurs de protocole (erreurs dans les paramètres de l'élément de protocole, éléments de protocole inconnus, éléments de protocole valides mais dans un contexte incorrect, ...) ;
- les erreurs de service (primitives de service non attendues, primitives de service attendues mais avec des paramètres invalides ou non spécifiés).

Ces erreurs sont obtenues grâce au générateur. En effet, les erreurs de protocole sont produites dans le centre de test lors de la construction des éléments de protocole et sont transmis à l'implémentation à tester via la connexion de test. Les erreurs de service sont également générées dans le centre de test et sont communiquées à l'implémentation à tester via le répondeur et la connexion de service. Il est aussi possible d'induire des retards dans la transmission des éléments de protocole afin de vérifier la manipulation des timers par l'IAT.

4.2.5.4. Méthodes de réalisation des tests

L'exécution des tests dans le centre de test est basée sur le fait qu'après chaque envoi d'un message sur la connexion de service (CS) ou sur la connexion de test (CT), tout autre envoi ne peut avoir lieu sans avoir reçu une réponse à ce message ou un signal d'expiration de timer. Cette réponse peut être un message reçu sur la CS ou la CT. Un scénario dans le cadre de l'architecture STP (System for Testing Protocol) est une suite de commandes devant être exécutées par le générateur. Ces commandes représentent les événements attendus ou à générer sur les deux connexions, et ce, afin de contrôler le comportement de l'IAT à différents moments (établissement d'une connexion, transfert des données (normales ou express), fermeture d'une connexion).

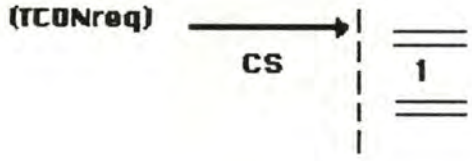
De tels scénarios sont obtenus de deux façons :

a) la méthode du "*testing graph*"

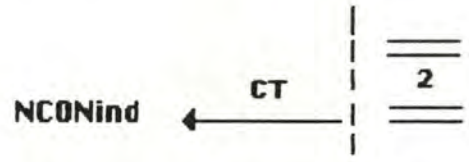
Le "testing graph" est un graphe orienté, composé d'un ensemble fini de noeuds reliés entre eux par des arcs. Chaque noeud spécifie soit l'envoi ou la réception d'un message, sur la connexion de test ou la connexion de service, soit l'expiration d'un timer.

Shématiquement, un noeud est représenté par un numéro et soit par un flèche "<--" pour indiquer la réception d'un message ou par une flèche "-->" pour indiquer l'émission d'un message. La survenance d'un événement (réception ou envoi) sur la connexion de service est signalée par une flèche (<-- ou -->) sur laquelle figure la mention "**CS**" tandis que la survenance d'un événement sur la connexion de test est signalée par une flèche (<-- ou -->) sur laquelle figure la mention "**CT**". Les messages entourés de parenthèses signifient qu'ils sont envoyés ou reçus par des primitives de transfert de données de niveau (N-1).

Ainsi par exemple,



signifie que l'on accède au noeud 1 du graphe de la figure 4.16 lorsque le générateur envoie (flèche -->) sur la connexion de service (CS) la primitive T_CONNECT.request. Par contre, on accède au noeud 2 présenté à la figure 4.16,



si le générateur reçoit (flèche <--) sur la connexion de test (CT) une primitive N_CONNECT.indication. En fonction des événements et produits par le générateur, le parcours des différents arcs du graphe se fera différemment, ce qui correspondra à la production des divers scénarios de test.

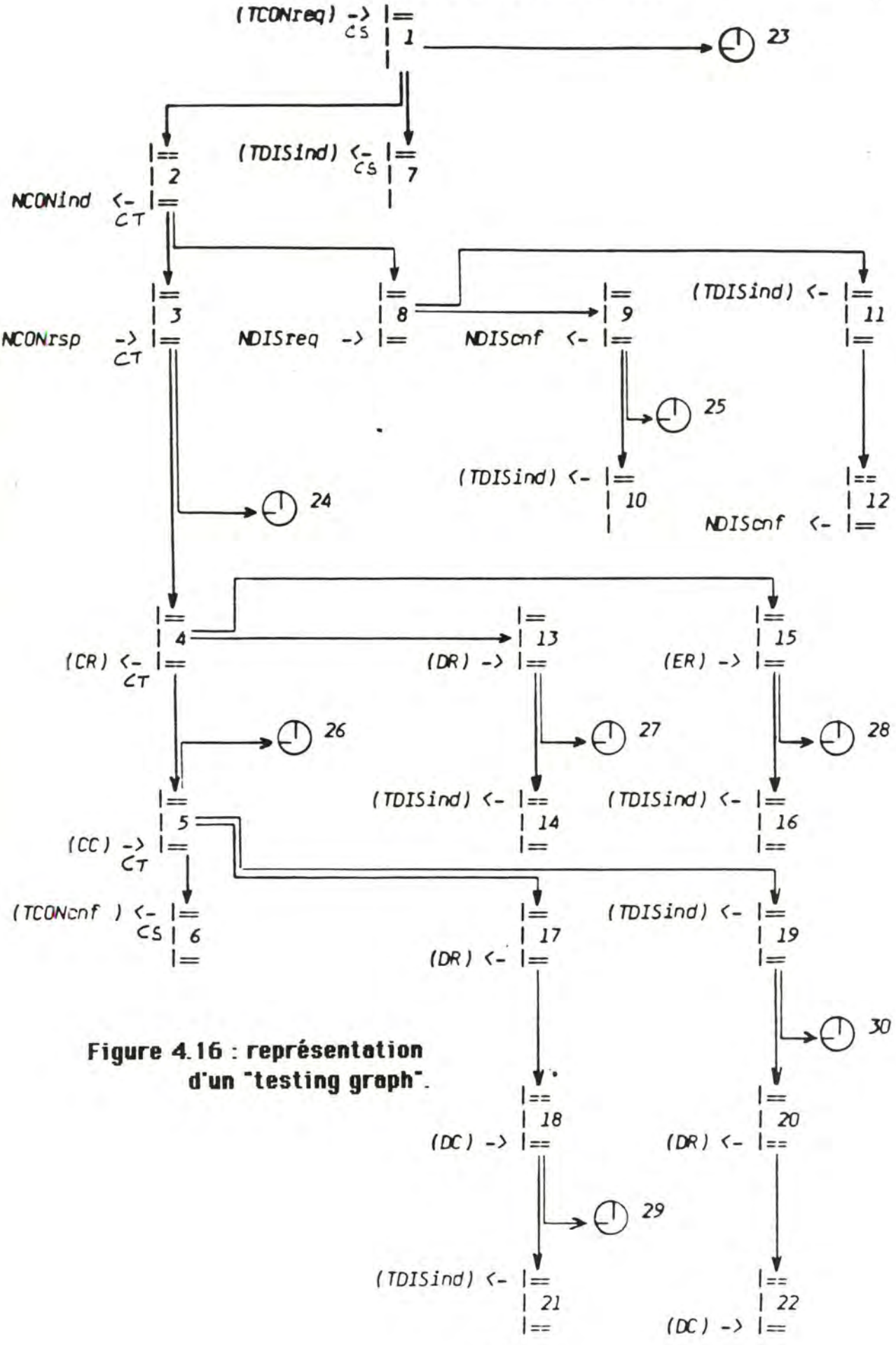


Figure 4.16 : représentation d'un "testing graph".

Par construction, le générateur après avoir envoyé un message sur une connexion ne peut qu'attendre l'arrivée d'un message en provenance du site client. Dans notre cas, la primitive T_CON.request sera transmise via la connexion de service à l'IAT. Celle-ci devra réclamer l'établissement d'une nouvelle connexion (par un N_CONNECT.request), qui sera en fait la nouvelle connexion de test. Cette demande d'établissement sera signalée au centre de test par un N_CONNECT.indication. Dès ce moment, le noeud 2 du graphe est atteint. Le générateur répond sur cette connexion par un N_CONNECT.response (noeud 3). Cette réponse parviendra à l'IAT sous forme d'un N_CONNECT.confirm. La connexion réseau est maintenant établie. L'IAT va générer un élément de protocole d'établissement de connexion de transport (CR_TPDU) contenant les paramètres spécifiés dans le T_CONNECT.request. Cet élément de protocole est envoyé sur la connexion de test venant d'être construite à l'aide d'une primitive réseau de transfert de données. Lorsque ces données sont reçues par le générateur, le noeud 4 est atteint. Le générateur construit ensuite un élément de protocole de confirmation de connexion de transport (CC_TPDU) et l'envoit sur la connexion de test (noeud 5).

Lorsque cet élément de protocole sera reçu par l'IAT, celle-ci devra signaler à son utilisateur (c'est-à-dire le répondeur) par la primitive T_CONNECT.confirm que la connexion de transport a pu être établie. Lorsque cette primitive sera communiquée au générateur, par le répondeur et via la CS, le test sera terminé, le noeud 6 (noeud terminal) étant atteint.

4.3.6. L'architecture de GMD

4.3.6.1. Introduction

La GMD (Gesellschaft für Mathematik und Datenverarbeitung) a mis au point une architecture pour tester les implémentations de protocole, dans le cadre du projet TESDI (TESting and Diagnosis aid for high level protocols) entamé en mars 1981.

La GMD mène ce projet en collaboration avec le NPL et s'intéresse plus particulièrement au test des implémentations de la couche session (niveau 6).

L'architecture proposée par la société allemande doit aider les réalisateurs d'un protocole lors du développement de leur implémentation et doit pouvoir établir, à la demande de ceux-ci, la conformité de l'implémentation élaborée par rapport au protocole standard (" Arbitration testing "). C'est dans cette optique que sera étudiée cette architecture.

4.3.6.2. Graphe de l'architecture

Hypothèses

- les services de test sont offerts par la GMD via le réseau public allemand de communication par paquets DATEX-P ;

- aucune exigence n'est imposée sur les caractéristiques du système où est localisée l'implémentation à tester ;

- l'utilisateur du système GMD qui désire tester son implémentation a la responsabilité de la gestion du test (sur le site client et le centre de test). Aucune personne autre que lui-même n'est requise pour suivre l'évolution du test. Cet utilisateur dispose pour réaliser les tests :

1. d'un terminal de contrôle relié, via DATEX-P au centre de test. Ce terminal lui permet de commander et de suivre l'exécution des tests ;

2. d'une console reliée directement au site client. Cette console lui permet de commander et de suivre l'exécution des tests.

Architecture

L'architecture de la GMD est représentée globalement à la figure 4.17. Dans cette figure, on peut remarquer les trois éléments de base constituant cette architecture décentralisée :

- le centre de test, contenant l'implémentation servant d'implémentation de référence,
- le site client, contenant l'implémentation à tester,
- le terminal de contrôle, à partir duquel l'opérateur engagera les différents tests de conformité et suivra leurs exécutions.

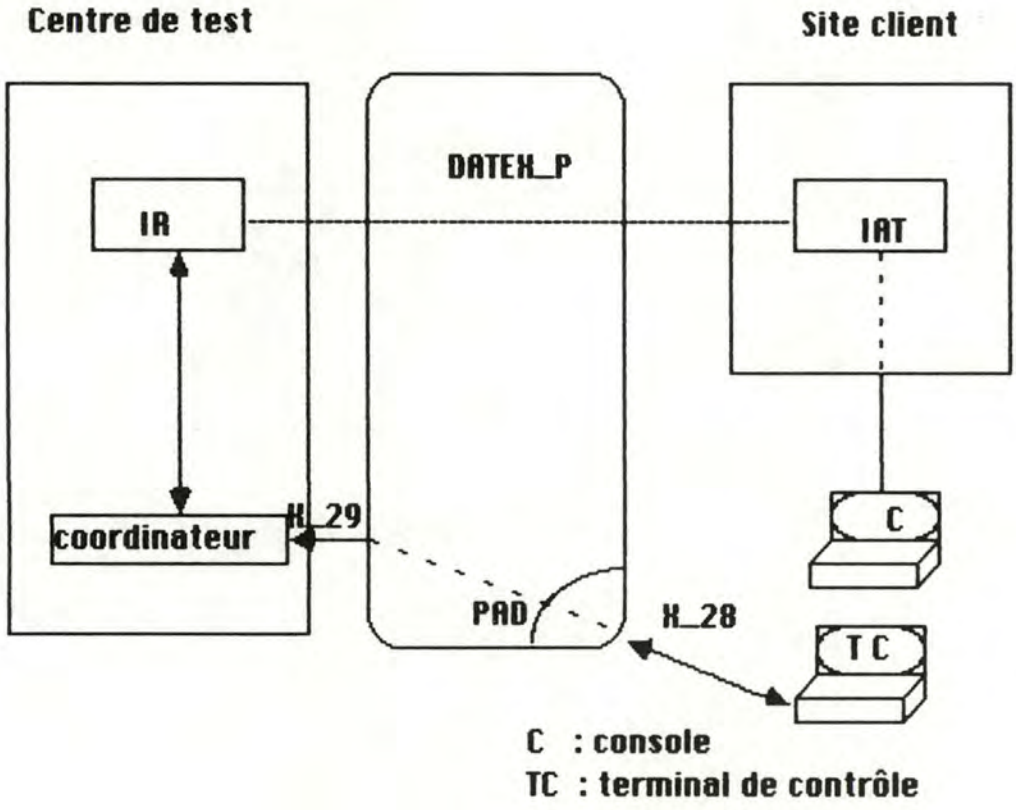


Figure 4.17 : présentation globale de l'architecture de la GMD.

Les composants contenus dans le centre de test et le site client sont détaillés respectivement dans les figures 4.18 et 4.19 et sont expliqués dans la section suivante.

4.3.6.3. Analyse des composants de l'architecture

Composants du centre de test

Le centre de test illustré à la figure 4.18 possède deux composants principaux, à savoir le **testeur actif** et l'**ordinateur central**. Comme son nom l'indique, c'est dans le testeur actif que se déroule l'essentiel de l'activité de test. L'ordinateur central offre surtout de grandes capacités de stockage d'informations dont ne dispose pas le testeur actif.

TESTEUR ACTIF

Dans la figure 4.18, la configuration du testeur actif est représentée dans le cas où l'implémentation à tester est celle d'un protocole de transport. On peut y distinguer :

1. **Module X25**

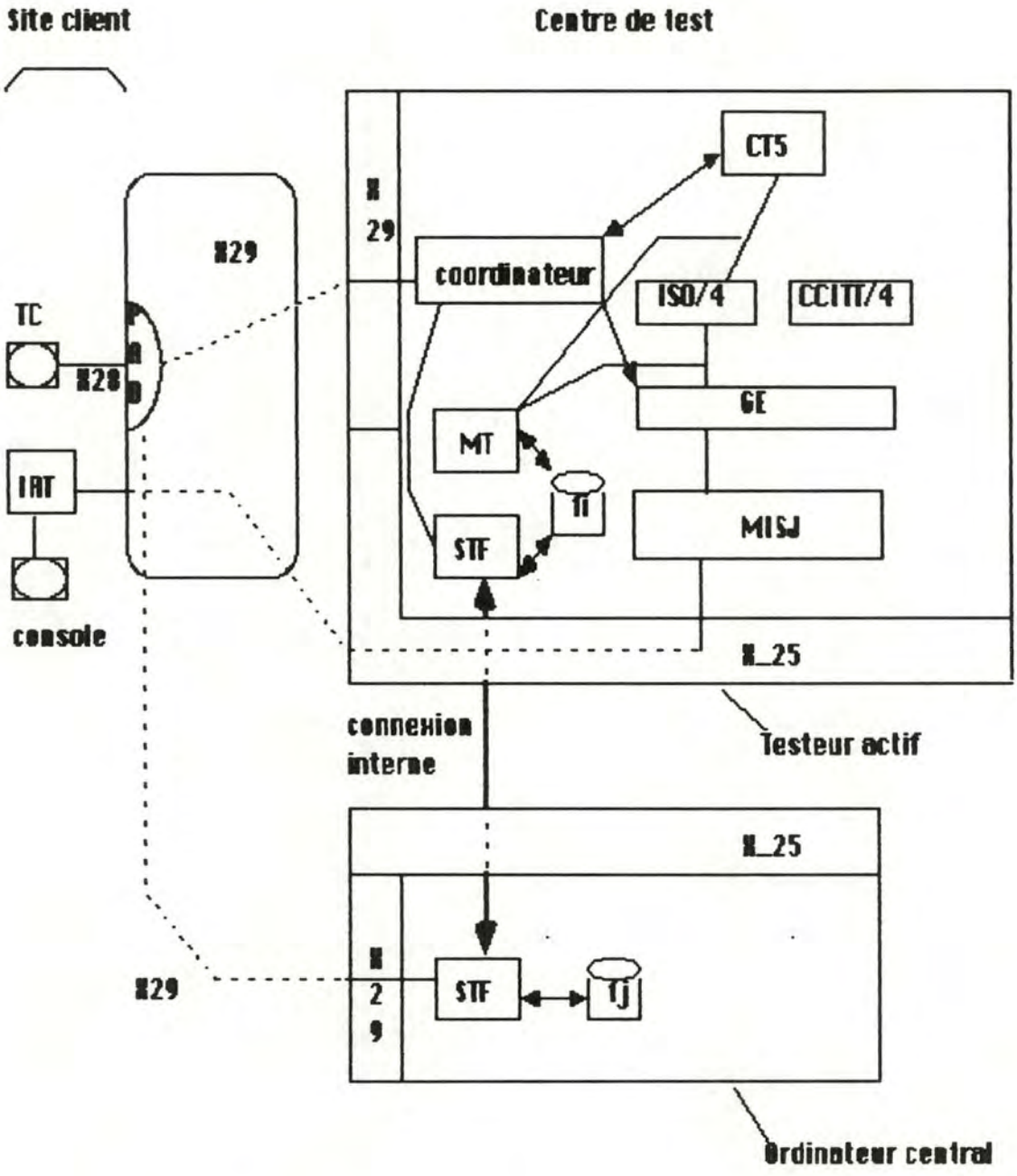
Ce module permet au testeur actif d'établir des connexions de type X25 :

- avec le site client pour maintenir (ou créer) une connexion de test avec l'implémentation à tester ;
- avec l'ordinateur central pour permettre le transfert des résultats de test ; la connexion utilisée entre ces deux composants du centre de test est appelée connexion interne.

2. **Module X29**

Rappelons que l'avis X29 du CCITT règlemente l'échange de commandes et de données entre un équipement terminal de traitement de données (ETTD) fonctionnant en mode paquet et un système d'assemblage et de désassemblage de paquets (PAD).

Ce module gère l'échange d'informations entre le testeur actif et le PAD auquel est relié le terminal de contrôle.



STF : système de transfert de fichiers

fj : fichier journal
fi : fichier intermédiaire

MISJ : module des implémentations sous-jacentes

GE : générateur d'exceptions

MT : module des traces

CT5 : conducteur de tests (implémentation de niveau 5)

Figure 4.18 : composants du centre de test.

3. Système de transfert de fichiers (STF)

Ce module réalise le transfert de fichiers en communiquant avec son homologue dans l'ordinateur central. Ce transfert est enclenché automatiquement par le coordinateur de test (voir point 4 ci-dessous), lorsque la fin du test lui aura été signalée par l'opérateur. Ceci évite des interférences entre le trafic sur la connexion de test et le trafic relatif au transfert de fichiers (ralentissement du transfert des éléments de protocole dû à une activité supplémentaire du module X25).

4. Coordinateur de test

A ce niveau une remarque concernant la réalisation des tests s'impose. En effet, il existe deux stratégies permettant de mener les tests sur cette architecture, à savoir :

a) les tests **manuels**, dans lesquels le conducteur de test est contrôlé manuellement par l'opérateur grâce à des commandes de test ;
b) les tests **automatiques** dans lesquels le conducteur de test est contrôlé par des commandes extraites d'un fichier.

Dans cette étude, seule la méthode des tests manuels sera abordée, la mise en oeuvre de l'autre méthode constituant une voie de recherche ultérieure. Ainsi, les commandes de tests seront communiquées par l'opérateur au coordinateur, une à une, via le module X29. Les fonctions réalisées par le coordinateur sont :

- la vérification syntaxique des commandes de tests transmises par l'opérateur et la transmission de celles-ci au conducteur de test ;
- l'envoi de messages concernant le déroulement du test sur le terminal de contrôle de l'utilisateur ;
- la configuration de l'environnement de test, qui consiste en :

  * la sélection d'une implémentation de référence-(N) et du conducteur de test-(N) associé en fonction d'une commande d'initialisation de test. En effet, plusieurs implémentations de référence sont disponibles pour tester une même couche de l'OSI. Par exemple, on peut vérifier la conformité d'une implémentation par rapport à une implémentation de référence du CCITT ou par rapport à une implémentation de référence de l'ISO. Le choix de l'implémentation de référence s'effectue en fonction du protocole qui a servi de base à l'implémentation à tester.

* les autres implémentations de référence sous-jacentes à l'implémentation de référence-(N) sélectionnée et non comprises parmi les implémentations du module X25 sont mises en oeuvre par le coordinateur pour mener à bien les tests.

5. Conducteur de tests (CT)

Le conducteur de tests est considéré comme l'utilisateur de l'implémentation de référence-(N). Son rôle est de :

- générer les primitives de demande de service-(N) à partir des commandes reçues par le coordinateur (voir point 4 ci-dessus) et de les communiquer à l'implémentation de référence ;

- transmettre les indications de services reçues de l'implémentation de référence au coordinateur.

6. Implémentation de référence

La technique de l'implémentation de référence est utilisée par la GMD pour générer les éléments de protocole-(N). Cependant, par opposition au NBS, où une seule implémentation de référence est disponible par couche testée, GMD met à la disposition des utilisateurs plusieurs implémentations de référence pour une même couche du modèle OSI. (Cela dépend de la disponibilité de standards de protocoles émis par les différents organismes (ISO, CCITT, ...)). L'utilisateur pourra ainsi sélectionner parmi ces implémentations celle qui constituera son implémentation de référence.

7. Module des traces (MT)

Si l'opérateur en a fait la demande au début des tests, ce module provoque l'enregistrement dans un **fichier temporaire (FT)** des événements se produisant aux interfaces (supérieure et inférieure) de l'implémentation de référence.

8. Générateur d'exceptions (GE)

Cet élément permet de modifier le comportement de l'implémentation de référence, notamment :

- par la variation de certains paramètres contenus dans les éléments de protocole produits ;
- par la génération d'éléments de protocole invalides ou valides mais dans un contexte incorrect ;
- en retardant la transmission des éléments de protocole et cela, afin de contrôler les mécanismes de timer mis en oeuvre par l'IAT.

9. Module des implémentations sous-jacentes (MISJ)

Ce module contient les implémentations vérifiées inférieures à celle testée et non comprises dans le module X25. Le module des implémentations sous-jacentes est inutile dans la figure 4.18 étant donné que l'on teste l'implémentation d'un protocole de transport.

ORDINATEUR CENTRAL

L'ordinateur central se compose d'un :

1. Module X25

Ce module permet à l'ordinateur central d'établir des connexions de type X25 avec le testeur actif pour permettre à son **système de transfert de fichiers (STF)** d'entrer en communication avec son homologue sur le testeur actif et ce, pour le transfert du contenu du fichier intermédiaire (tenu à jour dans le testeur actif par le module des traces) dans le **fichier journal (FJ)** de l'ordinateur central.

2. Module X29

Ce module permet l'établissement d'une connexion de type X29 entre l'ordinateur central et le PAD auquel est relié le terminal de contrôle. Cette connexion permettra à l'opérateur, après une session de tests, de lire les événements qui se sont produits pendant les tests, et qui ont été sauvegardés par le système de transfert de fichiers (STF) dans le fichier journal. L'opérateur accède au fichier journal par l'intermédiaire du STF.

COMMANDES DESTINEES AU COORDINATEUR

Ces commandes sont envoyées, via le terminal de contrôle, au coordinateur par l'opérateur qui controle l'exécution du test, "pas à pas". Les commandes sont de trois sortes :

1. Commandes d'initialisation de test

Ces commandes permettent au coordinateur :

- de déterminer l'implémentation de référence et le conducteur de test adéquats, en fonction des besoins de l'opérateur ;
- de mettre en activité le module des traces, si l'opérateur le juge nécessaire.

2. Commandes destinées au conducteur de test

Ces commandes sont traduites par le coordinateur en primitives de service pour l'implémentation de référence.

3. Commandes de génération d'erreurs

Ces commandes sont destinées au générateur d'exceptions et permettent les manipulations énoncées dans la description de ce module (voir point 8 ci-dessus).

Composants du site client

Le site client est illustré à la figure 4.19.

1. Module X25

Ce module permet au site client d'établir des connexions de test de type X25 avec le testeur actif, via le réseau public DATEX-P.

2. Module des implémentations sous-jacentes (MISJ)

Ce module présente la même fonctionnalité que celui du centre de test.

3. Implémentation à tester (IAT)

L'IAT est l'implémentation d'un protocole d'une couche du modèle OSI et pour laquelle on désire établir la conformité par rapport aux spécifications de ce même protocole.

4. Conducteur de test (CT)

Ce composant a pour fonction :

(1) de générer les primitives de demandes de service à l'implémentation à tester à partir des commandes transmises au moyen de la console par l'opérateur ;
(2) de transmettre à la console les indications de service reçues de l'IAT.

5. Le terminal de contrôle et la console

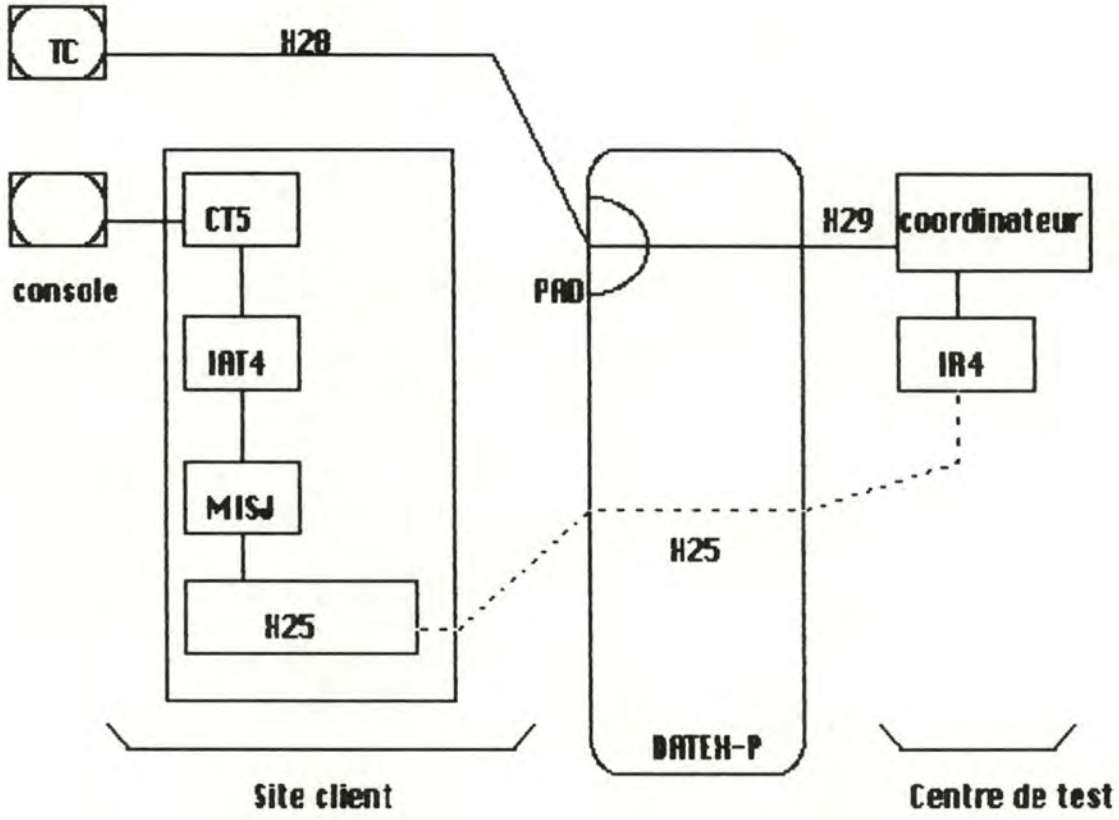
Comme il l'a déjà été signalé précédemment dans les hypothèses, l'opérateur contrôle à lui seul l'exécution des tests sur le site client et le centre de test au moyen respectivement de la console et du terminal de contrôle.

a) le terminal de contrôle

Le terminal de contrôle est relié à un PAD au moyen d'une liaison X28. Ce PAD se charge d'acheminer les données transmises par l'opérateur au centre de test (testeur actif et/ou ordinateur central) en établissant une liaison X29 avec ce dernier.
Rappelons que grâce à ce terminal, l'opérateur peut :

- envoyer au testeur actif des commandes :
  * d'initialisation de test,
  * représentant des primitives de service destinées à l'implémentation de référence,
  * de génération d'erreurs.
- communiquer avec l'ordinateur central afin de prendre connaissance du contenu du fichier journal et de faire l'analyse des événements survenus au cours de la session.

- se rendre compte de l'activité de l'interface supérieure de l'implémentation de référence (notamment en connaissant les indications de service qui surviennent).



IAT4 : implémentation à tester de la couche transport

IR4 : implémentation de référence de la couche transport

CT5 : conducteur de test, représentant la couche session

TC : terminal de contrôle

Figure 4.19 : composants du site client.

b) <u>la console</u>

La console est reliée directement au conducteur de test du site client et permet à l'opérateur :

- d'envoyer des commandes représentant des primitives destinées à l'implémentation à tester,

- de prendre connaissance de l'activité de l'interface supérieure de l'implémentation à tester.

4.2.6.4. Méthodes de réalisation des tests

L'opérateur, au moyen de son terminal de contrôle, établit une connexion avec le coordinateur du centre de test. Il lui communique les directives à suivre lors du déroulement du test de même qu'au fur et à mesure les commandes destinées à l'implémentation de référence.

A l'aide de sa console, il transmet au conducteur de test du site client les commandes d'interface destinées à l'implémentation à tester. Il peut donc contrôler et observer le déroulement des tests à la fois sur le site client et le centre de test par la réception respectivement sur la console et le terminal de contrôle de messages représentant des indications de service. Ces indications de service sont générées par l'implémentation à tester ou par l'implémentation de référence.

Une fois le test terminé, l'opérateur peut entrer en communication avec l'ordinateur central du centre de test pour obtenir la liste des événements survenus au cours du test dans le but d'en faire une analyse.

Dans l'exemple suivant les commandes introduites par l'utilisateur sur son terminal de contrôle sont précédées d'un *, celles introduites à la console sont précédées d'un □ ; les messages reçus sur le terminal de contrôle ou sur la console sont respectivement précédés de ** ou □□.

1. **Initialisation du test**

* LOGON 4567658954	, demande de connexion avec le coordinateur;
* TRACE FILE	, demande d'enregistrement des événements dans un fichier ;
* MANUEL ISO4	, le test sera du type manuel et l'implémentation de référence utilisée sera celle de la couche transport de l'ISO.

2. Exécution du test

Le test qui suit a pour but d'établir, à partir du centre de test, une connexion avec l'implémentation que l'on teste.

* CONREQ 1265432879 , T_CONNECT.request destiné à l'implémentation de référence pour établir une liaison avec l'implémentation se trouvant à l'adresse indiquée ;

□□ CONIND 1287539002, message T_CONNECT.indication signalant à l'opérateur une demande de connexion en provenance de l'adresse spécifiée ;

□ CONRES 1287539002, commande destinée à l'implémentation que l'on teste et représentant une primitive de service T_CONNECT.response ;

** CONCON 1287539002, message signalant la réception sur l'interface supérieure de l'implémentation de référence d'un T_CONNECT.confirm.

5. COMPARAISON DES DIFFERENTES ARCHITECTURES DECENTRALISEES

5.1. Prérequis d'une bonne architecture : critères de comparaison

Après avoir analysé, dans le cadre du chapitre 4, diverses architectures pour les tests de conformité des protocoles de haut niveau (4 et supérieurs), nous nous proposons maintenant de comparer celles-ci.

Pour ce faire, il convient de définir les critères susceptibles d'argumenter les jugements apportés. A ce titre, les éléments suivants pourront être pris en considération pour l'évaluation d'un système de test :

la généralité : le système doit être conçu de manière suffisamment générale (portable) afin de pouvoir tester un maximum d'implémentations d'une même couche, peu importe la façon dont une implémentation est réalisée (hardware ou software), le matériel ou le système d'exploitation sous-jacent à l'implémentation. Ainsi, le système de test et en particulier les éléments de celui-ci à insérer au niveau du site client devront poser un minimum de contraintes sur ce site.

le caractère explicatif : le système ne doit pas seulement permettre d'affirmer si oui ou non une implémentation est conforme à un standard mais il doit surtout fournir au client les informations qui ont permis d'établir ce jugement. Il s'agit entre autres de la liste des tests menés et de leurs résultats ainsi que l'indication d'éventuels problèmes survenus. Etant donné que ces informations s'adressent au réalisateur des tests, leur présentation doit être la plus claire et la plus significative possible.

le caractère exhaustif : le système de test doit offrir la possibilité de mener les différents types de tests nécessaires à la vérification de conformité d'un protocole. Une architecture pourra donc être évaluée quant aux limitations qu'elle impose à ce niveau, c'est-à-dire quant aux types d'erreurs qu'elle permet ou qu'elle ne permet pas de détecter.

la facilité d'utilisation/automatisation : les tests doivent pouvoir être menés sans qu'il soit nécessaire de faire appel de façon permanente à un responsable du centre de test. Pour ce faire, le système doit être automatisé le plus possible, facile d'emploi et disposer d'une bonne documentation. L'automatisation est un facteur-clé dans les problèmes de vérification de conformité où le nombre de tests à effectuer est souvent très important.

la facilité d'implémentation des éléments du site client : le système de test doit minimiser la complexité du ou des éléments spécifiques au test devant s'insérer sur le site client. Un équilibre doit être trouvé entre ce critère et le critère d'exhaustivité.

la modularité : tout comme il est primordial d'élaborer un système composé de "blocs" relativement indépendants qui correspondent à chacune des couches de l'architecture (ceci est d'autant plus vrai que les spécifications de certaines couches peuvent subir des modifications au cours du temps alors que d'autres resteront inchangées), cette propriété doit également être vérifiée pour l'ensemble des composants de l'architecture.

l'adaptabilité : le système de test doit pouvoir tenir compte de l'évolution des exigences du client, des modifications des standards de protocoles et des techniques hardware et de communication.

la possibilité de réutilisation : ayant testé une couche de niveau-(N) du modèle ISO/OSI, le système de test doit être tel qu'il permette une réutilisation d'un maximum de composants (ou d'une partie des composants) pour la réalisation de tests de la couche de niveau-(N+1). Cette propriété, fortement dépendante des deux précédentes, est très importante pour le(s) composant(s) du site client.

Les différents critères d'évaluation venant d'être cités peuvent être utilisés pour comparer les architectures proposées pour les tests décentralisés.

5.2. Evaluation des architectures étudiées

Reprenant les divers critères d'évaluation définis dans le cadre des prérequis d'une bonne architecture, une classification des architectures décentralisées étudiées peut être réalisée. Dans celle-ci, nous n'avons pas repris l'architecture proposée par ADI pour laquelle certaines options importantes n'ont pas encore été spécifiées. Pour chacun des critères retenus (certains de ceux-ci ont été regroupés étant donné leur forte dépendance), une appréciation quant au respect de ce critère est donnée pour toutes les architectures (NBS/fig 4.6, NPL/fig 4.7, BULL/fig4.15, GMD/fig4.18). Les notations suivantes ont été adoptées :

- \- : propriété non vérifiée ;
- \+ : propriété satisfaite ;
- ++ : architecture vérifiant le mieux le critère.

Architecture / Critères	NPL	NBS	BULL	GMD
Généralité	++	−	−	+
Caractère explicatif	+	++	+	+
Caractère exhautif	++	+	+	+
Facilité d'utilisation (automatisation)	+	+	+	−
Facilité d'implémentation	+	+	−	++
Réutilisable (modularité/ adaptabilité)	−	−	−	−

Figure 5.1 : tableau comparatif des architectures décentralisées.

Remarque : les différentes lignes du tableau sont relativement dépendantes : une note "++" peut être la cause d'une ou plusieurs notes "-" et inversément. Ainsi, la note "++" pour le caractère explicatif de l'architecture du NBS (possibilité de datation des événements) est une des causes de la note "-" au niveau de sa généralité (nécessité d'une horloge interne).

Généralité

* NBS (-) : l'architecture NBS impose sur le site client diverses contraintes matérielles dont la principale est certainement la possibilité de stockage d'informations (+ de 100K sont nécessaires pour le sauvetage des scénarios et des résultats des tests) sur mémoires auxiliaires. Une telle contrainte réduit le domaine d'application de l'architecture notamment pour les tests de protocoles implémentés sur processeur "front-end" ne disposant généralement pas de telles ressources.

* BULL (-) : basée sur un répondeur à double niveaux nécessitant un accès non seulement à l'IAT mais également à une couche de niveau inférieur, l'application de l'architecture BULL se voit limitée aux systèmes présentant cette disposition.

* GMD (+) : la simplicité du répondeur (consistant en une simple interface) fait qu'aucune disposition ni contrainte particulière ne sont requises.

* NPL (++) : l'utilisation d'un protocole de test évitant le stockage d'informations sur le site client et la définition d'un comportement par défaut pour le répondeur afin de lui permettre de réaliser les tests initiaux assurant la fiabilité des tests ultérieurs, apportent une solution aux exigences des architectures NBS et BULL.

La section 5.3.1 commente de manière plus approfondie les répondeurs de chacune des architectures ainsi que les contraintes qu'ils imposent.

Comparaison des architectures décentralisées

Caractère explicatif

* Chacune des architectures met à la disposition du réalisateur de test des fichiers journaux reprenant l'évolution des tests menés. Les informations du fichier journal de l'architecture NBS sont d'autant plus précises que cette architecture offre un dispositif de datation des événements .

Caractère exhaustif

* Une étude plus complète de l'étendue des possibilités offertes par les différentes techniques à savoir celle de l'implémentation de référence (GMD, NBS), de l'encodeur/décodeur (NPL) et du générateur d'éléments de protocole (BULL), ainsi qu'une comparaison de leur efficacité, est donnée au point 5.3.2.

Facilité d'utilisation/automatisation

* GMD (-) : dans l'architecture GMD, l'activité de l'interface supérieure de l'IAT est suivie et contrôlée intégralement par l'opérateur via une console. Une telle approche peut s'avérer utile lors des premiers tests de l'implémentation où ceux-ci sont relativement courts et interrompus. Cependant, lorsque des tests plus longs et plus complexes doivent être réalisés, la tâche de l'opérateur devient fort ardue.

* NPL - NBS - BULL (+) : aucune distinction n'a été introduite à ce niveau en ce qui concerne les trois architectures, chacune disposant de scénarios facilitant la tâche de l'opérateur. Néanmoins certaines différences peuvent être notées :

- possibilité d'imbrication de scénarios (NPL, NBS) ;
- possibilité d'alternatives dans l'évolution des tests (NPL, BULL).

Facilité d'implémentation

* BULL (-) : la gestion de deux connexions parallèles (une connexion de test et une connexion de service) accroît la complexité du répondeur. De plus, l'accès à deux couches différentes nécessite une interface généralisée dont la réalisation requiert un investissement supplémentaire avant la mise en oeuvre des tests.

* GMD (++) : le répondeur employé étant non intelligent (son unique fonction est de transformer les messages à destination et en provenance de la console), il est aisé à implémenter.

* NPL - NBS (+) : la complexité de réalisation de l'interpréteur de scénarios (NBS) et du répondeur (NPL) est moyenne par rapport aux deux extrêmes que représentent BULL et GMD.

Réutilisable ? (modularité/adaptabilité)

* Quelle que soit l'architecture, deux modules au moins devront être remaniés si on envisage le test d'une autre couche. Il s'agit du testeur et au niveau du site client du répondeur. Ceci signifie que la totalité du travail investi par le client devra être reproduit pour tenir compte des spécificités de la nouvelle IAT. La section suivante proposera une solution permettant de réduire la tâche préparatoire aux tests. Quant aux architectures du NPL et BULL, elles comprennent également un autre composant à modifier en cas de changement de couche à savoir l'encodeur/décodeur (NPL) et le générateur (BULL).

5.3. Comparaison des composants essentiels des architectures

Deux éléments centraux peuvent être dégagés des architectures abordées précédemment :

- le composant du site client agissant en tant que couche utilisatrice (répondeur) des services fournis par l'implémentation que l'on teste ;
- le(s) composant(s) du centre de test placé(s) en parallèle du répondeur et de l'implémentation que l'on teste.

Dans ce qui suit, nous analyserons d'une part (section 5.3.1) les caractéristiques des différents types de répondeurs rencontrés et d'autre part (section 5.3.2), les équivalents possibles de l'implémentation à tester pour le centre de test.

5.3.1. L'architecture du répondeur

Les différentes architectures analysées dans l'approche décentralisée exigent, comme condition première à leur application, que l'implémentation à tester (IAT) possède une interface vers le haut utilisable afin d'accéder au service fourni par cette IAT. Le comportement de la couche utilisant cette interface doit nécessairement être prévisible de manière à ce que les résultats des tests dépendent uniquement du seul composant de l'architecture restant à comportement indéterminé, à savoir l'IAT. C'est pourquoi chacune des architectures décrit un composant, le **répondeur** (repris sous différentes appellations dans les architectures analysées : Répondeur (NPL - ADI - BULL), Interpréteur de scénarios du site client (NBS), Conducteur de test (GMD)) qui est incorporé au sein du site client et qui n'est utilisé uniquement lors de la réalisation des tests.

Il convient de remarquer que l'implémentation du répondeur dépend de la réalisation locale de l'interface vers le haut permettant d'accéder aux services fournis par l'IAT. Etant donné que les détails de cette interface sont propres au système local, le répondeur est défini uniquement en termes de services (requête, indication, confirmation, réponse), ce qui constitue une abstraction de cette interface.

De son côté, le réalisateur des tests devra, sur base des spécifications du répondeur en termes de services, implémenter une interface entre son répondeur et l'IAT. Cette méthode évite aux centres de test d'avoir à connaître les détails de chaque interface locale.

Le répondeur doit être aussi simple que possible puisqu'il doit pouvoir être inséré aisément dans tout système que l'on désire tester en imposant un minimum de contraintes sur celui-ci. D'autre part, il devra permettre la réalisation de tous les tests que l'on désirerait mener. Ces deux objectifs antinomiques de flexibilité et de généralité ont donné naissance à diverses approches qui vont être rappelées et comparées dans ce qui suit.

La structure du répondeur est fortement influencée par les suppositions qui peuvent être faites au sujet de l'environnement dans lequel il sera implanté. L'objectif de la comparaison qui suit est d'évaluer quatre approches décrites au chapitre 4 et ce, en fonction des cinq critères suivants :

(1) Possibilité de stockage d'informations sur mémoires auxiliaires sur le site client ;
(2) Existence d'une horloge sur le site client ;
(3) Existence d'une interface vers un terminal de contrôle ;
(4) Possibilité pour le répondeur de traiter plusieurs connexions simultanées ;
(5) Utilisation d'un protocole entre le testeur et le répondeur.

Le répondeur manuel (GMD)

La première approche, adoptée par la GMD, consiste en un répondeur manuel. Cette technique est basée sur une interface vers une console sur laquelle viennent s'afficher les différentes primitives de service reçues par l'IAT. En fonction de ces primitives, un opérateur détermine les réponses associées à envoyer et évalue les résultats des tests.

Cette approche ne requiert que la contrainte (3) et n'impose aucune restriction sur l'espace mémoire disponible. Cependant, en raison du nombre important de tests et de la lenteur de ce procédé, celui-ci, lorsqu'il est utilisé, reste très limité quant à la portée des tests réalisés ("debugging phase").

Le répondeur guidé par scénarios (NBS)

La seconde approche développée au NBS établit une relation de similitude dans le fonctionnement du testeur (interpréteur de scénarios du centre de test) et celui du répondeur (interpréteur de scénarios du site client) en ce sens que le comportement de chacun d'eux est régi par des séquences de commandes. Ces séquences, reprenant exclusivement des primitives de service, sont appelées scénarios. Le scénario utilisé pour guider le comportement du répondeur doit être complémentaire à celui utilisé pour guider le testeur (par exemple, si l'un d'eux contient une primitive de type request, l'autre contiendra la primitive correspondante du type indication).

La synchronisation entre ces deux composants peut être réalisée de différentes manières :

- soit que l'on utilise un protocole très simple entre le testeur et le répondeur ; ce protocole permetterait de transférer le nom du prochain scénario à utiliser (approche non retenue au NBS car elle complique le testeur) ;
- soit que l'on utilise un scénario dit "maître" communiqué à chacun des sites et définissant l'ordre dans lequel seront réalisés les tests ;
- soit que l'on utilise une communication téléphonique entre les opérateurs situés sur chacun des sites (approche retenue pour l'instant).

L'avantage d'une telle méthode est avant tout sa simplicité d'utilisation. Les scénarios exécutés sont aisément compréhensibles.

De plus, ils permettent une grande flexibilité vu que d'une part, ils sont facilement modifiables par simple édition d'un fichier et d'autre part, ils sont paramétrables, notamment au niveau des adresses (ce qui les rend adaptables). Cette simplicité d'utilisation se traduit par une simplicité au niveau de la structure interne du répondeur d'autant plus qu'aucun protocole de test n'est requis dans cette approche. L'analyse des résultats est également aisée puisque des fichiers résultats fournissent les événements auxquels a été soumise l'IAT, les réponses qu'elle a fournies ainsi que le moment de survenance des divers événements (cette datation des événements impose la contrainte (2)).

Par contre, l'un des principaux inconvénients de l'architecture proposée par le NBS est qu'elle requiert de grandes possibilités de stockage (± 100K octets) pour les scénarios et les résultats des tests au niveau du site client (contrainte (1)).

Ceci réduit son domaine d'application notamment pour les tests de protocoles implémentés sur processeur "front-end" ne disposant généralement pas de telles ressources. Un protocole de test permettant de transférer les scénarios et les résultats des tests entre le testeur et le répondeur (et inversément) apporterait certainement une solution à cet inconvénient mais irait à l'encontre de la simplicité caractérisant cette approche. Une autre limitation des scénarios actuels est qu'ils ne permettent pas d'exprimer les alternatives. L'approche du répondeur guidé par scénarios n'impose pas les contraintes (3), (4) et (5).

Le répondeur à deux connexions (BULL - ADI)

La troisième approche adoptée par la firme Bull ainsi que dans le projet RHIN de l'ADI utilise une connexion de service (CS) entre le testeur et le répondeur (Strictement parlant, on ne peut parler de protocole de test étant donné qu'il n'y a pas échange de commandes et réponses). L'objectif de cette connexion est de permettre au testeur de communiquer au répondeur quel doit être son comportement lors de la survenance de chaque événement (le testeur transmet une à une les réponses que devra fournir le répondeur). D'autre part, cette connexion de service est également utilisée par le répondeur pour transmettre au testeur les primitives que l'IAT lui a fournies afin d'analyser, sur le centre de test, les résultats des tests.

L'utilisation de la connexion de test évite le stockage d'informations au niveau du site client et supprime la contrainte (1). Néanmoins la connexion de service étant utilisée pour contrôler les tests menés en parallèle sur d'autres connexions (connexions de test CT), cette approche suppose que le répondeur puisse traiter plusieurs connexions simultanément (au moins deux : CT et CS). L'organisation cohérente des différentes connexions accroît la complexité du répondeur. Cette approche n'impose donc que la contrainte (4).

Le répondeur avec protocole de test (NPL)

L'approche développée au NPL est basée sur l'utilisation d'un protocole de test entre le testeur et le répondeur. Ce protocole est utilisé d'une part, pour fixer le comportement du répondeur et d'autre part, pour obtenir les résultats des tests. La synchronisation entre les deux composants est établie en ayant recours à une commande du protocole de test (changer le mode) qui permet au testeur de paramètrer le comportement du répondeur en définissant les réponses qu'il doit fournir aux événements pouvant survenir (le répondeur fonctionne comme un automate dont les outputs sont définis par le testeur). L'application du répondeur est restreinte à une connexion à la fois. Conceptuellement, des connexions parallèles seront donc traitées par différentes "copies" d'un même répondeur. Au niveau du site client, cela pourra être implanté comme différentes "copies" d'un programme non réentrant ou comme un programme réentrant que plusieurs processsus exécutent.

Cette approche qui ne requiert aucun des points (1) à (4) pose un minimum de contraintes sur le système à tester. Elle maximise ainsi le nombre d'implémentations pouvant être testées selon ce principe. La mémorisation d'informations sur le site client est limitée par l'utilisation du protocole de test.

Cependant, deux problèmes inhérents à l'utilisation d'un protocole de test se posent dans l'approche NPL. Tout d'abord, un tel protocole ralentit l'exécution des tests puisqu'un certain nombre d'informations (paramètrisation du répondeur et transfert des résultats) doivent être transmises entre le répondeur et le testeur afin d'éviter un stockage d'informations sur le site client. Notons que ce phénomène de ralentissement est également présent pour les architectures Bull et ADI où une connexion de service est utilisée. D'autre part, l'utilisation du protocole n'est pas immédiatement effective (un certain nombre de tests doivent avoir été réalisés auparavant). En effet, les éléments de ce protocole empruntent une "voie traversant l'IAT".

Cette implémentation doit donc avoir été testée afin de vérifier sa possibilité de répondre correctement aux demandes d'établissement de connexion et à la réception de données et ce afin que les éléments du protocole de test puissent être acheminés correctement. Ces tests appelés tests de base peuvent être réalisés en initialisant le répondeur selon un mode par défaut.

5.3.2. Les architectures implémentation de référence, encodeur/décodeur et générateur d'éléments de protocole.

Dans la plupart des architectures étudiées, les deux principaux composants du centre de test sont :

- le conducteur de test qui se situe, dans l'architecture, au même niveau que le répondeur de test du site client (décrit au point 5.3.1).
- le module de production et de reconnaissance d'éléments de protocole à transmettre ou à recevoir de l'implémentation à tester. Ces deux services (production et reconnaissance) sont offerts au conducteur de test.

De manière plus générale, les différents types de services fournis par ce module devraient permettre de :

1) générer et décoder toute séquence valide d'éléments de protocole;
2) générer des éléments de protocole syntaxiquement valides mais dans un contexte incorrect ;
3) décoder des éléments de protocole syntaxiquement valides mais dans un contexte incorrect ;
4) produire des éléments de protocole syntaxiquement invalides ;
5) interpréter des éléments de protocole syntaxiquement invalides ;
6) simuler, à l'interface inférieure de l'implémentation que l'on teste, des événements inattendus.

Trois méthodes concernant la réalisation de ce module, à savoir celle de l'**implémentation de référence (1)**, celle de l'**encodeur/décodeur (2)** et celle du **générateur (3)** ainsi que leurs avantages et désavantages respectifs, sont analysées ci-dessous.

(1) Implémentation de référence

L'implémentation de référence est une implémentation déjà testée et correcte du protocole à partir duquel est écrite l'implémentation que l'on teste. Cette technique a été adoptée par le NBS et la GMD. Il apparaît que seuls les services du type 1 peuvent être assumés par l'implémentation de référence. Tout message autre que ceux entrant dans cette première catégorie sera rejeté par l'implémentation de référence utilisée.

Il sera notamment impossible pour celle-ci de transmettre à l'implémentation que l'on teste des éléments de protocole invalides ou hors contexte. Néanmoins, ces services de type 2 et 4 peuvent être fournis à condition de jumeler à l'implémentation de référence un générateur d'exceptions. Celui-ci peut être combiné de différentes manières avec l'implémentation de référence, comme le montre la figure 5.2 :

- **générateur à effet interne** : de cette façon, le générateur d'exceptions a accès à certaines variables internes de l'implémentation de référence et peut, sous les directives du conducteur de test, fausser le comportement de cette implémentation en modifiant la stratégie de production des éléments de protocole (par exemple par une modification de la structure des headers, ou une modification du contenu des variables d'adresse, etc...). Le générateur à effet interne est illustré à la figure 5.2(a) ci-dessous. Une telle disposition du générateur d'exceptions présente trois inconvénients.

* Tout d'abord, l'implémentation de référence est régulièrement soumise à des modifications demandées par le générateur d'exceptions, cela va à l'encontre de l'idée d'utiliser une implémentation correcte à laquelle il faut se référer, pour établir la conformité d'une autre implémentation.

* De plus, cette méthode nécessite un nombre considérable de manipulations. En effet, si le test prévoit la génération d'éléments de protocole invalides, cela suppose une première modification des variables internes de l'implémentation de référence. Si ensuite, le test doit se poursuivre par la génération d'éléments de protocole maintenant valides, il s'agit de rétablir (deuxième modification) l'état initial et correct de l'implémentation. La succession de ces manipulations rend ardue la conception de tels tests.

* Enfin, étant donné que les manipulations sont nombreuses, il en résulte que l'exécution de tels tests se caractérise par une grande activité de l'interface entre le conducteur de test et le générateur d'exceptions, interface par l'intermédiaire de laquelle transitent les commandes.

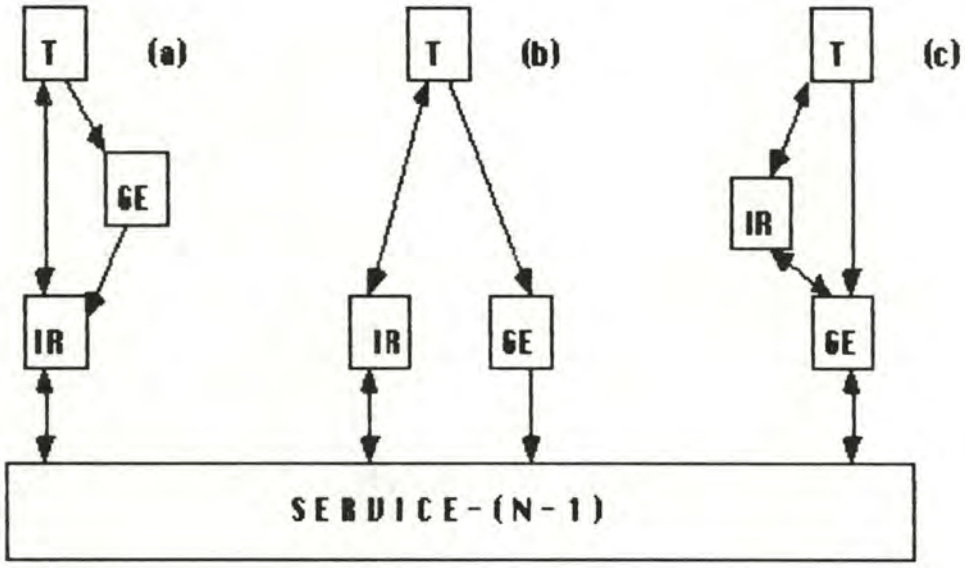
- **générateur à effet parallèle** : selon cette technique, le générateur d'exceptions a un accès direct au service-(N-1) utilisé par l'implémentation de référence et offre la possibilité au conducteur de test de court-circuiter l'implémentation de référence en générant n'importe quel élément de protocole dans n'importe quel contexte. Cependant, cette configuration pose un problème de coordination des requêtes transmises à l'implémentation de référence et au générateur.

La figure 5.2(b) ci-dessous représente ce type de générateur. Un générateur de ce type est employé par la GMD.

- **générateur à effet postérieur** : dans cette configuration, le générateur manipule aussi bien les éléments de protocole transmis que ceux reçus par l'implémentation de référence. Le générateur joue donc le rôle d'intermédiaire entre l'implémentation de référence de la couche transport (4) et l'interface supérieure de la couche réseau (3). Il n'entre en action que sous les directives qui lui sont transmises par le conducteur de test. Ses trois fonctions principales sont les suivantes :

  - simuler, de façon contrôlée, certaines erreurs pouvant être générées par la couche-(N-1) : perte, duplication, livraison hors séquence ...
  - introduire des erreurs dans le codage des éléments de protocole (suppression, ajout, modification de certains champs,...) ;
  - enregistrer sur un support magnétique les éléments de protocole échangés au niveau de l'interface réseau/transport afin d'en faire une analyse ultérieure. Cette analyse aura lieu lorsque le comportement de l'implémentation à tester n'est pas celui attendu.

La génération à effet postérieur est celle utilisée dans l'architecture du NBS et est représentée à la figure 5.2(c).



GE : générateur d'exceptions; IR : implémentation de référence; T :testeur

Figure 5.2 : configurations possibles entre le générateur d'exceptions et l'implémentation de référence.

remarque 1 :

Le générateur à effet postérieur peut également être utilisé pour fournir les services de type 5 et 3 (décodage d'erreurs). Pour les services de type 5, étant donné que le générateur est situé sur le chemin des éléments de protocole en provenance de l'implémentation que l'on teste, il suffirait d'y introduire une table donnant pour chacune des erreurs qui peuvent survenir dans les éléments de protocole, l'action à suivre : par exemple, ignorer le message ou le passer tel quel à l'implémentation de référence. Le fait d'ignorer le message reçu permet d'éviter que le test soit avorté. (En effet, si le message erroné était transmis à l'implémentation de référence, celle-ci devrait interrompre le test, étant donné qu'elle ne reconnaît que des éléments de protocole valides). Pour les services de type 3, l'état courant de l'implémentation de référence est communiqué en permanence au générateur d'exceptions. Lorsqu'un élément de protocole valide est reçu, le générateur d'exceptions vérifie si un tel message peut être reçu par l'implémentation de référence et, selon les directives transmises, l'ignorera ou le transmettra malgré tout à l'implémentation de référence.

Aucune de ces extensions n'a cependant été adoptée par le NBS pour son générateur d'exceptions. Peut-être entraîneraient-elles une complexification du générateur qui ne serait pas compensée par l'élargissement des possibilités offertes.

Les avantages de la technique de l'implémentation de référence ne sont pas négligeables :

1) Comme déjà mentionné au point 3.3.2, l'implémentation de référence peut être obtenue de manière quasi-automatique, à partir des spécifications formelles, et ce, pour chacune des couches de l'architecture.

2) Il est relativement aisé d'employer le même générateur d'exceptions pour les différentes couches. Il suffit pour cela :

a. de modifier dans le générateur les tables donnant, pour la couche testée, les différents types d'éléments de protocole possibles, et les champs qui les constituent ;
b. de calculer l' "offset" qui permettra au générateur d'exceptions de travailler sur les éléments de protocole d'une couche particulière, et ce, à partir des éléments de la couche transport transmis (ou reçus) sur l'interface transport/réseau. En effet, les éléments de protocole des couches supérieures sont acheminés en tant que données dans les éléments de protocole des couches inférieures.

Ces deux avantages assurent la polyvalence de cette architecture. Par contre, au passif de cette technique :

1) Il faut souligner tout d'abord le fait qu'elle ne permet pas de simuler des événements inhabituels pouvant survenir à l'interface inférieure de l'implémentation que l'on teste (c'est-à-dire le type de service 6).

2) De plus, en attendant une normalisation plus avancée des protocoles, certaines options non encore fixées doivent être déterminées lors de l'implémentation. Ainsi, la réalisation de l'implémentation de référence a supposé un choix (parfois arbitraire) de ces options par le réalisateur.

Par conséquent, le comportement de l'implémentation de référence peut, lors du test de conformité, se différencier de celui de l'implémentation que l'on teste, en raison d'un choix d'options différents pour chacune des implémentations. Il est donc possible que des implémentations différentes d'un même protocole ne puissent collaborer, même si toutes les deux sont testées.

remarque 2 :
Le générateur d'exceptions tel qu'il est défini par le GMD est plus puissant que celui utilisé par le NBS en ce sens qu'il offre en plus les services de type 4. Les commandes de génération d'éléments de protocole sont communiquées au générateur d'exceptions qui, après avoir construit les éléments de protocole de niveau-(N), les transmet au moyen des primitives de transfert de données de la couche réseau. Malgré les avantages offerts, la génération d'éléments de protocole entraîne de nombreuses modifications lorsqu'on désire tester l'implémentation d'une couche supérieure dans l'architecture. En effet, il faut non seulement communiquer la structure des éléments de protocole susceptibles d'être générés (voir le deuxième avantage ci-dessus), mais en plus, il faut prévoir la construction et l'adjonction à ces éléments de protocole des "headers" des couches inférieures.

remarque 3 :
On peut envisager de simplifier l'introduction du générateur d'exceptions dans une architecture en supprimant la connexion inter processus qui existe entre le testeur et le générateur. Il suffit de placer dans un fichier les commandes à exécuter par le générateur au cours du test. Le nom de ce fichier de commandes sera communiqué au générateur au début du test, de la même façon que le scénario de test choisi est transmis au testeur. (Les commandes de génération d'exceptions figurant dans le fichier sont celles qui devraient normalement figurer dans le scénario "complet", lorsqu'il existe une liaison entre le testeur et le générateur d'exceptions). Cette amélioration n'a pas été mise à profit dans les architectures de test étudiées car elle présente malgré tout un inconvénient majeur : la synchronisation du test doit non seulement être réalisée entre les deux processus que constituent les interpréteurs de scénarios (du centre de test et du site client) mais en plus elle doit tenir compte d'un troisième processus, le générateur d'exceptions qui est devenu à présent indépendant.

(2) Encodeur/décodeur

Une solution alternative à l'implémentation de référence et au générateur d'exceptions est de leur substituer un module d'encodage et de décodage des éléments de protocole. En supposant que l'on ait à tester une implémentation de niveau-(N), le service fourni au testeur (niveau-(N+1)) par l'encodeur/décodeur de niveau-(N) est en réalité un service-(N) "amélioré" car, en plus du service qu'offrirait toute implémentation de référence-(N) (service de type 1 ci-dessus),

- il réalise des demandes de service pour la génération d'erreurs de protocole (types 2 et 4),
- il indique au testeur les éventuelles erreurs de protocole constatées (types 3 et 5) ; dans la technique précédente, de tels événements auraient été rejetés par l'implémentation de référence qui entamerait la procédure de rétablissement prévue par le protocole en cas d'erreur,
- il permet de faire apparaître des erreurs de protocole dans le niveau-(N-1) en sollicitant les primitives appropriées de l'interface supérieure de la couche-(N-1) (par exemple, si l'on teste une implémentation d'un protocole de transport, un RESET.request à l'interface supérieure de la couche réseau pourra être généré).

Cette technique a été adoptée par le NPL. Les principaux avantages de celle-ci par rapport à la technique précédente sont les suivants :

1) Dans la technique de l'encodeur/décodeur, il n'existe qu'un seul type d'échange de messages entre le conducteur de test et le module d'encodage/décodage, contrairement aux deux types d'échanges vus précédemment entre le conducteur de test et l'implémentation de référence d'une part et le générateur d'exceptions d'autre part. L'interface supérieure de l'encodeur/décodeur présente le même aspect que celle d'une implémentation de référence, à quelques différences près, à savoir :

a. chaque primitive se voit associée une liste de paramètres, dont certains sont supplémentaires par rapport aux primitives traditionnelles, qui donnent certaines directives à l'encodeur/décodeur pour la construction (primitives *requête* et *réponse*) et la réception (primitives *confirmation* et *indication*, des éléments de protocole.

b. il existe des primitives supplémentaires pour la génération de certains types d'erreurs ou pour indiquer leur survenance. La liste des paramètres associée à ces primitives permet de soumettre l'implémentation que l'on teste à toutes les erreurs possibles, et ce, de manière contrôlée.

Le comportement de l'encodeur/décodeur dépend seulement de la disponibilité et de l'état du service-(N-1). Ainsi, l'encodeur/décodeur ne peut donner de suite favorable à une demande de génération et de transmission d'éléments de protocole sans qu'une connexion (N-1) n'existe déjà.

2) L'avantage non négligeable de cette technique est sa facilité d'adaptation aux choix des options prises dans l'implémentation du protocole que l'on teste (contrôle de flux, segmentation, ...). En effet, en modifiant le mode de construction des éléments de protocole, on peut facilement tenir compte de ces particularités alors que l'utilisation d'une implémentation de référence impose un choix préalable de ces options.

Les désavantages de la technique de l'encodeur/décodeur par rapport à celle de l'implémentation de référence sont :

1) Son coût de développement élevé dû à sa spécificité ; en effet pour chacune des couches du modèle ISO/OSI que l'on désire tester, il est nécessaire de concevoir un nouvel encodeur/décodeur car la construction et la reconnaissance des éléments de protocole, ainsi que les primitives d'interface avec l'encodeur/décodeur sont différentes pour chacune de ces couches.

2) La complexité de l'interface supérieure de l'encodeur/décodeur en raison de l'étendue et de la flexibilité des manipulations offertes par le module.

(3) Générateur d'éléments de protocole

La technique de génération d'éléments de protocole (appelée aussi technique de substitution) est basée sur la construction directe des éléments de protocole-(N) par un module indépendant : le **générateur**. Cette technique a été adoptée par Bull et ADI.

En règle générale, le rôle de tout générateur de niveau-(N) est :

- d'établir et de gérer des connexions-(N-1) au moyen des primitives-(N-1) ;
- de construire des éléments de protocole-(N) (service de type 1 décrit ci-dessus) et de les transférer au moyen des primitives de service-(N-1) ;
- de recevoir des éléments de protocole-(N) par l'intermédiaire des primitives de service-(N-1) ;
- d'utiliser les timers.

En plus de ces fonctions de base, le générateur peut réaliser les services de type 2, 3, 4 et 5 cités plus haut. Le générateur joue également le rôle du conducteur de test et son fonctionnement est guidé :

- manuellement, par une interface utilisateur (cas pour l'ADI/Genepi) ;
- automatiquement, par la lecture d'un fichier scénario contenant les commandes de génération (cas pour Bull) ;
- au moyen d'une table d'états (automate) (cas pour l'ADI/Genepi_A).

L'avantage principal du générateur est sa capacité d'adaptation aux caractéristiques du protocole (voir avantage 2 de l'encodeur/décodeur) tandis que son défaut majeur est son coût élevé dû à sa spécificité (voir désavantage 1 de l'encodeur/décodeur).

5.4. Proposition d'architecture

5.4.1. Introduction

Après avoir analysé les points forts et les points faibles de différentes architectures, nous nous proposons maintenant de décrire une nouvelle architecture dont les caractéristiques répondront au mieux aux critères définis dans les prérequis d'une bonne architecture. Comme le montre le tableau comparatif de la figure 5.1, c'est surtout au niveau de la modularité que la critique est la plus vive.

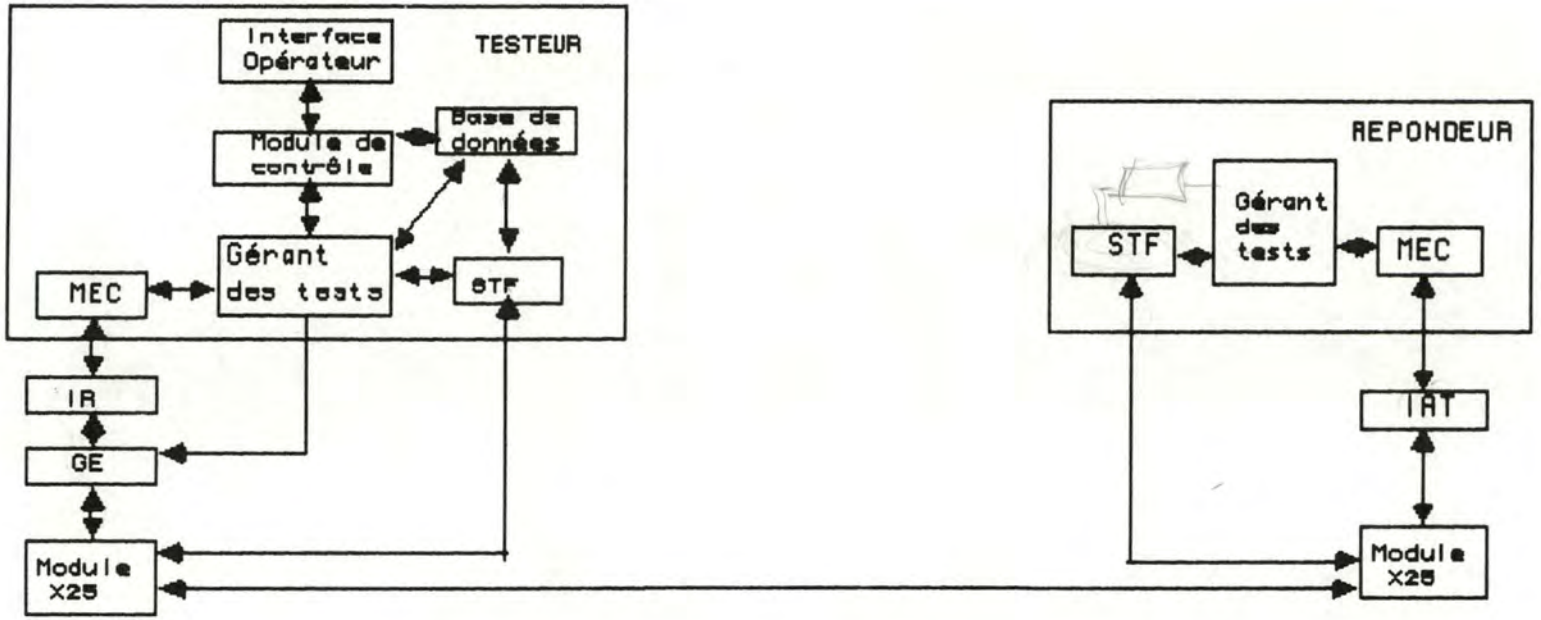
En effet, bien que les architectures proposées présentent un certain degré de modularité, certains de leurs composants doivent être revus lorsque l'on envisage le test d'une couche autre que celle pour laquelle le système a été conçu. C'est notamment le cas pour les deux éléments de base de ces architectures à savoir le testeur et le répondeur. En ce qui concerne le testeur, on peut considérer qu'il s'agit d'un demi-mal étant donné que celui-ci reste valable pour tester des implémentations de différents clients. Inversément, le répondeur, une fois les tests d'une implémentation terminés, devient inutile. Si ce n'est l'expérience qu'il a acquise, le client perd donc l'investissement consenti.

On voit donc qu'un redécoupage de ces deux composants s'avère essentiel. Ce découpage aura pour but de restreindre les dépendances vis-à-vis de l'implémentation à quelques modules spécifiques du testeur ou du répondeur. C'est donc vers un tel objectif que s'oriente la proposition d'architecture qui suit.

5.4.2. Graphe de l'architecture

Hypothèse

Basée sur la comparaison de la section précédente, l'architecture proposée ici évite d'imposer des contraintes sur le site client. Elle nécessite cependant de faibles possibilités de stockage au niveau du site client, à savoir la possibilité de mémoriser un scénario et un fichier de résultat simultanément.



GE : générateur d'exeptions
MEC : module d'établissement de correspondance
IAT : implémentation à tester
STF : système de transfert de fichiers

Figure 5.3. : proposition d'architecture (illustrée pour les tests d'un protocole de niveau 4).

Architecture

La figure 5.3 illustre les différents composants de l'architecture proposée ainsi que les relations qui les unissent.

5.4.3. Analyse des composants de l'architecture

Composants du centre de test

1. Testeur

Sur base des fonctions assignées au testeur dans les architectures, il est possible de subdiviser ce composant en plusieurs modules fonctionnels, l'implémentation de ceux-ci pouvant être réalisée indépendamment. La figure 5.3 contient entre autres une modélisation du testeur où les composants suivants peuvent être identifiés :

* la base de données : ce module comprend un ensemble de tests sous forme de scénarios susceptibles d'être utilisés pour tester une couche donnée. L'utilisation d'un testeur guidé par scénario a été choisi pour sa simplicité ainsi que pour sa facilité d'utilisation et de compréhension. Par rapport à l'architecture NBS développant la même approche, les scénarios offriront en plus :

  - la possibilité d'utiliser des commandes de génération d'éléments de protocole destinées au générateur d'exceptions ;
  - la possibilité d'exprimer des alternatives dans la réception des messages.

* l'interface opérateur : ce module prend en charge la totalité du dialogue entre l'opérateur et le système de test. C'est un module indépendant du niveau testé.

* le module de contrôle : ce module a la responsabilité des décisions au sujet de la planification des tests, du contrôle de leur progression et de l'analyse des résultats. Pour réaliser ces fonctions, il doit interagir avec la base de données, l'interface opérateur et le gérant des tests. Il s'agit d'un module indépendant du niveau testé.

* le gérant des tests : ce module, recevant les références d'une paire de scénarios (1 scénario à exécuter sur le centre de test et 1 scénario complémentaire à exécuter sur le site client) du module de contrôle, envoie les informations nécessaires à la réalisation des tests au répondeur, via le STF décrit ci-après, ainsi qu'au module d'établissement de correspondance du testeur. Il s'agit d'un module indépendant de la couche testée.

* le module d'établissement de correspondance : ce module établit une correspondance entre les séquences de test reçues du gérant des tests en termes de primitives de service abstraites en primitives d'interface pour l'implémentation de niveau-(N) et vice-versa. La spécification de ce module dépend de la couche où se situe l'IAT.

* le système de transfert de fichiers (STF) : ce module, implémenté sur le niveau 3 de X25, permet de transmettre vers le site client les fichiers scénarios et inversément de recevoir du site client les fichiers résultats. Présentant la même utilité qu'un protocole de test (de type NPL), un système de transfert de fichiers offre en plus les avantages suivants :

- un tel module est utile à d'autres fins que la réalisation des tests. L'investissement consenti est donc plus facilement rentable. De plus la firme cliente peut déjà disposer d'un tel module ;
- ce module s'appuie sur une implémentation du protocole X25 et son utilisation est totalement indépendante du comportement de l'IAT; il ne requiert aucun test initial nécessaire à l'établissement de certaines propriétés de base de l'IAT (pour rappel, l'utilisation d'un protocole de test comme pour le NPL supposait que les tests d'ouverture de connexion et de réception de données n'avaient révélé aucune erreur) ;
- ce module est indépendant de la couche testée ;
- il existe des spécifications de protocole de transfert de fichiers émis par des organismes internationaux de standardisation qui peuvent servir de base à la réalisation d'un tel module.

2. Implémentation de référence (IR)

L'utilisation d'une implémentation de référence a été choisie en raison de la simplicité d'obtention de celle-ci (grâce au mécanisme de génération automatique, cité en 3.3.2) et pour sa fiabilité. Ce mécanisme permet de plus d'obtenir rapidement une nouvelle implémentation de référence lorsque le choix des options a été modifié.

3. Générateur d'exceptions

Le générateur d'exceptions a pour objectif de faire apparaître des erreurs au niveau du protocole-(N) mais en plus, il peut être utilisé pour générer des événements inattendus à l'interface inférieure de l'implémentation que l'on teste. Les commandes qui lui sont transmises par le gérant des tests sont donc de deux types :

- Les premières permettent la manipulation des éléments de protocole (ex. modification de certains champs des éléments de protocole, suppression ou retard dans la transmission de ceux-ci, etc ...) en provenance ou à destination de l'implémentation de référence;

- Les secondes concernent la génération de certaines requêtes "inhabituelles" au niveau de l'interface supérieure de l'implémentation-(N-1). Le générateur court-circuite ainsi l'implémentation de référence-(N). En effet, celle-ci n'aurait pu produire ces requêtes étant donné que, par définition, son comportement est correct. De par sa position entre la couche-(N) et la couche-(N-1) (et non plus systématiquement entre les couches 3 et 4, comme c'était le cas dans les architectures étudiées), ce générateur fournit ce service quelle que soit la couche testée. Par exemple, si l'on teste la couche session, le générateur d'exceptions est positionné entre les implémentations de référence de la couche session et de la couche transport, et permet de faire apparaître (à tout moment du test), à l'interface inférieure de l'implémentation à tester-(5) des primitives T_DISCONNECT.indication inattendues par celle-ci (et ce, grâce à la production par le générateur du centre de test de primitives T_DISCONNECT.request à l'interface supérieure de l'implémentation de la couche transport).

4. Module X25

Ce module constitue une implémentation vérifiée du protocole X25 (niveaux 1-2-3) du CCITT. Il permettra d'accéder au réseau de transmission de données.

Composants du site client

1. Répondeur

Bien que la fonction du répondeur s'apparente à celle du testeur du centre de test, sa structure interne est plus simple étant donné qu'il s'agit d'un composant passif de l'architecture. Par rapport au testeur, seuls les composants suivants sont nécessaires :

- module d'établissement de correspondance : ce module a la même fonction que son homologue du centre de test ;
- le gérant des tests : ce module remplit un sous-ensemble des fonctions remplies par le gérant du centre de test. Ainsi, il n'a aucune commande à transmettre à un générateur d'exceptions, ne reçoit de commande que via le système de transfert de fichiers et ne gère qu'un seul fichier par test ;
- le système de transfert de fichiers : communiquant avec son homologue du centre de test, ce module assure l'échange de fichiers (paramétrage du répondeur et transfert de résultats). Il permet ainsi de lever une contrainte quant aux capacités de stockage d'informations sur le site client.

2. Implémentation à tester (IAT)

L'IAT est une implémentation d'un protocole des niveaux 4 à 7 pour laquelle on désire établir la conformité par rapport à ses spécifications.

3. Module X25

Ce module est analogue à celui utilisé dans le centre de test.

5.4.4. Méthode de réalisation des tests

Pour terminer et afin de bien fixer les interactions possibles entre les différents modules, un exemple de déroulement de test sur une telle architecture peut être donné. Une session de test est typiquement composée de deux parties : la planification des tests à mener et leur exécution.

Durant la phase de planification, le module de contrôle fournit à l'opérateur un menu de test parmi lequel il peut choisir la classe de tests qu'il désire réaliser (par exemple : tests sur la phase d'établissement d'une connexion, tests sur la phase de transfert de données normales, tests sur la phase de transfert de données express, tests sur la phase de libération d'une connexion, tests combinant plusieurs tests précédents). Durant la sélection des tests, diverses interactions sont possibles entre l'opérateur et le module de contrôle pour définir les paramètres (les adresses notamment) et les valeurs de timers. Le module de contrôle prépare ensuite, en fonction des instructions de l'opérateur et des informations de la base de données, une liste de paires de scénarios à utiliser pour vérifier la conformité de l'IAT vis à vis des critères choisis.

La phase d'exécution des tests peut être subdivisée en 6 étapes :

1 - le module de contrôle parcourt séquentiellement la liste de paires de scénarios qu'il a créées lors de la phase de planification. Il transmet les références des fichiers constituant la première paire de la liste et attend les résultats de l'exécution de ce test avant de poursuivre son analyse ;

2 - recevant les deux références, le gérant du centre de test paramètre le répondeur en transmettant via le module STF le scénario de test (prélevé dans la base de données) relatif au site client. Une fois ce transfert effectué correctement, le test proprement dit peut commencer ;

3 - les gérants du site client et du centre de test analysent les commandes de leur scénario respectif et transmettent vers les modules d'établissement de correspondance les commandes des scénarios (le gérant du centre de test peut également envoyer des commandes au générateur d'exceptions). Les modules d'établissement de correspondance transforment les primitives de service abstraites reçues en primitives réelles afin de fournir à la fois à l'implémentation de référence et à l'implémentation à tester les événements stimuli. Ce processus se poursuit jusqu'à ce que les deux scénarios soient entièrement parcourus ou qu'une erreur soit détectée ;

4 - le gérant du site client transmet via le module STF le fichier résultat qu'il aura créé durant l'exécution des tests ;

5 - le gérant du centre de test transmet au module de contrôle la référence du fichier résultat qu'il a reçu du site client, la référence du fichier de résultat qu'il a lui-même créé ainsi qu'une indication concernant la réalisation du test (Terminé ou Interrompu). Le test est alors terminé ;

6 - recevant ces informations, le module de contrôle les transmet via l'interface à l'opérateur et décide en fonction des résultats déjà obtenus pour les tests précédents de la poursuite (retour à l'étape 1) ou de l'abandon de l'analyse de la liste de paires de scénarios.

Considérant l'architecture venant d'être présentée et les critères d'évaluation définis précédemmment, les conclusions suivantes peuvent être tirées :

* Les contraintes imposées par le répondeur sur le site client sont limitées, seules de très faibles capacités de stockage sont requises. Le protocole de test de certaines architectures est remplacé par un protocole de transfert de fichiers jugé plus simple et réutilisable.

* Le caractère explicatif, le caractère exhaustif et la facilité d'utilisation de l'architecture se situe dans la moyenne comparativement aux autres architectures.

* L'implémentation de l'architecture est facilitée d'une part, par le fait que de nombreux composants sont récupérables et d'autre part, par le fait que le site client peut, dans certains cas, disposé déjà d'un protocole de transfert de fichiers implémenté.

* La modularité du testeur et du répondeur a été améliorée par rapport aux autres architectures et ce via une décomposition fonctionnelle.

De telles modifications permettent d'obtenir une architecture pour laquelle une note "+" sera au moins attribuée pour chacun des critères utilisés (note "++" pour la facilité d'implémentation et la modularité).

6.TESTS DES PROTOCOLES DE BAS NIVEAU

6.1. Introduction

Tandis que les chapitres précédents étaient axés essentiellement sur les tests d'implémentation de protocole de haut niveau (4 et supérieurs), l'objectif de ce chapitre sera d'étudier les techniques pouvant être mises en oeuvre pour vérifier la conformité des protocoles de bas niveau (3 et 2). Etant donné que la première couche concerne principalement les caractéristiques physiques et électriques de la ligne de transmission , les tests de cette couche consistent à vérifier tous les signaux échangés entre le DTE et un modem. Ces tests sortent du cadre de cette étude et ne seront pas abordés dans ce chapitre.

Tout d'abord, on peut signaler que toute la démarche décrite pour les tests centralisés tant dans l'approche black box que white box reste applicable pour les protocoles de bas niveau ; la proposition d'architecture de la section 4.2.3. prenant comme exemple d'entité à tester une implémentation de la couche 3.

Quant aux architectures étudiées dans le cadre de l'approche décentralisée, elles étaient spécifiques aux protocoles de haut niveau et donc non applicables aux protocoles de bas niveau. En effet, alors que les protocoles de haut niveau sont de bout en bout (l'information de contrôle n'a de signification que pour les entités paires communiquant entre elles), l'information de contrôle utilisée par les protocoles de bas niveau a une signification pour les différents intermédiaires du réseau et est traité (éventuellement modifiée par ceux-ci lors de leur acheminement)

Entre autres, la transmission d'un élément de protocole (paquet ou trame) invalide du centre de test vers l'implémentation à tester s'avère impossible puisque l'erreur sera détectée dès le noeud d'entrée du réseau et ne parviendra donc jamais à l'IAT.

On constate donc que si l'on désire tester une implémentation d'un protocole de bas niveau dans un environnement réel, le développement d'architectures spécifiques devient nécessaire.

L'objectif des deux sections qui suivent sera dès lors d'une part, d'analyser comment l'architecture proposée par le NPL pour les protocoles de haut niveau peut être modifiée pour tenir compte des spécificités des protocoles de bas niveau et d'autre part, d'étudier une architecture propre au test d'un tel protocole.

6.2. Adaptation de l'architecture du NPL

La transmission de paquets (niveau 3) ou de trames (niveau 2) invalides tant pour leur contenu que pour leur séquence étant impossible dans l'architecture proposée à la figure 4.7 (page 51) (les éléments de protocole étant bloqués au niveau du noeud d'entrée dans le réseau), le NPL a développé parallèlement à cette architecture une extension pouvant être utilisée pour vérifier la conformité de ces protocoles. Le principe de la solution apportée consiste à introduire un nouveau composant : le PTU (Portable Testing Unit). Celui-ci s'insère entre le réseau et le site client comme le montre l'architecture physique illustrée à la figure 6.1.

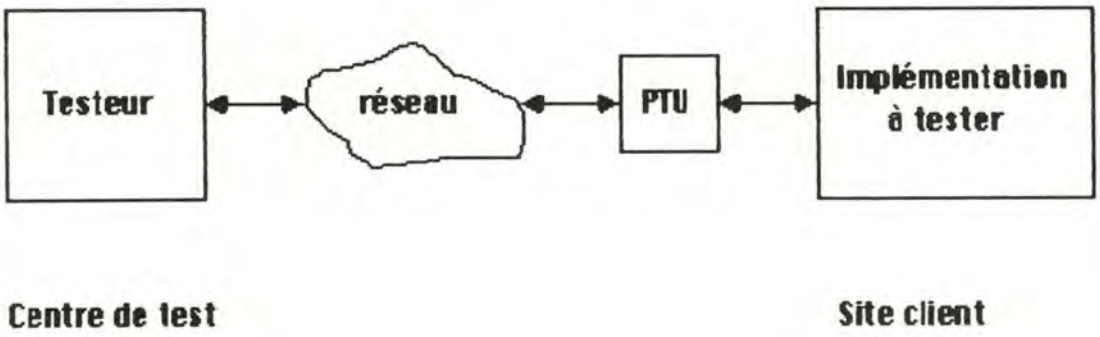


Figure 6.1 : architecture physique pour le test des protocoles de bas niveau.

Les différents composants logiciels identifiés dans l'architecture logique de la figure 4.7 sont désormais répartis entre le centre de test et le PTU.

Le testeur reste entièrement dans le centre de test, l'implémentation de niveau (N-1) améliorée (complétée par un GE) est transférée complètement au niveau du PTU tandis que l'encodeur/décodeur de niveau-(N) (N = niveau à tester) est réparti sur ces deux sites comme le montre la figure 6.2. Un protocole non standard (propre à l'architecture de test) entre le centre de test et le PTU est également introduit. Celui-ci permettra le transfert de commandes de manipulation et de génération d'éléments de protocole. Des éléments de protocole valides pourront être transmis à partir de l'encodeur/décodeur. De même, des éléments de protocole (reset, clear pour le niveau 3) pourront être générés directement par le PTU lorsqu'il en recevra "l'ordre" du centre de test. Inversément, les éléments de protocole en provenance de l'IAT seront être analysés par l'encodeur/décodeur. Ainsi, un ensemble d'événements tant normaux qu'anormaux pourront être soumis à l'IAT afin d'évaluer sa conformité au standard. En plus de ces fonctions de protocole, le PTU peut analyser l'activité de l'interface X25 du client et établir les performances (notamment par la mesure du "throughput") de l'IAT.

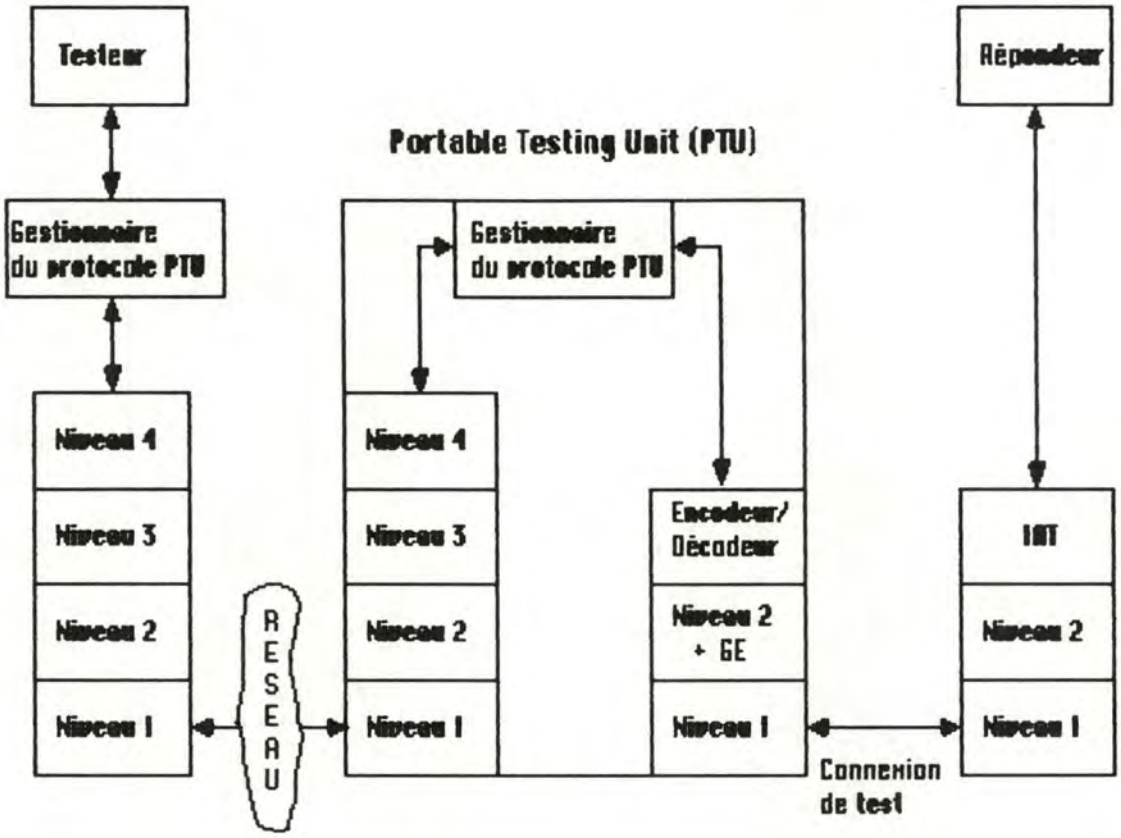


Figure 6.2 : adaptation de l'architecture du NPL pour le test d'un protocole de bas niveau (niveau 3).

Remarque

Si l'on reprend les fonctions affectées à l'outil **Cerbère** proposé par l'ADI, on se rend compte qu'elles correspondent à celles du PTU proposé par le NPL. La différence entre ces deux outils vient du fait que Cerbère est un outil autonome tandis que le PTU est paramétré à partir du centre de test.

6.3. Une architecture de test pour les protocoles de bas niveau

6.3.1. Introduction

Le protocole X25 gérant l'échange d'informations entre un équipement de terminaison de circuit de données (ETCD) et un équipement terminal de traitement des données (ETTD), il semble à priori inutile d'utiliser un réseau de transmission de données pour tester ces couches en environnement réel. En effet, une liaison directe est suffisante et simplifie fortement l'architecture.

6.3.2. Graphe de l'architecture

Comme l'illustre la figure 6.3, une liaison téléphonique et deux modems permettent de réaliser les tests de conformité.

6.3.3. Analyse des composants de l'architecture

Les composants principaux de cette architecture sont les suivants :

1. **Testeur**

Suivant les commandes des fichiers de test, le testeur transmet au simulateur/générateur les ordres adéquats pour la transmission des éléments de protocole. Il gère également la mise à jour des fichiers journaux.

2. **Simulateur/générateur**

Ce composant sera tel :

* qu'il offrira un langage de commandes
- permettant de spécifier de manière mnémonique la transmission de tous les types de paquets/trames ;
- donnant la possibilité de spécifier les champs de ces éléments de protocole

* que chaque niveau puisse opérer soit en mode automatique soit en mode manuel. En mode automatique, le niveau adhère au protocole X25 et répond aux éléments de protocole lui parvenant de l'IAT (simulateur). En mode manuel, le niveau transmet des éléments de protocole uniquement sur base de commandes qu'il reçoit du testeur (générateur) ; le simulateur/générateur travaille en mode manuel pour le niveau à tester et en mode automatique pour les autres.

* qu'il transmettra tous les paquets/trames reçu(e)s de l'IAT vers le testeur.

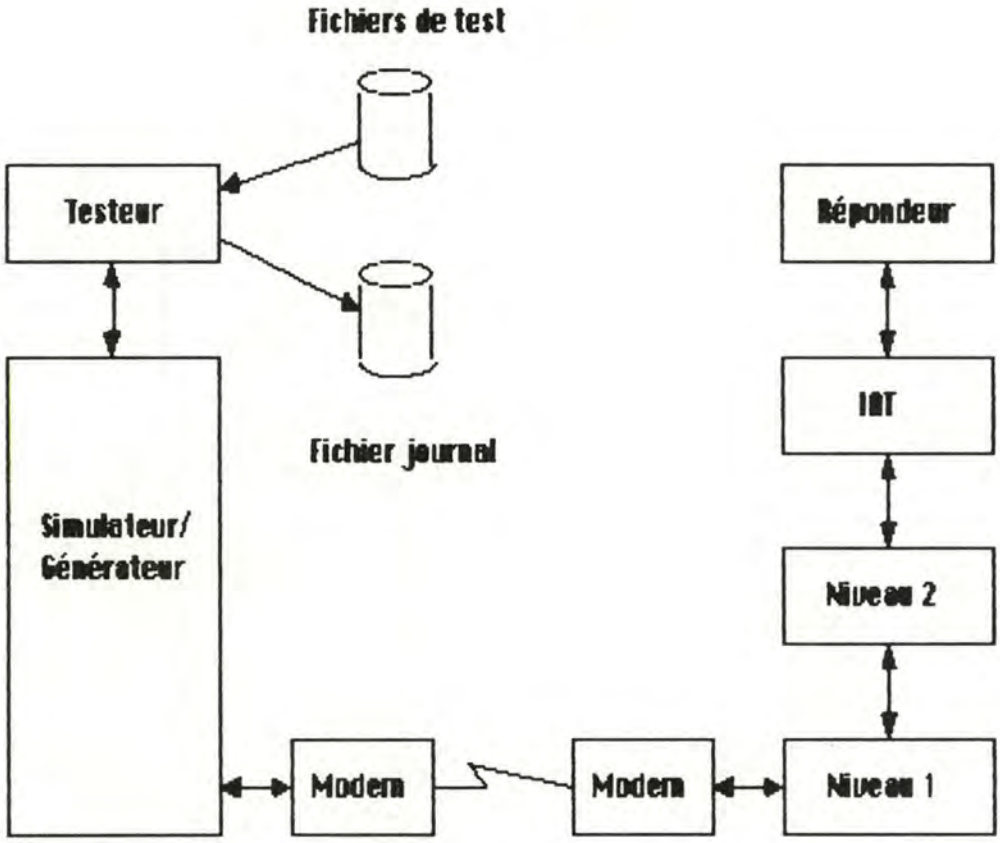


Figure 6.3 : architecture logique pour les tests de protocoles de bas niveau (niveau 3).

A titre documentaire, on peut signaler que des outils réalisant les fonctions du simulateur/générateur sont disponibles sur le marché, notamment : PRO/TESTER distribué par Applied Data Communications, DYNAPAC NET/18 HS distribué par Telindus et CHAMELEON distribué par Tekelec.

3. Répondeur

Agissant en tant que couche utilisatrice de l'IAT, le répondeur est paramétré à partir du testeur par la transmission de commandes précisant son comportement (répondeur de type NPL). Ainsi, le comportement du répondeur pour les implémentations à tester des couches 2 ou 3 est déterminé par le contenu des trames d'information (couche 2) ou des paquets de données (couche 3) reçus.

Les fichiers suivants sont également utilisés :

Fichiers de test

Ces fichiers résultent d'une phase préparatoire au test proprement dit : la phase de construction des tests. Sur base de fichiers de test paramétrés et d'un fichier des paramètres, un préprocesseur opére la substitution des paramètres par leur valeur réelle. Ce preprocessing assure l'adaptabilité des fichiers de test à différentes configurations de test.

L'ensemble de tests cherchera à mettre à l'épreuve complètement l'IAT dans des conditions tant normales qu'anormales de fonctionnement. En particulier, chaque test :

* spécifiera la transmission des paquets/trames. Le type de paquet/trame sera mentionné de manière mnémonique et chaque champs de l'élément de protocole pourra être spécifié ; ces informations seront transmises une à une par le testeur au simulateur/générateur.
* spécifiera la réponse attendue de l'IAT lors de la réception par celle-ci d'un paquet/trame donné.

Fichiers journaux

Ces fichiers fournissent après chaque test des indications concernant le degré de conformité de l'implémentation testée.

6.3.4. Méthode de réalisation des tests

Cette section décrit la façon de procédé pour vérifier la conformité d'une implémentation de niveau 3 ou 2. Bien que le contenu des fichiers de test soient évidemment différents pour les deux pouvant être testées, la méthode à suivre, à savoir :

1. détermination de la couche à tester ;
2. détermination des paramètres spécifiques à l'implémentation à tester et formation du fichier des paramètres ;
3. preprocessing des fichiers de test paramètrés ;
4. installation du répondeur comme couche utilisatrice de l'IAT ;
5. établissement de la communication avec modems ;
6. vérification de conformité (grâce aux fichiers de test) ;
7. transmission des fichiers journaux au réalisateur de l'IAT.

L'étape 6 de vérification est entièrement guidée par le testeur. Les fichiers de test qu'il utilise sont composés de paires de commandes : 1 commande d'émission d'un(e) paquet/trame et la 1 réponse attendue de l'IAT ou vice-versa. Considérant la commande d'émission, le testeur la transmet au simulateur/générateur et attend la réponse de l'IAT. Quant à l'éventuelle réponse reçue par l'IAT, si elle est différente de celle spécifiée dans le fichier de test ou si aucune réponse n'est reçue endéans un certain temps, le test est clôturé. Dans le cas contraire, le test est poursuivi. Le procédé est utilisé pour une série de tests relatifs au niveau testé et un rapport final est produit. Celui-ci comprendra :

* l'indication des tests passés ou échoués (avec mention de la raison dans ce cas) ;
* le détail de tous les paquets/trames échangé(e)s entre le testeur et l'IAT ;
* un résumé du nombre total de tests passés et échoués.

7. LES TESTS

7.1. Introduction

Rappelons que les tests qui nous concernent ont pour objectif de vérifier si une implémentation est conforme à un protocole, c'est-à-dire :

- qu'elle répond correctement à des événements valides en provenance de l'interface supérieure et de l'entité paire ;
- qu'elle rejette les erreurs transmises sur cette interface et les erreurs de l'entité paire ;
- qu'elle manipule correctement les timers.

Une implémentation de protocole est testée comme une "boîte noire" par l'intermédiaire d'un moyen de communication (réseau) qui relie le site client où elle se trouve et le centre de test. Le centre de test contient un testeur actif tandis que le site client contient un répondeur agissant en tant qu'utilisateur des services fournis par l'implémentation à tester (IAT). Pendant le test, on établit la conformité de l'implémentation en observant les messages échangés d'une part entre l'IAT et sa couche utilisatrice et d'autre part entre l'IAT et la couche qu'elle utilise. Il y a donc mise en place d'un échange d'informations entre le testeur et le répondeur via l'IAT. Ces messages échangés sont le résultat de l'exécution de scénarios de test, soit sur le site client et le centre de test (optique du NBS), soit uniquement sur le centre de test (optique du NPL). Ces scénarios sont déduits de séquences de test obtenues manuellement ou automatiquement. Comme il a déjà été dit au point 3.3.4, une séquence de test est un enchaînement d'événements stimuli pour l'implémentation à tester et de réponses correspondantes attendues de cette implémentation.

Une des parties principales dans le test d'une implémentation est la génération automatique de séquences de test.

Etant donné qu'aucune supposition n'est faite en ce qui concerne la structure interne de l'implémentation à tester (approche black box), la génération des séquences de test est basée sur les spécifications du protocole que l'on implémente. Les différentes méthodes de génération automatique décrites dans la suite respecteront cette exigence.

Tout test ainsi généré doit satisfaire les caractéristiques suivantes :

- il a un contenu bien défini, c'est-à-dire qu'il est toujours possible de connaître précisement l'objectif d'un test ;
- il peut être exprimé de manière concise et non ambiguë dans un langage de test dans le but de constituer le scénario correspondant et ce, afin d'atteindre l'objectif du test ;
- son résultat est connu.

7.2. Ordonnancement des tests

Les tests de vérification de conformité d'une implémentation peuvent être subdivisés en quatre classes. Ces différentes catégories de tests sont détaillées dans HENLEY R. F. L. et RAYNER D. (1981 b).

1. Tests des primitives fournies par l'implémentation

Les tests des primitives fournies par l'implémentation ont pour objectif d'établir quelles sont les primitives de services offertes par l'implémentation. Une primitive sera jugée supportée par l'implémentation lorsque le test destiné à la détecter aura réussi au moins une fois, alors qu'elle ne sera jugée comme non fournie qu'après un nombre (fixé arbitrairement au préalable) de tests infructueux.

2. Tests de transitions d'états

Les tests de transitions d'états sont également appelés **tests de conformité**. Ils ont pour objectif de vérifier si l'implémentation que l'on teste réalise, lors de la survenance d'événements bien précis, les transitions d'états et les actions qui y sont associées, et ce, selon les spécifications du protocole. L'implémentation à tester doit au moins être conduite dans chacun des états possibles et répondre à tous les types d'événements possibles pouvant survenir dans cet état.

La génération des séquences de test vérifiant la conformité d'une implémentation peut être obtenue de manière automatique grâce notamment aux méthodes
- de parcours des transitions ;
- de la séquence caractéristique ;
- de la séquence d'identification ;

Ces trois méthodes sont décrites par la suite (voir point 7.3)

3. Tests de variation des paramètres

Les tests de variation des paramètres sont construits afin de vérifier le comportement de l'implémentation à tester face à des modifications de la valeur des paramètres des primitives de service et des éléments de protocole qui lui sont transmis. (Dans le deuxième type de test, ces primitives et les éléments de protocole auront déjà été testés mais avec des valeurs par défaut). Ces variations concernent entre autres les paramètres suivant :

- adresse de destination ;
- adresse d'origine ;
- qualité de service requis ;
- motif de terminaison d'une connexion

et consistent à leur affecter aussi bien des valeurs valides que des valeurs invalides.

4. Tests de combinaison de primitives

Les tests de combinaison de primitives sont réalisés afin de vérifier si la réaction de l'implémentation à un événement particulier dans un état spécifique dépend des événements antérieurs qui ont conduit l'implémentation dans cet état. Ces tests auront principalement pour but de mettre en évidence les effets sur le flux de données des différentes primitives telles que le RESET, EXPEDITED, DISCONNECT, etc ...

Les différentes classes de test définies précédemment seront parcourues dans l'ordre dans lequel elles ont été décrites. Pour chacune d'elles, les séquences de test prévues seront exécutées les unes après les autres.

La conformité de l'implémentation à son standard sera ou non établie en fonction des résultats obtenus.

En plus des tests de vérification de conformité d'une implémentation, il est possible de tester cette implémentation du point de vue de sa robustesse et ses performances.

1. Tests de robustesse

Par robustesse, on entend la capacité de l'implémentation à faire face et à répondre aux erreurs des inputs reçus ainsi qu'aux erreurs internes (générées par elle-même).

Les tests de robustesse sont destinés à déterminer les circonstances qui poussent l'implémentation que l'on teste à produire un message RESET ou DISCONNECT.

2. Tests de performance

La performance d'une implémentation de niveau-(N) dépend des performances des couches sous-jacentes (de niveau-(N-1) et inférieures) et de la charge du réseau employé pour faire transiter les messages entre les implémentations.

La performance d'une implémentation est souvent déterminée par un ensemble de "benchmarks" la confrontant à différentes conditions de fonctionnement qui varient en fonction de :

- la charge du réseau (légère, moyenne, élevée) ;
- du type de lignes utilisées
  - * dans le réseau,
  - * entre le réseau et le centre de test d'une part, et le réseau et le site client d'autre part ;
- de la localisation des implémentations.

L'ensemble des confrontations de l'implémentation à différentes conditions de fonctionnement donne un aperçu de la qualité de service qu'elle offre. Cependant, il est difficile de chiffrer cette qualité de service car aucune unité de mesure n'est appropriée. Néanmoins, on peut tenter de déterminer la qualité du service offert par l'implémentation dans deux domaines particuliers :

a) le transfert de données

on détermine cette valeur en calculant la durée qui sépare le moment où les données sont communiquées à l'implémentation que l'on teste, par l'utilisateur de celle-ci, et le moment où elles sont reçues par l'utilisateur de l'entité paire ;

b) l'établissement d'une connexion

par la durée nécessaire à l'établissement d'une connexion.

Etant donné que les mesures de performance sortent du cadre de notre étude, il n'en sera plus question dans la suite. Néanmoins, le lecteur désirant un complément d'informations à ce sujet pourra consulter Nightingale (1982) et Mc Coy, Colella et Wallace (1981).

7.3. Outils d'aide à la génération automatique des tests

7.3.1. Méthodes basées sur une modélisation FSM

Lorsque l'on dispose d'une architecture de test telle que celles décrites dans les chapitres 4 et 6, se pose alors le problème du choix des tests à mener sur cette architecture pour vérifier la conformité d'une implémentation de protocole. La section 7.2 classifie les différents tests à réaliser pour atteindre cet objectif : tests des primitives fournies, tests de transitions d'états, tests de variation de paramètres, tests de combinaison de primitives. Reste donc à envisager la génération de séquences de test pour chacune de ces classes. Cependant, seuls les tests de transitions d'état sont envisagés ici.

Comme déjà signalé au niveau de l'analyse des outils de description et validation formelle (point 3.3.1.), le modèle d'automate à états finis (FSM Finite State Machine) est particulièrement bien adapté pour décrire le comportement d'un système de communication. Rappelons que dans ce modèle, un système est composé d'un nombre fini d'états reliés par des transitions. La survenance d'un événement (input) provoque le passage du système de l'état courant à un nouvel état via une transition consistant en l'exécution de traitements et en la génération d'outputs. Le formalisme Ei ---X/Y---> Ej employé par la suite signifiera que la survenance de l'événement X dans l'état Ei provoque la génération des outputs Y et le passage à l'état Ej. L'ensemble des états et transitions décrivent un **système**.

L'objectif des sections suivantes sera de décrire trois méthodes de génération de séquences de test figurant parmi les plus répandues à savoir la méthode de parcours des transitions, la méthode de la séquence caractéristique et la méthode de la séquence de vérification. Une analyse de celles-ci en terme d'efficacité dans la détection des erreurs et dans la longueur de la séquence à générer sera également proposée. Les figures 7.1 et 7.2 donnent respectivement une description FSM et une table d'états décrivant le protocole de transport classe 0 tel qu'il est défini par ISO/CCITT (y compris les abréviations utilisées pour les primitives et les éléments de protocole). Ce protocole servira d'illustration pour les trois méthodes exposées ici.

D'autres méthodes de génération automatique de séquences de test sont également applicables. On pourra notamment consulter LINN et McCOY (1983) ainsi que URAL et PROBERT (1983) pour des méthodes basées sur une spécification des tests dans une grammaire générative ("attributed context free grammar and generative grammar").

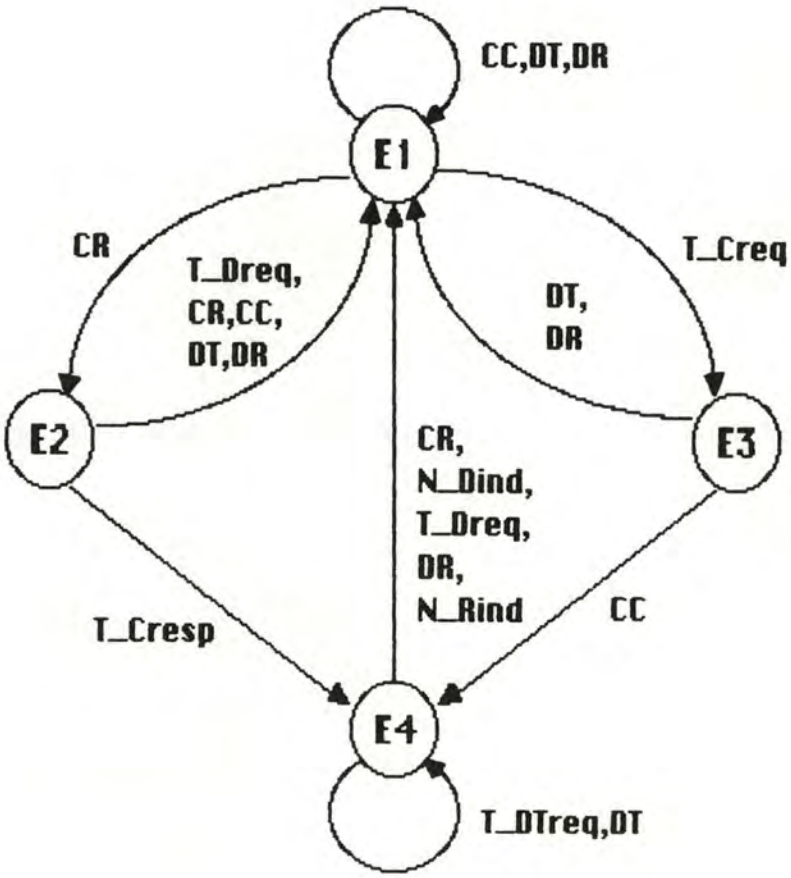


Figure 7.1 : Description du protocole de transport classe 0 sous forme d'automate à états finis.

Nombre d'états n = 4
Nombre de transitions q = 21

Etat / Eu	E1	E2	E3	E4
T_Creq	3/CR	-	-	-
T_Dreq	-	1/DR	-	1/N_Dreq
T_Cresp	-	4/CC	-	-
T_DTreq	-	-	-	4/DT
CR	2/T_Cind	1/ERR	-	1/ERR
CC	1/-	1/ERR	4/T_Cconf	-
DT	1/-	1/ERR	1/-	4/T_DTind
DR	1/-	1/ERR	1/T_Dind, N_Dreq	1/N_Dreq
N_Dind	-	-	-	1/T_Dind
N_Rind	-	-	-	1/T_Dind

Nouvel état/output(s)

Etats

E1 : Idle (état initial)
E2 : Wait_for_Accept_Resp
E3 : Wait_for_TPDU Connect_confirm
E4 : Data Transfert

Evénements en provenance du testeur

CR : Connect_Request PDU
CC : Connect_Confirm PDU
DT : DaTa_request PDU
DR : Disconnect_Request PDU
N_Dind : Network_Disconnect_indication
N_Rind : Network_Reset_indication

Evénements en provenance du répondeur

T_Creq : T_Connect_request
T_Dreq : T_Disconnect_request
T_Cresp : T_Connect_response
T_DTreq : T_DaTa_request

Outputs à destination du testeur

ERR : ERRor PDU
CC : Connect_Confirm PDU
DR : Disconnect_Request PDU
DT : DaTa_request PDU
N_Dreq : Network_Disconnect_request
N_Dind : N_Disconnect_indication
N_Rind : N_Reset_indication

Outputs à destination du répondeur

T_Cind : T_Connect_indication
T_Cconf : T_Connect_confirm
T_DTind : T_DaTa_indication
T_Dind : T_Disconnect_indication

Figure 7.2 : table de transitions pour le protocole de transport classe 0.

7.3.1.1. Méthode de parcours des transitions

Description de la méthode

Une séquence de test est appelée **parcours des transitions** si elle couvre toutes les transitions de la table d'états au moins une fois. Considérant le graphe de description d'un protocole selon la méthode FSM, le problème de la génération d'une telle séquence consiste à déterminer un chemin (séquence d'arcs tel que le noeud terminal d'un arc coïncide avec le noeud initial de l'arc suivant) empruntant au moins une fois chacun des arcs du graphe. L'algorithme A donné en annexe 3 réalise une telle opération.

L'existence d'un tel chemin requiert que le système soit fortement connexe (c'est-à-dire tel que pour tout état E_i et E_j, il existe toujours une séquence d'événements stimuli tel que partant de E_i et appliquant cette séquence, on aboutit à l'état E_j) et totalement spécifié (c'est-à-dire tel que pour tout état et pour tout événement pouvant survenir dans cet état, les outputs et le nouvel état sont invariables et connus).

Application

Appliquant l'algorithme A au graphe 7.1 représentant le protocole de transport classe 0, on obtient la séquence de test suivante :

Les tests

E.P.	Evénements	Outputs générés	E.S	E.P.	Evénements	Outputs générés	E.S.
1	CR	T_Cind	2	1	CR	T_Cind	2
2	T_Dreq	DR	1	2	DR	ERR	1
1	CC	-	1	1	T_Creq	CR	3
1	DT	-	1	3	DR	T_Dind	1
1	DR	-	1			N_Dreq	
1	CR	T_Cind	2	1	T_Creq	CR	3
2	CR	ERR	1	3	CC	T_Cconf	4
1	T_Creq	CR	3	4	CR	ERR	1
3	DT	-	1	1	CR	T_Cind	2
1	CR	T_Cind	2	2	T_Cresp	CC	4
2	T_Cresp	CC	4	4	DR	N_Dreq	1
4	T_DTreq	DT	4	1	CR	T_Cind	2
4	DT	T_DTreq	4	2	T_Cresp	CC	4
4	T_Dreq	N_Dreq	1	4	N_Dind	T_Dind	1
1	CR	T_Cind	2	1	CR	T_Cind	2
2	CC	ERR	1	2	T_Cresp	CC	4
1	CR	T_Cind	2	4	N_Rind	T_Dind	1
2	DT	ERR	1				

E.P. = états présents, E.S. = états suivants

La séquence d'événements générés est reprise dans les colonnes "Evénements". Chaque ligne du tableau représente une transition telle qu'elle est décrite dans la norme. La comparaison des différents outputs repris dans les colonnes "Outputs générés" et des outputs effectivement produits lors de l'utilisation de la séquence permettra d'établir le résultat des tests.

Il est à noter que dans ce cas le système n'est pas totalement spécifié (la réaction du système suite à la survenance d'un événement n'est pas définie pour certains états) mais ceci n'empêche pas l'application de la méthode.

Longueur de la séquence

L'objectif de ce point est de définir une formule exprimant la longueur maximale de la séquence (longueur correspondant au pire des cas pouvant survenir). Cette longueur maximale permettra une comparaison des différentes méthodes analysées qui sera plus générale que la comparaison des longueurs obtenues selon ces diverses méthodes pour l'exemple étudié.

Une limite supérieure pour la longueur de la séquence de parcours des transitions pour un système incomplétement défini est donné par la formule :

$$lg(T) \leq q + (q-1)(n-1) \qquad (1)$$

où q représente le nombre de transitions spécifiées dans le protocole et n le nombre d'états. La longueur de la séquence donnée ci-dessus est de 34 (dont 13 arcs parcourus au moins deux fois) alors que la borne supérieure selon la formule (1) serait 81 (21+20*3).

7.3.1.2. Méthode de la séquence caractéristique

Description de la méthode

Une **séquence caractéristique** est formée de la concaténation (représentée par • dans ce qui suit) des éléments de deux ensembles, ceux de l'ensemble caractéristique (W) et ceux d'un ensemble (P).

L'ensemble caractéristique (W) d'un système défini en tant qu'automate à états finis est une séquence d'un ou plusieurs d'événement(s) permettant de distinguer n'importe quel état vis-à-vis des autres du point de vue de son comportement lors de la réception de cette séquence d'événement(s). Cet ensemble étant souvent appelé W, on parle aussi de la méthode W.

L'ensemble P constitue un ensemble de séquences d'événements telles que pour chaque transition d'un état E_i à un état E_j suite à la survenance de l'événement X, les séquences T et T • X sont reprises dans P ; T représente la séquence d'événement(s) permettant de passer de l'état initial à l'état E_i.

Une méthode de construction de l'ensemble P consiste à créer tout d'abord l'arbre de test du protocole. Chaque chemin partiel de cet arbre constituant un élément de l'ensemble P (Un chemin partiel est une suite d'arcs consécutifs partant de la racine de l'arbre et se terminant en un noeud terminal ou non). Chaque arc de l'arbre se voyant associé un événement, P sera constitué de séquences d'événements (la séquence vide représentée par {} est considérée comme faisant partie de P). L'algorithme B donné en annexe 3 peut être utilisé pour construire un tel arbre.

L'utilisation de cette méthode nécessite que le système soit fortement connexe et qu'il possède un état initial. Sous ces conditions, une séquence caractéristique existe.

Application

Bien que l'existence d'un ensemble caractéristique ne soit pas garantie pour un système incomplètement défini, l'ensemble {DR} en constitue malgré tout un pour le protocole de transport (cfr figure 7.1). En effet, en parcourant les différentes lignes de la table de la figure 7.2., on remarque que l'événement DR a la particularité de provoquer quatre réactions différentes selon l'état du protocole, ce qui n'est le cas pour aucun autre événement. On peut toutefois noter que la séquence {T_Dreq • CC} constitue un autre ensemble caractéristique. Cependant, on choisira de préférence l'ensemble caractéristique comprenant le moins d'éléments afin de minimiser la longueur de la séquence générée.

Quant à l'ensemble P, il peut être construit en ayant recours à l'arbre de test. Se basant sur l'algorithme B, l'arbre de test du protocole de transport est le suivant :

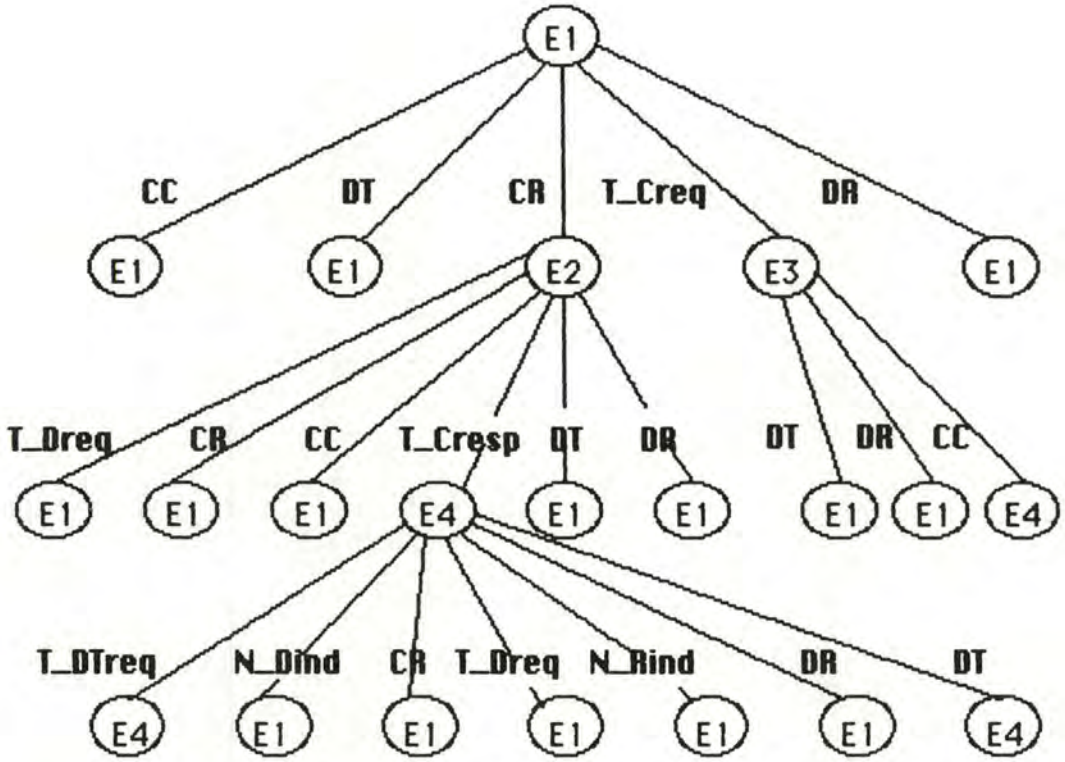


Figure 7.3 : arbre de test du protocole de transport classe 0.

La concaténation de l'ensemble caractéristique (W = {DR}) et des différentes séquences de P (différents chemins partiels de l'arbre de test P = {CC, DT, CR, T_Creq, DR, CR•T_Dreq,...) forment la séquence caractéristique suivante :

P•W = { {} • DR, CC • DR, DT • DR, CR • DR, T_Creq • DR, DR • DR,
CR • T_Dreq • DR, CR • CR • DR, CR • CC • DR,
CR • T_Cresp • DR, CR • DT • DR, CR • DR • DR,
T_Creq • DT • DR, T_Creq • DR • DR, T_Creq • CC • DR,
CR • T_Cresp • T_DTreq•DR, CR • T_Cresp • N_Dind•DR,
CR • T_Cresp • CR•DR, CR • T_Cresp • T_Dreq•DR,
CR • T_Cresp • N_Rind•DR, CR • T_Cresp • DR,
CR • T_Cresp • DT•DR }

Etant donné que chaque sous-séquence de P • W doit être appliquée à partir de l'état initial, le retour à cet état est nécessaire après chaque sous-séquence. Ceci est réalisé par des séquences de transfert appelées réinitialisation.

Dans l'exemple présenté, de telles séquences sont cependant inutiles puisque l'ensemble caractéristique {DR} présente en plus la particularité d'amener le système dans son état inital. Mais ce n'est pas le cas pour tous les ensembles caractéristiques.

Longueur de la séquence

La longueur totale de la séquence définie par la méthode des séquences caractéristiques est donnée par :

{(nombre d'événements de l'ensemble caractéristique W)
* (nombre de séquences dans l'ensemble P)}
+ {(nombre d'événements de l'ensemble P)
* (nombre de séquences dans l'ensemble caractéristique W)}

Une borne supérieure peut être donnée en considérant que l'arbre de test est dégénéré (le nombre d'arcs partant de chaque noeud est inférieur ou égal à 1), ce qui représente le pire des cas :

$$lg(W) \leq (1/2)\, knw(n+1) + m(q+1) \qquad (2)$$

où k représente le nombre maximum de transitions spécifiées pour un état donné de la table d'états du système incomplétement spécifié, n le nombre d'états, m le nombre de séquences dans l'ensemble W, q le nombre total de transitions spécifiées dans la table et w le nombre d'événements de l'ensemble W. La longueur de la séquence donnée ci-dessus est de 66 alors que selon la formule (2) la borne supérieure serait de 92 ((1/2)*7*4*1*5 + 1*22).

7.3.1.3. Méthode de la séquence de vérification

Description de la méthode

Cette méthode est réservée à tout système possédant une **séquence d'identification (DS Distinguish Sequence)**. Une telle séquence peut être définie comme une suite d'un ou plusieurs événement(s) tel que l'output généré par le système lors de la survenance de ce(s) événement(s) est différent selon l'état initial du système.

Une séquence de vérification sera composée de trois parties :

- une sous-séquence initiale permettant de faire passer le système dans un état initial donné. Considérant un graphe du type de celui de la figure 7.1, une telle séquence consistera en un chemin permettant de rejoindre l'état initial à partir de l'état courant (la longueur de la séquence initiale pourra être minimisée en choisissant le chemin le plus court entre ces deux états);

- une sous-séquence de reconnaissance des états qui liste les réponses du système suite à la survenance dans chaque état de(s) l'événement(s) constituant la DS. Pour que chaque état soit reconnu au moins une fois, cette séquence doit être telle que la DS soit appliquée à chaque état au moins une fois.

Si tel est le cas et si le système fonctionne correctement, on observera n (=nombre d'états) réponses différentes, par définition de la DS. On pourra donc associer à chaque état E_i une caractéristique qui l'identifiera à savoir sa réponse à la DS. Si on désire déterminer quel est l'état suivant (E_j) lors de la survenance de l'événement DS dans l'état E_i, on devra cependant appliquer deux fois la DS. En effet, l'état du système étant une variable interne, il est non accessible dans l'approche "black box". Dés lors, l'état E_j ne sera reconnu qu'en appliquant une seconde fois la DS qui l'identifie parmi l'ensemble des états. Cette sous-séquence peut être construite grâce à l'algorithme C donné en annexe 3 appliqué à un (DS)-diagramme qui constitue un diagramme de transition inter-états pour l'événement DS et uniquement cet événement.

- une sous-séquence d'analyse des transitions qui est utilisée pour vérifier individuellement chaque transition de la table d'états. L'étude d'une transition E_i ---X/Y---> E_j consiste à vérifier que la survenance de l'événement X dans l'état E_i provoque :
  - la génération des outputs Y
  - et le passage à l'état E_j tels que cela est décrit dans la norme.

Une telle étude nécessite l'application de la séquence (X • DS) ; l'application de la DS à l'état E_j (état résultant de la survenance de l'événement X dans l'état E_i) permet d'identifier ce nouvel état. Si l'on désire effectuer la même opération pour l'ensemble des transitions décrites pour le protocole (et ainsi définir la sous-séquence d'analyse des transitions), un parcours des transitions doit être effectué.

Ce parcours sera réalisé sur un graphe G déduit de la description FSM du système et construit de la manière suivante : pour tout état Ei du système et pour tout événement X pouvant survenir dans cet état, ajouter à G l'arc représentant la transition Ei --(X • DS)--> Ej. Considérant un tel graphe, l'algorithme A de parcours des transitions donné en annexe 3, réalise le parcours des transitions.

Application

En examinant les différentes lignes de la table 7.2 décrivant le protocole de transport, on constate que la séquence {DR} constitue une DS. Etant donné que l'état 1 (idle) est considéré comme état initial dans lequel peut commencer le test, la séquence initiale est vide. Se basant sur le graphe de la figure 7.4 (ainsi que sur la figure 7.1 donnant les séquences de transfert inter-état) et appliquant l'algorithme C, la séquence de reconnaissance des états suivantes est obtenue :

{ CR • DS • DS • DS • T_Creq • DS • DS • T_Creq • CC • DS • DS } **(S1)**

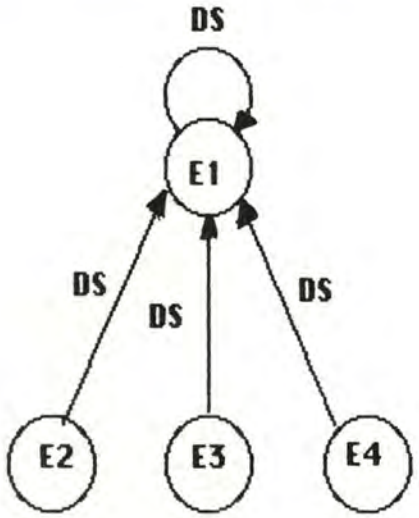


Figure 7.4 : (DS)-diagramme du protocole de transport.

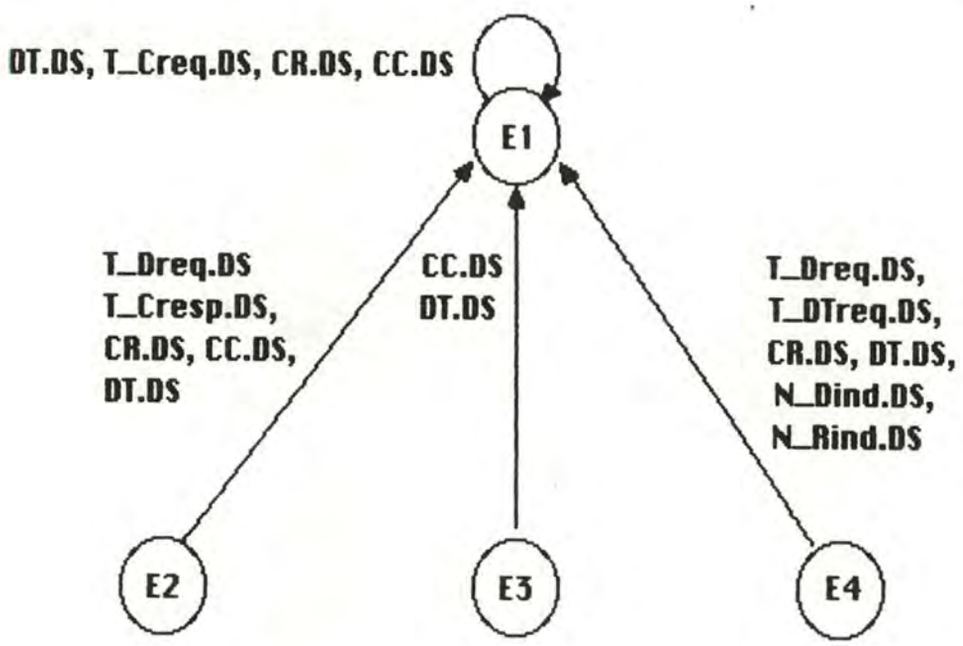


Figure 7.5 : graphe de test pour le protocole de transport.

Quant à la séquence d'analyse des transitions, elle peut être obtenue par application de l'algoritme A au graphe 7.5 :

{ (T_Dreq • DR) • (CC • DR) • CR • (T_Cresp • DR) • (CR • DR) • CR
• (CR • DR) • (T_Creq • DR) • CR • (CC • DR) • (DT • DR) • CR
• (DT • DR) • T_Creq • (CC • DR) • T_Creq • (DT • DR) • CR • T_Cresp
• (T_Dreq • DR) • CR • T_Cresp • (T_DTreq • DR) • T_Creq • CC
• (CR • DR) • T_Creq • CC • (DT • DR) • T_Creq • CC • (N_Dind • DR)
• T_Creq • CC • (N_Rind • DR) }

(S2)

Etant donné que la sous-séquence d'analyse des transitions commence par l'état 2 et que la sous-séquence de reconnaissance des états se termine par l'état 1, une séquence de transfert est nécessaire pour assurer la concaténation de ces deux sous-séquences. La séquence {CR} peut, dans l'exemple être utilisée à cet effet. Dés lors, la séquence de vérification sera : { (S1) • CR • (S2) }.

Longueur de la séquence

GÖNEÇ (1970) donne et commente une expression de la longueur maximale de la séquence de vérification :

séquence de reconnaissance des états : $2\,nL + (n-1)^2$
séquence de vérification des transitions : $q(n-1) + q(L+1)$

$$\lg(V) \leq 2\,nL + (n-1)^2 + q(n+L) \qquad (3)$$

où q représente le nombre de transitions spécifiées dans la table, n le nombre d'états et L la longueur de la DS. La longueur de la séquence donnée ci-dessus est de 64 (dont 19 événements constituant des transferts) alors que selon la formule (3) la borne supérieure serait de 122 (2*4*1 + 3*3 + 21(4+1)).

7.3.2. Comparaisons et évaluations

Type d'erreurs détectées

Afin de pouvoir comparer les différentes méthodes quant à leur efficacité dans la détection des erreurs, on peut distinguer deux types d'erreurs. Considérons deux systèmes A et A' (A' constituant le système de référence supposé correct et A celui qui est évalué) modélisés selon la méthode FSM et possédant les mêmes inputs, on distinguera :

- **les erreurs d'opération** : on dit qu'un système A possède des erreurs d'opération si A n'est pas équivalent à A' et si A peut être modifié pour le rendre équivalent à A', uniquement en changeant un ou plusieurs outputs de certains états de A (sans ajouter ou supprimer d'états). Concrètement, une erreur d'opération correspond à la génération d'un mauvais output lors de la survenance d'un événement donné dans un état donné ;

- **les erreurs de transfert** : on dit qu'un système A possède des erreurs de transfert si A n'est pas équivalent à A' et si A peut être modifié, pour le rendre équivalent à A', uniquement en changeant l'état de destination de certaines transitions. Concrètement, une erreur de transfert correspond à un mauvais passage d'état lors de la survenance d'un événement donné dans un état donné.

Les erreurs d'opération

A titre d'exemple, si on envisage, que la survenance de l'événement T_Creq dans l'état E_1 provoque la génération d'un DR au lieu d'un CR comme cela est exigé dans la table de la figure 7.2, le système ainsi décrit contient une erreur d'opération. Cette erreur sera cependant détectée par les trois méthodes venant d'être exposées. En effet, chacune d'entre elles prévoit l'étude de la survenance de l'événement T_Creq dans l'état E_1 :

Input :	. . . . .	T_Creq	. . . . .
Etat :	 1		3 . . . .
Output observé:		DR (erreur)	
Output attendu:		CR	

Observant les résultats produits lors de l'utilisation d'une séquence, l'utilisateur (réalisateur des tests) pourra aisément détecter l'erreur par comparaison entre les outputs observés et les outputs attendus de part les spécifications (table d'états). Il en est de même pour les autres erreurs d'opération.

Les erreurs de transfert

Si on envisage que la survenance de l'événement CC dans l'état E_3 provoque le passage à l'état E_2 au lieu de l'état E_4 mentionné dans la table de la figure 7.2, le système ainsi décrit contient une erreur de transfert.

* La séquence de vérification définie à la section 7.3.1.3 appliquée à cette implémentation fautive fournira les résultats suivants :

Input :	. . . . .	CC	DR	. . . . .
Etat effectif :	 3	2(erreur)	1	. . . .
Outputs observés:	. . . T_Cconf	ERR (erreur)		. . . .
Outputs attendus:	T_Cconf	N_Dreq		

L'application de la DS {DR} après l'événement CC permet de déceler l'erreur de transition (différence entre l'output attendu et l'output observé). En effet, de par la définition de la DS, l'output est différent selon l'état initial. Ainsi, l'observation d'un ERR (lors de l'émission d'un CC) est typique de l'état 2 alors que l'output spécifique à l'état 4 (N_Dreq) aurait dû être observé.

* Pour la même raison, la séquence caractéristique permettra elle aussi de détecter l'erreur (par application de la séquence {DR} après chaque événement)

* La séquence de test générée par la méthode de parcours des transitions ne permet pas quant à elle de détecter une telle erreur. En effet, l'analyse des outputs ne montrera aucune différence par rapport à un système correct :

Input :	. . . . .	CC		CR	. . . .
Etat effectif :	. . . .	3	2(erreur)		1
Output observé:	. . .	T_Cconf		ERR	. . . .
Output attendu:	. . .	T_Cconf		ERR	

Longueur des séquences générées

Parmi les trois méthodes, la méthode de parcours des transitions est celle qui fournit la plus courte séquence de test. La longueur des séquences générées par les deux autres méthodes dépend fortement de la longueur de la DS et de l'ensemble W. Etant donné que ces séquences avaient dans notre exemple une longueur de 1, des séquences relativement courtes ont pu être définies. Mais ce n'est pas toujours le cas.

7.3.3. Le problème de synchronisation

Présentation du problème

La génération de séquences de test ne peut se limiter à l'application pure et simple d'une des trois méthodes décrites ci-dessus. En effet, les séquences de test générées selon ces méthodes le sont dans l'hypothèse où le testeur actif et le centre de test résident sur la même machine. Or, il faut rappeler que l'architecture générale permettant de tester une implémentation de protocole est décentralisée comme l'indique la figure 7.6.

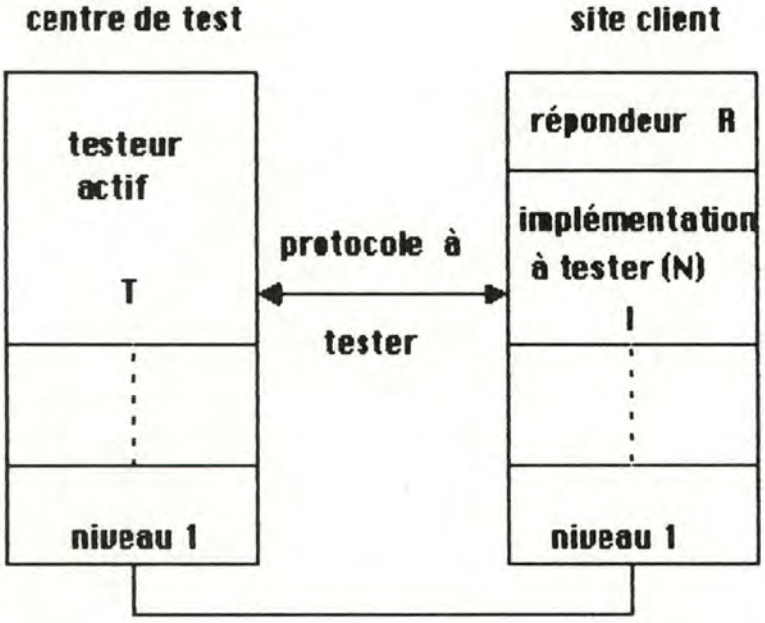


Figure 7.6 : architecture générale des tests.

Etant donné que le testeur et le répondeur sont répartis sur deux sites distincts, se pose le problème dit de "**synchronisation**". En effet, les séquences de test obtenues précédemment ne tiennent pas compte de la synchronisation devant exister entre le testeur actif et le répondeur, en ce sens que l'un d'eux doit parfois attendre, avant de pouvoir poursuivre sa séquence de test, la durée d'une transaction dans laquelle il n'a pas pris part.

Notations et définitions

Dans la suite de notre étude, il sera fait référence, au même exemple que celui utilisé pour la présentation des méthodes de génération automatique, à savoir le protocole de transport illustré à la figure 7.1. Considérons maintenant une des **séquences de tests (ST1)** obtenues selon la méthode de parcours des transitions décrite précédemment. Les événements attendus par l'implémentation en provenance du répondeur ou du testeur sont représentés en caractères gras, alors que les événements à produire par l'implémentation à destination du répondeur ou de l'entité paire sont représentés en italique. La lettre T (respectivement R) mentionnée devant chaque événement signale que celui-ci concerne un échange de messages (émission ou réception) entre l'implémentation et le testeur (respectivement le répondeur) :

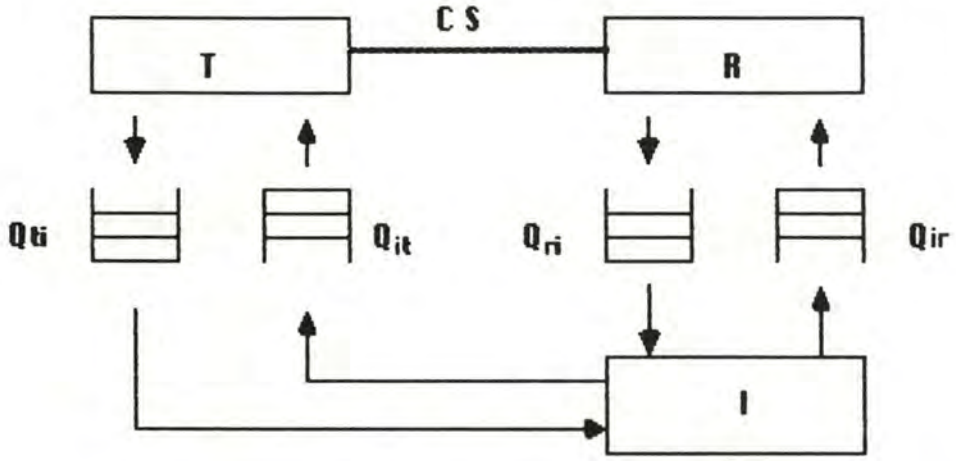
1 **T.CR**	2 **R.T_Dreq**	1 **T.CC**	1 **T.DT**	1 **T.DR**	1 **T.CR** 2
R.T_Cind	*T.DR*	-	-	-	*R.T_Cind*
T.CR	1 **R.T_Creq**	3			**(ST1)**
T.ERR	*T.CR*	.....			

Cette séquence est donc une suite d'événements à communiquer à l'implémentation. Toute suite de messages reçus et/ou envoyés (suite pouvant éventuellement contenir qu'un seul élément) est appelée **transition de base.** Ainsi les suites (T.CR,R.T_Cind), (R.T_Dreq,T.DR) ou même (T.CC) sont des transitions de base.

L'architecture générale de la figure 7.6 peut être modélisée par trois processus :

- **I** (implémentation à tester qui consiste en un système à états finis;
- **T** (testeur actif) ;
- **R** (répondeur) ;

communiquant entre eux au moyen de files FIFO. Cette nouvelle représentation du système est illustrée à la figure 7.7. Le système possède un état initial où toutes les files sont vides.



Qxy désigne la file d'inputs provenant du processus x et destinés au processus y.

Figure 7.7 : modélisation de l'architecture générale.

Lors du déroulement d'un test, le testeur et le répondeur peuvent recevoir des messages de l'implémentation à tester via les files Qit et Qir, tandis que celle-ci peut recevoir des messages en provenance du testeur et du répondeur, via respectivement les files Qti et Qri ; ces messages conduisent l'implémentation que l'on teste à réaliser de nouvelles transitions.

Pour mieux mettre en évidence le problème de synchronisation, nous allons faire abstraction, par la suite, du type précis des messages échangés dans le système. Ainsi, il ne sera tenu compte que du fait que le message est reçu (R, received) ou envoyé (S, send) avec la mention de la file concernée (it, ir, ti, ri). La séquence des messages abstraits R et T correspondant à une transition de base est appelée **interaction de base** ou **pas d'exécution**. Par exemple, les interactions de base correspondant aux transitions de base (T.CR,R.T_Cind), (R.T_Dreq,T.DR) et (T.CC) sont respectivement Rti Sir, Rri Sit et Rti . Une **séquence d'interactions de base** peut être obtenue à partir de la séquence de test ST1. Nous la noterons SIB1 :

Rti Sir Rri Sit Rti Rti Rti Rti Sir Rti Sit Rri Sit ... **(SIB1)**

On constate qu'à toute transition de base de ST1 on peut associer une et une seule interaction de base ou pas d'exécution. Les pas d'exécution possibles sont de 4 types :

1. Rxy Sxy
2. Rxy Sxz
3. Rxy Sxy Sxz
4. Rxy où x,y et z représentent chacun un des processus I, R et T (peu importe lequel).

Les pas d'exécution de type 1,2 ou 3 représentent la réception d'un message (du répondeur ou du testeur) suivi de l'envoi d'un message (au testeur et/ou au répondeur). Les pas de type 4 représentent la réception d'un événement sans qu'il n'y ait de message à envoyer suite à celui reçu.

La SIB1 donnée ci-dessus peut être retranscrite (SIB2) en mettant en évidence les pas d'exécution :

Rti Sir / Rri Sit / Rti / Rti / Rti / Rti Sir / Rti Sit / Rri Sit **(SIB2)**

Les différentes interactions de base sont les suivantes. :

1. Rti (type 4)
   Réception d'un événement du testeur sans production de réponse; cette transition de base est par exemple celle associée au troisième input (CC) de la séquence ST1 ;

2. Rri (type 4)
   Réception d'un événement du répondeur sans production de réponse ;

3. Rti Sit (type 1)
   Réception d'un événement du testeur avec production d'un message pour le testeur ; ex. septième input (CR) de la séquence ST1 ;

4. Rti Sir (type 2)
   Réception d'un événement du testeur avec production d'un message pour le répondeur ; ex. premier input (CR) de la séquence ST1 ;

5. $R_{ri}\ S_{it}$ (type 2)
   Réception d'un événement du répondeur avec production d'un message pour le testeur ; ex. deuxième input (T_Dreq) de la séquence ST1 ;

6. $R_{ri}\ S_{ir}$ (type 1)
   Réception d'un événement du répondeur avec production d'un message pour le répondeur ;

7. $R_{ti}\ S_{it}\ S_{ir}$ (type 3)
   Réception d'un événement du testeur avec production d'un message pour le testeur et le répondeur ;

8. $R_{ri}\ S_{ir}\ S_{it}$ (type 3)
   Réception d'un événement du répondeur avec production d'un message pour le répondeur et le testeur ;

On peut remarquer que les transitions de base 2, 6, 7 et 8 n'apparaissent pas dans la séquence d'interactions de base (SIB1) déduite de la partie de séquence de test (ST1), utilisée comme illustration.

Il est à présent indispensable d'identifier à partir de la séquence SIB1 les messages à recevoir (R_{xy}) ou à envoyer (S_{xy}) à chacun des processus T et R impliqués dans le test.

La construction de la SIB propre à T (que l'on notera SIB_T) peut être réalisée en parcourant un à un les pas d'exécutions de la SIB du processus I (ces pas d'exécution sont mis en évidence dans la séquence SIB2 ci-dessus) et en effectuant les transformations suivantes :

- si dans le pas d'exécution analysé il est mentionné une interaction du type R_{ti}, alors écrire dans la séquence SIB_T l'interaction opposée S_{ti} ;
- si dans le pas d'exécution analysé il est mentionné une interaction du type S_{it}, alors écrire dans la séquence SIB_T l'interaction opposée R_{it} ;
- si dans le pas d'exécution analysé le processus T n'est mentionné nulle part, écrire dans la séquence SIB_T le caractère spécial @.

De même, on obtient la séquence d'interactions associée au processus R (que l'on notera SIB_R) par application de la méthode décrite ci-dessus, en remplaçant, dans le raisonnement, précédent t_i et i_t respectivement par r_i et i_r.

Ainsi les trois séquences d'interactions de base pouvant être déduites de la séquence de test d'origine sont :

I :	Rti Sir Rri Sit Rti Rti Rti Rti Sir Rti Sit Rri Sit	**(SIB1)**
T :	Sti Rit Sti Sti Sti Sti Sti Rit Rit	**(SIBT)**
R :	Rir Sri @ @ @ Rir @ Sri @	**(SIBR)**

Remarque : Les séquences SIBT et SIBR sont celles qui pourraient servir de point de départ pour construire les scénarios de tests à exécuter respectivement sur le centre de test et le site client (par exemple pour les scénarios du NBS).

Détection des problèmes de synchronisation

L'exécution en parallèle des séquences de test SIBT et SIBR soulève le problème de synchronisation présenté plus haut. Pour illustrer ce problème, reprenons rapidement l'exécution de ces séquences. Le testeur entame le test et envoie un message à l'implémentation (I). Après réception d'un message en provenance de I, le répondeur envoie à celle-ci une réponse. Puis c'est au tour du testeur d'envoyer quatre messages à I. Les trois premiers ne nécessitent pas de réponse de la part de l'implémentation. A la réception du quatrième, celle-ci doit envoyer un message au répondeur. Le testeur envoie un nouveau message à I qui y répond immédiatement. Puis c'est au tour du répondeur à produire un message destiné à l'implémentation. C'est à ce moment que se manifeste le problème de la synchronisation, car il est impossible pour le répondeur de savoir qu'il doit seulement transmettre son message après que T ait reçu sa réponse de l'implémentation. A ce niveau, il est bon de constater que le processus I représentant l'implémentation à tester n'est pas à l'origine du problème de synchronisation, mais que celui-ci se situe au niveau du répondeur et du testeur. L'implémentation à tester ne fait que réagir aux événements produits par ceux-ci.

De manière plus générale, un problème de synchronisation survient lorsqu'une séquence d'interactions (SIBT ou SIBR) possède une interaction de type S (send) précédée d'un ou plusieurs caractères @. Dans la séquence SIBR ci-dessus le problème de synchronisation est indiqué par @ .

On peut aussi détecter d'éventuels problèmes de synchronisation d'une autre manière, par exemple en observant les séquences d'interactions obtenues pour l'implémentation à tester (SIB1). Ainsi, la présence de paires d'interactions non synchronisables (c'est-à-dire de paires d'interactions où la deuxième interaction ne peut suivre la première sans générer un conflit de synchronisation) est l'indication d'un tel conflit. Si l'on repère, par exemple, dans la séquence d'interactions de base associée à l'implémentation la paire Rti Rri, on constate (cf SIB2) que Rti constitue à lui seul un pas d'exécution. La séquence d'interactions de base propre à R (SIBR) contiendra le caractère @ (étant donné que "r" n'est pas repris dans Rti) suivi de Sri (correspondant au Rri). Ce qui constitue la condition d'apparition d'un problème de synchronisation.

Mis à part Rti Rri, les autres paires et sous-séquences non synchronisables sont :

- Rri Rti ;
- Rti Sit Rri ; **(liste1)**
- Rri Sir Rti.

Dans la séquence SIB1, la sous-séquence non synchronisable à l'origine du conflit est Rti Sit Rri.

Solutions du problème

Une solution pourrait à priori résoudre le problème de synchronisation. Elle consisterait dans notre exemple à communiquer au répondeur la durée des deux dernières interactions du testeur avec l'implémentation (Sti Rit) ; cependant il s'avère difficile d'imposer de telles contraintes sur l'exécution de processus parallèles.

La façon la plus efficace d'éviter l'apparition de tels problèmes est d'empêcher la génération de séquences de test pouvant mener à ces conflits (l'approche préventive est préférée à une approche de résolution de conflit). Les trois méthodes de génération qui ont été expliquées dans la section 7.3.1. vont être modifiées de manière à ne produire que des séquences de test dont les paires d'interactions de base sont synchronisables. L'idée sous-jacente à ces nouvelles modifications est de vérifier si tout nouvel élément que l'on ajoute à la séquence d'interactions est synchronisable avec ses prédécesseurs.

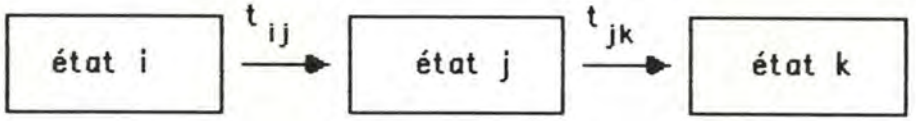
Cette vérification se fait sur base d'une liste (semblable à la liste1) contenant l'ensemble des sous-séquences non synchronisables.

Pour plus d'informations, on peut consulter Sarikaya et Bochman (1983) qui donnent et commentent les modifications à apporter aux algorithmes de génération de séquences de test afin d'éviter ces problèmes de synchronisation.

En tenant compte des modifications apportées aux algorithmes de génération de séquences, il est possible d'obtenir des SIB synchronisables pour l'implémentation que l'on teste, à partir desquelles seront déduites les SIB pour le testeur (SIBT) et le répondeur (SIBR). Les séquences SIBT et SIBR seront alors transcrites en scénarios de test qui contiendront à la fois les événements inputs attendus de l'implémentation que l'on teste et les événements outputs à produire, de même que les paramètres associés à ces événements.

Cependant, en fonction des spécifications de certains protocoles, il est parfois impossible d'éviter l'apparition de problèmes de synchronisation, même en utilisant les algorithmes améliorés de génération de test.

Comme exemple, supposons que l'on désire tester, pour une implémentation I , le parcours successif des deux transitions t_{ij} (de l'état i à l'état j) et t_{jk} (de l'état j à l'état k), comme l'indique le shéma ci-dessous :



En supposant que :

- la transition t_{jk} nous amenant en k est une transition du type Rri ou Rri Sit ou Rri Sir Sit,
- la transition t_{ij} est du type Rti ou Rti Sit,

on obtient dans tous les cas une séquence non synchronisable (voir la liste1 ci-dessus) :

- soit $R_{ti}\ R_{ri}$ [...],
- soit $R_{ti}\ S_{it}\ R_{ri}$ [...].

Si l'on désire malgré tout tester ces deux transistions t_{ij} et t_{jk}, il est indispensable d'adjoindre à l'architecture existante un mécanisme de synchronisation entre le testeur et le répondeur. Il s'agit en fait de relier ces deux processus par une **connexion de synchronisation (CS)** via laquelle seraient véhiculés des messages de synchronisation. Cette connexion de synchronisation est représentée en pointillé dans la figure 7.7.

Les scénarios obtenus par les méthodes précédentes devront être remaniés afin de tenir compte des commandes de synchronisation. Celles-ci sont au nombre de deux :

- la commande **W (wait)** ; elle signale au processus testeur (ou répondeur) qu'il doit attendre la réception d'un message en provenance du répondeur (respectivement testeur) sur la connexion de synchronisation, avant de poursuivre la séquence de test.

- la commande **S (send)** ; elle indique au processus testeur (ou répondeur) qu'il doit envoyer au processus répondeur (respectivement testeur), sur la connexion prévue à cet effet, un message de synchronisation.

Ainsi, si l'on désirait résoudre le problème de synchronisation présent dans les séquences SIB_T et SIB_R à l'aide de cette technique, ces deux séquences devraient être modifiées et deviendraient SIB_{T2} et SIB_{R2} :

T : S_{ti} R_{it} S_{ti} S_{ti} S_{ti} S_{ti} S_{ti} R_{it} **S** R_{it} **(SIB_{T2})**

R : R_{ir} S_{ri} @ @ @ R_{ir} **W** S_{ri} @ **(SIB_{R2})**

La séquence de test destinée au répondeur (SIB_{R2}) contient une commande W interrompant le déroulement du test jusqu'à la réception d'un message de synchronisation. Celui-ci sera généré par le testeur sur la connexion de test suite à la commande S, figurant dans la séquence de test SIB_{T2} .

Le processus en attente (c'est-à-dire celui qui a rencontré dans sa séquence de test une commande **W**) devra signaler au processus paire (c'est-à-dire celui qui a rencontré dans sa séquence de test la commande **S**) la réception du message de synchronisation par un accusé de réception (acknowledgement). Le message de synchronisation doit être retransmis jusqu'à la réception de cet acknowledgement (ou jusqu'à concurrence d'un nombre de fois fixé préalablement). Ce mécanisme d'accusé de réception permet de faire face aux problèmes pouvant éventuellement survenir sur la connexion de synchronisation.

8. CONCLUSION

Un ensemble de standards internationaux permettant à des systèmes de gammes différentes ou de constructeurs différents de collaborer pour assurer le transfert d'informations sont à présent disponibles. La présence de tels standards est bénéfique tant pour les utilisateurs (qui voient s'étendre les possibilités d'interconnexion des équipements dont ils disposent) que pour les constructeurs (qui voient apparaître pour leurs produits de nouveaux marchés). Cependant, l'interconnexion de ces différents systèmes ne sera réellement effective que si la mise en oeuvre de ces standards respecte bien la fonctionnalité de ceux-ci. Seule la vérification de conformité des implémentations de protocole permettra d'atteindre un tel objectif.

Ce problème n'a réellement recueilli l'attention internationale que depuis 1980. A partir de cette date, d'importantes recherches visant le développement de techniques de test de conformité d'un équipement aux standards ont été menées par la communauté informatique internationale. Ainsi, au cours de ce travail, différentes architectures permettant d'établir la conformité ou l'absence de conformité d'une implémentation ont été abordées. Une comparaison des différentes méthodes employées a permis de souligner les points forts et les points faibles de chacune d'elles. Se basant sur les résultats de cette analyse, nous avons tenté d'apporter une solution à l'inconvénient majeur de l'ensemble de ces architectures de test, à savoir le manque de modularité de certains composants. Le problème de la génération des tests a également été abordé.

Différents centres de test mettant en oeuvre les diverses méthodes étudiées dans le cadre de ce mémoire sont toujours au stade d'élaboration en Europe et en Amérique du Nord (Canada, Etats-Unis, France, Grande-Bretagne, République Fédérale d'Allemagne, ...).

Conclusion

Dans le cycle de vie d'une implémentation, il est prévu de faire appel aux services d'un tel centre à plusieurs reprises :

par le fournisseur/réalisateur :

* durant le développement de l'implémentation (outil d'aide au développement),
* pour établir les performances de l'implémentation, à l'issue de sa réalisation,
* pour attribuer à une implémentation un "certificat de conformité";

par l'acheteur/utilisateur :

* pour s'assurer de la conformité d'une implémentation qu'il désire se procurer,
* pour contrôler les éventuelles modifications apportées à l'implémentation après son acquisition ;

par plusieurs utilisateurs :

* pour apporter une solution aux problèmes d'incompatibilité pouvant survenir entre deux implémentations certifiées.

Dans un futur proche, ces centres devront offrir à tout réalisateur d'un protocole la possibilité de vérifier le travail qu'il aura réalisé. Aussi faudra-t-il, pour que de tels centres obtiennent la confiance des prochains utilisateurs normaliser l'ensemble des tests à mener pour chaque protocole. Cette nouvelle standardisation assurerait l'uniformité des tests quelque soit la méthode utilisée et le centre de test auquel on s'adresse.

Cependant, il ne faut pas perdre de vue qu'une amélioration dans la formulation des standards permettrait d'atteindre plus rapidement le but recherché par ceux-ci, à savoir l'interconnexion de systèmes hétérogènes. Or, il faut bien se rendre à l'évidence, bien que ces standards existent, bon nombre d'entre eux sont imprécis, notamment en ce sens que leurs options sont partiellement (ou pas du tout) définies. Cela oblige donc le réalisateur et le testeur à choisir arbitrairement ces options, compliquant ainsi la vérification de conformité en multipliant le risque de non-compatibilité entre les versions.

Conclusion

Mis à part le développement et l'expérimentation des architectures proposées, différentes voies de recherche pourraient être poursuivies dans le futur. Celles-ci viseraient entre autres à accroître l'automatisation des tests ainsi que la qualité des renseignements fournis sur le déroulement des tests. Pour ce qui est de la première perspective, on pourrait envisager un système qui, à partir d'un objectif de test choisi par l'opérateur (par exemple, le transfert des données), serait capable de sélectionner et d'assurer l'enchaînement des différents scénarios de test appropriés, et ce, avec autant de jeux de valeurs différents qu'il est nécessaire pour atteindre l'objectif choisi pour le test. Un tel système pourrait être qualifié de système expert. Quant à la seconde perspective, elle permettrait à l'opérateur de détecter plus précisément la cause d'un éventuel problème et ainis d'y remédier plus rapidement.

La disponibilité d'implémentations conformes ne sera vraiment possible que si les différentes méthodes de test sont effectives et reconnues par l'ensemble des centres de recherche internationaux. Or, dans le contexte de compétition économique que nous connaissons, le risque de tomber dans une "*concurrence informatique*" est grand. Cela se traduirait par une diversification des recherches et une absence de coordination entre elles.

LISTE DES FIGURES

Chapitre 2

Chapitre 3

Chapitre 4

Chapitre 5

Chapitre 6

Chapitre 7

REFERENCES A : LIVRES ET NORMES

[DEA] DEASINGTON R. J. (1985), X_25 explained : Protocols for packet switching networks, Ellis Horwood Limited, Chichester (U. K.).

[NPL] RAYNER D. et HALE R. W. S. (1981), Protocol testing - Towards proof ?, Proceedings - volume 2 - Testing and certification, National Physical Laboratory, Teddington (U. K.).

[PS2] SUNSHINE, C. (eds) (1982), Proceedings of the IFIP WG6.1, Second international workshop on protocol specification, testing and verification, North Holland publisher company.

[PS3] RUDIN, H. et WEST C. H. (eds) (1983), Proceedings of the IFIP WG6.1, Third international workshop on protocol specification, testing and verification, North Holland publisher company.

[PS4] YEMINI Y., STROM R. et YEMINI S. (eds) (1983), Proceedings of the IFIP WG6.1, Fourth international workshop on protocol specification, testing and verification, North Holland publisher company.

[RFM] International Standards Organization, Information processing systems - Open Systems Interconnection - basic reference model, également publié dans Computer Networks, vol. 5, 81-118 (1981).

[STA] STALLINGS, W. (1985), Data computer communications, Collier Macmillian publisher, London (U. K.).

[TAN] TANENBAUM, A. S. (1981), Computer Networks, Prentice-Hall, Englewood Cliffs (U. K.).

[TRP] International Standards Organization, Information processing systems - Open Systems Interconnection - connection oriented transport protocol specification, ISO/DIS 8073 (octobre 1983).

[TRS] International Standards Organization, Information processing systems - Open Systems Interconnection - transport service definition, ISO/DIS 8072 (avril 1984).

[X25] CCITT recommendation X_25, Level 3 as applicable to the packet switched network of the Régie des télégraphes et des téléphones, Régie T. T. Service de transmission de données (11 avril 1980).

[Z10] CCITT recommendation Z100-Z101, Introduction to SDL a specification and description language for protocols (mai 1983).

REFERENCES B : ARTICLES

ANSART J. P. (1981 a), "Documents remis à la commission réseaux du club de la péri-informatique : projet pilote RHIN ; outils de certification de protocoles de haut niveau; protocole expérimental - service et protocole de transport", Agence de l'informatique, 92084 Paris-La Défense.

ANSART J. P. (1981 b), "Tools for the certification of standardized protocols. Cerbère and Genepi", dans [NPL], 127-130.

ANSART J. P. (1981 c), "Test and certification of standardized protocols", dans [NPL], 119-126.

ANSART J. P. (1982 a), "GENEPI/A, a protocol independent system for testing protocol implementation", dans [PS2], 523-528.

ANSART J. P. (1982 b), "CERBERE, a tool to keep an eye on high level protocols", dans [PS2], 529-537.

ANSART J. P., CHARI V., MEYER M., RAFIQ O. et SIMON D. (1983), "Description , simulation and implementation of communication protocols using PDIL", Computer Communication Review, vol. 13, no 2, 112-120.

BILLIARD J. F. (1983), "Methodology and tools for qualitative protocols validation", dans [PS3], 445-454.

BLUMER T. P. et BURRUS J. C., "Generating a service specification of a connection management protocol", dans [PS2], 161-170.

Bibliographie

BLUMER T. P. et TENNEY R. L. (1982), "A formal specification technique and implementation method for protocols", Computer Networks, vol. 6, 201-217.

BOCHMANN G. V. (1981), "On the theorical power of some testing methods prepared for the INWG/NPL Workshop on Protocol Testing", dans [NPL], 15-23.

BOCHMANN G. V. et autres (1982), "Some experience with the use of formal specifications", dans [SP2], 171-185.

CLARK G. E. et WONG M. K. (1985), "Verifying conformance to the X_25 standard", Data Communications, 153-161.

CHOW T. S. (1978), "Testing software designed modeled by finite-state machines", IEEE Transactions on Software Engineering, vol. 4, no 3, 178-186.

COWIN G. W., HALE R. W. S. et RAYNER D. (1983), "Protocol product testing - some comparisons and lessons", dans [PS3], 477-492.

DAVIDSON I. (1985), "Testing conformance to OSI standards", Computer Communications ", vol. 8, no 4, 170-179.

GIEßLER A. (1983), "Testing and diagnosis aids for higher level protocols", dans [PS3], 407-420.

GÖNENÇ G. (1970), "A method for the design of fault detection experiments", IEEE Transactions on Computer, juin, 551-558.

HALE R. W. S. et RAYNER D. (1983), "Protocol product testing - some comparisons", European Teleinformatics Conference, Elsevier Science Publishers, 599-607.

HÄNLE J. O. (1985), "The teletex test system PETRUS", dans [PS4], 603-610.

HARVEY G. A. (1983), "The routing certification system", dans [PS3], 465-476.

HENLEY R. F. L. (1981 a), "Implementation assessment of transport and network services : the test responder specification", dans [NPL], 73-104.

Bibliographie

HENLEY R. F. L. et RAYNER D. (1981 b), "Implementation assessment of transport network services. An informal description of tests", dans [NPL], 105-117.

HUNT R. (1984)," Open systems interconnection - the transport layer", Computer Communications, vol. 7, no 4, 186-197.

LARMOUTH J. (1985), "OSI enters the implementation phase", Data Processing, vol. 5, no 1, 11-14.

LINN R. J. et NIGHTINGALE J. S. (1983 a), "Testing OSI protocols at the National Bureau of Standards", Proceedings of the IEEE, vol. 71, no 12, 1431-1434.

LINN R. J. et NIGHTINGALE J. S. (1983 b), "Some experience with testing tools for OSI protocol implementations", dans [PS3], 521-531.

LINN R. J. et McCOY W. H. (1983), "Producing tests for implementations of OSI protocols", dans [PS3], 505-519.

LINN R. J. (1985), "An evaluation of the ICST test architecture after testing class 4 transport", dans [PS4], 611-621.

McCOY W. H., COLELLA R. P. et WALLACE M. A. (1981), "Assessing the performance of high-level computer network protocols", dans [NPL], 25-53.

MELICI J. A. (1982), "The BX.25 certification facility", Computer network , vol. 6, 319-329.

MILLS K. L. (1984), "Testing OSI protocols : NBS advances the state of the art", Data Communications, mars, 277-285.

NATIONAL BUREAU OF STANDARDS (1983 a), Institute for Computer Sciences and Technology, Center for Computer Systems Engineering, Systems and Network Architecture Division, Draft Report, "Specification of a remote scenario interpreter for implementations of the ICST transport protocol".

Bibliographie

NATIONAL BUREAU OF STANDARDS (1983 b), Institute for Computer Sciences and Technology, Center for Computer Systems Engineering, Systems and Network Architecture Division, Draft Report, "Users guide to the testing system for implementations of the ICST transport protocol".

NIGHTINGALE J. S. (1982), "Protocol testing using a reference implementation", dans [PS2], 513-522.

NPL Protocol Standards Group (1981), "Implementation assessment of transport and network services ; the test responder specification", dans [NPL], 73-104.

PALAZZO S., FOGLIATA P. et LE MOLI G. (1983), "A layer-independent architecture for a testing system of protocol implementations", dans [PS3], 393-405.

PAVEL J. R. et DWYER D. J. (1985), "Some experiences of testing protocol implementations", dans [PS4], 657-677.

PROBERT R. L. et URAL H. (1983), "Requirements for a test specification language for protocol implementation testing", dans [PS3], 437-443.

RAFIQ O. et ANSART J. P. (1983), "VADILOC, a protocol validator and its applications", dans [PS3], 189-197.

RAFIQ O. et HADDAD J. (1985), "Description of protocol testing scenarios", Direction Technique , Bull-Transac, Massy, France.

RAFIQ O. (1985 a), "A good approach for testing protocol implementations", Direction Technique , Bull-Transac, Massy, France.

RAFIQ O. (1985 b), "Tools and methodology for testing OSI protocols entities", Direction Technique , Bull-Transac, Massy, France.

RAFIQ O., CATANET R. et CHRAIBI C. (1985), "Towards an environment for testing OSI protocols", Direction Technique, Bull-Transac, Massy, France.

RAYNER D. (1981), "Protocol implementation assessment : Philosophy and Architecture", dans [NPL], 55-72.

Bibliographie

RAYNER D. (1982), "A system for testing protocols implementations", Computer Networks, vol. 6, 383-395. Egalement publié dans [PS2], 539-554.

RAYNER D. (1983 a), "Towards an objective understanding of conformance", dans [PS3], 493_503.

RAYNER D. (1983 b), "Progress in testing protocol implementations", International Conference on Communications, Boston, 1323_1327.

SARIKAYA B. et BOCHMANN G. V. (1982), "Some experience with test sequence generation for protocols", dans [PS2], 555-567.

SARIKAYA B. et BOCHMANN G. V. (1983), "Synchronization issues in protocol testing", Computer Communication Review, vol. 13, no 2, 121-128.

SUNSHINE C. A. (1983), "Experience with automated protocol verification", dans [PS3], 229-236.

TANENBAUM A. S. (1981), "Networks protocols", Computing Survey, vol. 13, no 4, 453-489.

URAL H. et PROBERT R. L. (1983), "User-guided test sequence generation", dans [PS3], 421-436.

VON STUDNITZ P. (1983), "Transport protocols : their performance and status in international standardization (july 1982)", Computer Networks , vol. 7, 27-35.

WEAVING K. (1980), "Euronet reference and test center", Computer Communications , vol. 3, no 5, 221-223.

WEAVING K. (1981), "Verification of high level protocol implementations", dans [NPL], 131-145. Egalement publié dans Computer Communications , vol. 4, no 2, 56-60.

YEMINI Y. et KUROSE J. F. (1982), "Can current protocol verification techniques guarantee correctness ?", Computer Networks , vol. 6, 377-381.

ZENG H. X. et RAYNER D. (1985), "Gateway testing techniques", dans [PS4], 637-655.

ANNEXES

ANNEXE 1

LES PROTOCOLES DES COUCHES 3 ET 4

A plusieurs reprises au cours de ce travail, il a été fait référence, en tant qu'exemple, aux procédures mises en oeuvre par la couche réseau ou la couche transport. Le but de cette annexe est de décrire brièvement l'objectif et les principes de fonctionnement de chacune de ces couches.

1. *La recommendation X25*

La recommendation X25 du Conseil Consultatif International pour la Téléphonie et la Télégraphie (CCITT) est relative à l'interface entre un équipement terminal de traitement de données (ETTD) et un équipement de terminaison de circuit de données (ETCD) pour terminaux fonctionnant en mode paquet, raccordés à un réseau public de transmission de données. A ce titre, trois niveaux y sont repris :

- le niveau 1 décrivant les caractéristiques physiques, électriques, fonctionnelles et de procédure pour établir, maintenir et déconnecter la liaison physique entre l'ETTD et l'ETCD (référence à la recommendation X21 ou X21bis) ;
- le niveau 2 décrivant la procédure d'accès à la liaison pour l'échange de données sur la liaison entre l'ETTD et l'ETCD (référence au protocole LAP/B) ;
- le niveau 3 décrivant le format des paquets et les procédures de commande pour l'échange des paquets entre l'ETTD et l'ETCD. Ces paquets contiennent des informations de supervision et les données de l'usager.

De plus amples informations concernant ce niveau ainsi que les niveaux 1 et 2 sont données dans [STA], [DEA], [TAN] et [X25].

Annexe 1 : les protocoles des couches 3 et 4

X25/3 : Phase d'établissement d'une connexion

Cette partie décrit la procédure d'établissement d'une connexion réseau entre 2 ETTDs raccordés à un réseau paquet en la complétant par les interactions entre les niveaux 3 et 4.

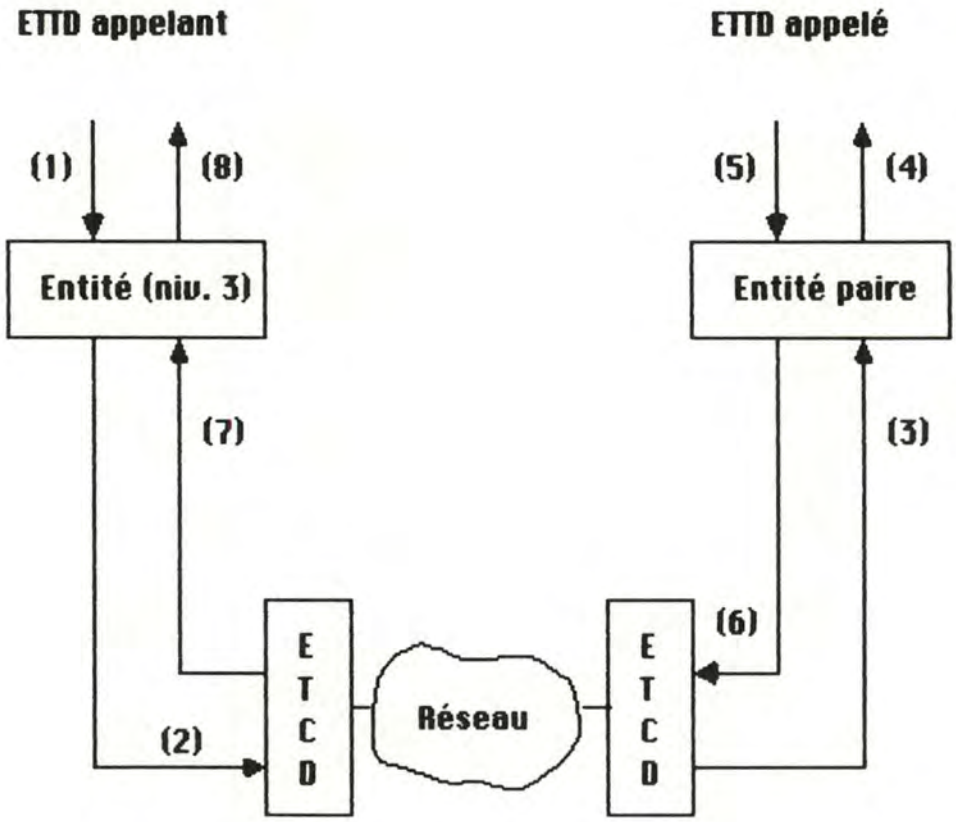


Figure A1 : établissement d'une connexion réseau.

(1) CONNECT REQUEST : demande (4 --> 3) d'établissement d'une connexion réseau.

(2) CALL REQUEST : l'ETTD appelant émet un paquet d'appel (CR) sur une voie logique libre, c'est-à-dire non utilisée pour un autre circuit virtuel.

(3) INCOMING CALL : le réseau (ETCD côté appelé) émet vers l'ETTD appelé un paquet d'appel entrant sur une voie logique libre.

(4) CONNECT INDICATION : le niveau 3 appelé prévient son niveau 4 d'une demande d'établissement de connexion réseau.

(5) CONNECT RESPONSE : le niveau 4 accepte l'établissement de la connexion réseau. Dans le cas contraire, une procédure de libération (DISCONNECT) est déclenchée.

(6) CALL ACCEPTED : l'ETTD appelé indique qu'il accepte la communication en émettant vers le réseau un paquet "communication acceptée" sur la même voie logique que l'appel entrant reçu. Il pourra par la suite y transmettre des paquets de données.

(7) CALL CONNECTED : le réseau (ETCD côté appelant) indique à l'ETTD avec lequel il est en contact que la communication est établie. Pour ce faire, il lui transmet un paquet "communication établie" sur la même voie logique que le premier paquet d'appel.

(8) CONNECT CONFIRMATION : le niveau 3 appelant prévient son niveau 4 que l'établissement de la connexion réseau a réussi et qu'il peut dès lors commencer à émettre des données.

Annexe 1 : les protocoles des couches 3 et 4

PHASE	ETTD APPELANT	ETTD APPELE
Etablissement d'une connexion réseau	CALL REQUEST CALL CONNECTED	INCOMING CALL CALL ACCEPTED
Transfert de données et contrôle de flux	DATA RECEIVER READY RECEIVER NOT READY REJECT	
Transfert de données express	INTERRUPT REQUEST CONFIRMATION	INTERRUPT INDICATION CONFIRMATION
Réinitialisation	RESET REQUEST CONFIRMATION	RESET INDICATION CONFIRMATION
Reprise	RESTART REQUEST CONFIRMATION	RESTART INDICATION CONFIRMATION
Libération	CLEAR REQUEST CONFIRMATION	CLEAR INDICATION CONFIRMATION

Table 1 : tableau récapitulatif des différents types de paquets.

2. *Le protocole de transport*

Le protocole de transport est un des ensembles de règles produit dans le but de faciliter l'interconnexion des divers systèmes d'ordinateurs.

Dans le contexte du modèle de référence OSI, la couche transport est la couche 4. Elle fournit un service de transport (disponible grâce aux points d'accès au service), aux entités session (couche 5), et ce, en exploitant les services fournis par la couche réseau (couche 3). La figure A2 situe la couche transport dans son environnement.

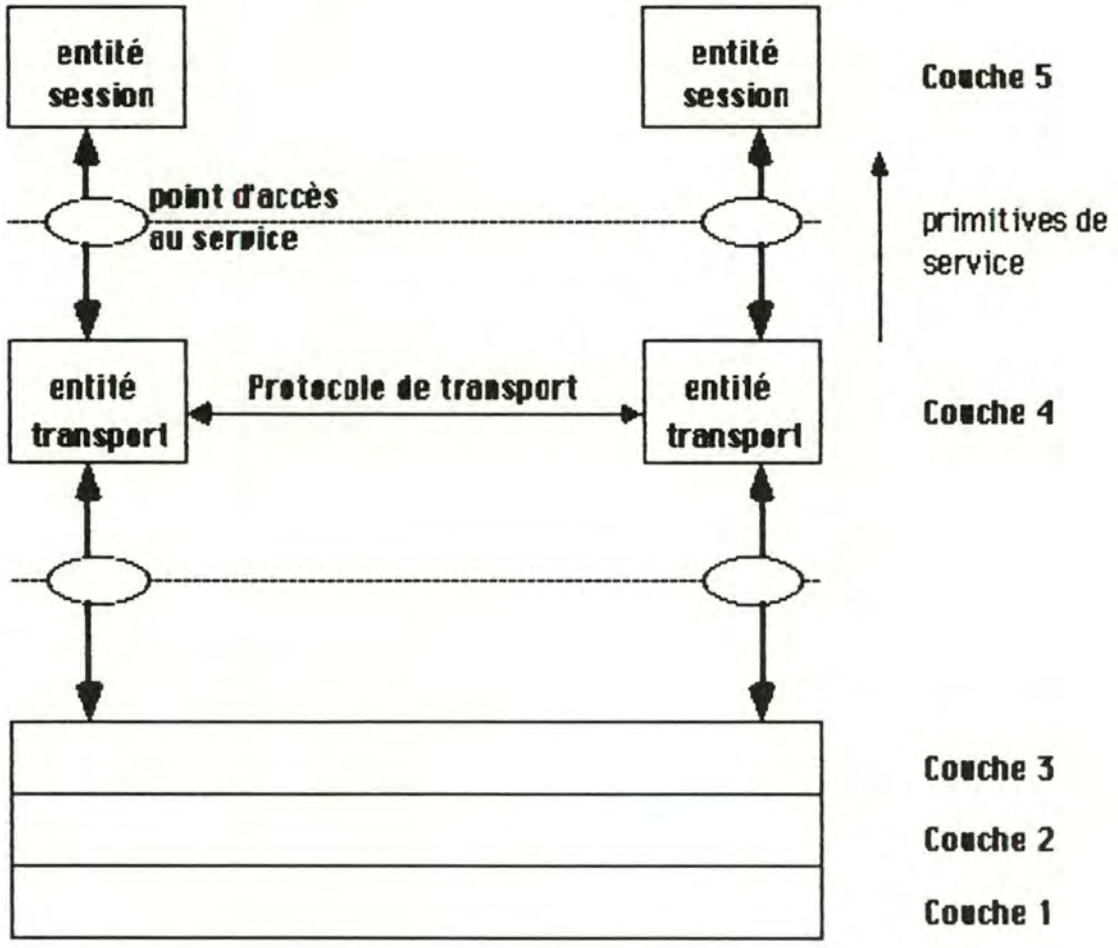


Figure A2 : la couche transport dans l'environnement OSI.

Annexe 1 : les protocoles des couches 3 et 4

La couche transport soulage ses utilisateurs de toutes préoccupations concernant le transport fiable et économique des données.

Le protocole fait la distinction entre différentes classes de service pouvant être offertes à la couche session. La classe de service appropriée est sélectionnée par l'entité de session de manière à pouvoir pallier à certaines défaillances éventuelles du réseau. Les différentes classes possibles sont au nombre de 5 :

- classe 0 : un tel protocole ne fournit pas une amélioration du service proposé par la couche réseau, qui est dès lors supposé suffisamment efficace;

- classe 1 : en plus de la classe 0, la classe 1 permet de corriger certaines erreurs pouvant être induites par le réseau (par exemple, déconnexion subite d'une liaison réseau);

- classe 2 : le but de cette classe est d'assurer le multiplexage de plusieurs connexions de transport sur une connexion réseau supposée efficace;

- classe 3 : en plus de la classe 2, la classe 3 permet de corriger certaines erreurs pouvant être induites par le réseau (par exemple, la déconnexion subite d'une liaison réseau);

- classe 4 : en plus de la classe 3, la classe 4 permet de détecter des erreurs dans les éléments de protocole de transport résultant d'un service réseau de faible qualité (par exemple, la perte, la duplication d'éléments de protocole ou la modification de certains champs de ceux-ci, ...)

La couche transport fournit un service dont les caractéristiques sont les suivantes :

- l'indépendance vis-à-vis des couches inférieures; le service de transport sélectionné pourra être fourni quel que soit le type de réseau (couche 3) ou des différents réseaux (si une concaténation de ceux-ci s'avère utile) impliqué(s) dans la connexion réseau ;

- la possibilité pour l'utilisateur de sélectionner certaines valeurs pour la qualité requise du service (ex. classe de service, throughput, taux résiduel d'erreurs souhaité, coût à ne pas dépasser, ...).

En résumé, la couche transport est sensée optimiser les ressources de communication disponibles pour assurer les performances réclamées par les utilisateurs.

Le service de transport se présente pour ses utilisateurs sous la forme de primitives de service auxquelles sont associés des paramètres. Ces primitives de service sont définies conceptuellement en ce sens qu'elles ne spécifient aucune réalisation syntaxique particulière. De même, elles n'imposent aucune contrainte sur la façon dont l'interface entre les couches transport/session est réalisée. Une telle spécification d'un service est nécessaire, particulièrement pour garantir l'indépendance de deux couches adjacentes dans le contexte du modèle de référence OSI. Les primitives et leurs paramètres respectifs sont présentés à la table 2.

Ces primitives sont rassemblées selon la phase dans laquelle elles interviennent :

- établissement d'une connexion (phase dont le but est d'établir une connexion transport entre deux utilisateurs distants du service transport);
- transfert de données sur une connexion (phase dont le but est de permettre la transmission "duplex" des données communiquées par les utilisateurs de la connexion transport établie précédemment);
- libération de la connexion.

phases	primitives	paramètres
établissement d'une connexion	T_CONNECT.request	(to_address, from_address, quality_of_service, expedited_data_option, user_data)
	T_CONNECT.indication	(to_address, from_address, quality_of_service, expedited_data_option, user_data)
	T_CONNECT.response	(from_address, quality_of_service, expedited_data_option, user_data)
	T_CONNECT.confirm	(from_address, quality_of_service, expedited_data_option, user_data)
transfert de données	T_DATA.request	(user_data)
	T_DATA.indication	(user_data)
	T_EXPEDITED.request	(user_data)
	T_EXPEDITED.indication	(user_data)
libération d'une connexion	T_DISCONNECT.request	(user_data)
	T_DISCONNECT.indication	(disconnect_reason, user_data)

Table 2 : les primitives de service de transport.

Annexe 1 : les protocoles des couches 3 et 4

1. établissement d'une connexion

T_CONNECT.request :
primitive (5 --> 4) transmise par l'utilisateur du service transport en vue de l'établissement d'une nouvelle connexion;

T_CONNECT.indication :
primitive (4 --> 5) signalant à l'utilisateur du service transport une demande de connexion;

T_CONNECT.response :
primitive (5 --> 4) transmise par l'utilisateur du service transport après la réception d'un T_CONNECT.indication, et lorsqu'il est favorable à l'établissement d'une nouvelle connexion;

T_CONNECT.confirm :
primitive (4 --> 5) signalant à l'utilisateur du service transport, ayant auparavant réclamé l'établissement d'une connexion de transport (par la primitive T_CONNECT.request), que la connexion désirée est à présent disponible;

2. transfert de données

T_DATA.request :
primitive (5 --> 4) transmise par l'utilisateur du service transport afin d'acheminer sur la connexion préalablement établie les données communiquées en paramètres;

T_DATA.indication :
primitive (4 --> 5) signalant à l'utilisateur du service transport la présence, sur la connexion, de données qui lui sont destinées ;

T_EXPEDITED.request :
primitive (5 --> 4) transmise par l'utilisateur du service transport afin d'acheminer, en mode "express", sur la connexion préalablement établie les données communiquées en paramètres. Ce mode est tel que si l'on utilise successivement les primitives T_EXPEDITED.request et T_DATA.request, les données acheminées au moyen du T_DATA.request ne pourront être communiquées au destinataire avant les données acheminées au moyen du T_EXPEDITED.request;

T_EXPEDITED.indication :
primitive (4 --> 5) signalant à l'utilisateur du service transport la présence de données qui lui sont destinées sur la connexion. Ces données ont été acheminées en mode "express";

3. libération d'une connexion

T_DISCONNECT.request :
primitive (5 --> 4) transmise par l'utilisateur du service transport afin de libérer la connexion transport;

T_DISCONNECT.indication :
primitive (4 --> 5) signalant à l'utilisateur du service transport la terminaison d'une connexion transport. Le motif de terminaison est indiqué en paramètres.
Il peut s'agir :

- d'une terminaison volontaire par l'utilisateur opposé de la connexion (grâce à un T_DISCONNECT.request);
- perte de la connexion réseau;
- expiration d'un timer;
- erreur de protocole;
- élément de protocole incorrect;
- etc, ...

Le lecteur désirant un complément d'informations sur le fonctionnement du protocole de transport pourra consulter [TRP], [TRS], [RFM], [STA] et Von Studnitz (1983).

ANNEXE 2

EXEMPLES DE SCENARIOS DE TEST

Comme il a été annoncé dans la section 4.3.2.4, lors de l'étude de l'architecture du NBS, cette annexe donne un éventail des différents types de scénarios pouvant être exécutés par les interpréteurs de scénarios du centre de test et du site client.

Les fichiers sont identifiés de la manière suivante :

- les initiales "rt" indiquent que le scénario doit être suivi par l'interpréteur du site client (rt = remote tester) tandis que les initiales "tc" indiquent que le scénario doit être suivi par l'interpréteur du centre de test (tc = test center) ;

- le chiffre qui suit ces deux lettres se rapporte à la couche du modèle ISO/OSI concernée par le test ;

- le chiffre suivant fournit des renseignements sur la catégorie du test dont il s'agit :

1 --> Séquences valides de primitives ;
2 --> Séquences invalides de primitives ;
3 --> Variations des paramètres des primitives ;
4 --> Séquences invalides d'éléments de protocole ;
5 --> Variations des paramètres des éléments de protocole ;
6 --> Tests relatifs aux timers ;

- les trois derniers chiffres permettent de distinguer différents groupes de test pour chacune des catégories données ci-dessus.

Par exemple, le nom du scénario "RT_4.1.100" signifie que ce fichier doit être exécuté sur le site client (RT), qu'il s'agit d'un scénario relatif à la couche transport (4) et qu'il concerne l'envoi d'une séquence valide de primitives (1).

Dans ce qui suit, seuls les scénarios relatifs au site client (RT) et au test d'une implémentation d'un protocole de transport seront cités avec pour chacun d'eux, une explication sommaire de leur objectif.

Pour certains d'entre eux, le scénario en entier sera donné, avec le scénario complémentaire à exécuter sur le centre de test.

Annexe 2 : Scenarios de test

4.1 Séquences valides de primitives

gestion d'une connexion (établissement & terminaison)

RT_4.1.100 - RT attend une connexion,
RT_4.1.101 - RT envoie une demande de connexion,
RT_4.1.102 - RT attend une connexion avec des données de l'utilisateur,
RT_4.1.103 - RT envoie une demande de connexion avec des données, attend la confirmation avec des données,
RT_4.1.104 - RT attend une demande de connexion, attend un disconnect,

log " --- rt_4.1.104 --- "
connect indication 2 1 0
connect response 0
listen 1
disconnect indication 0

log " --- tc_4.1.104 --- "
connect request 1 2 0
connect confirm 0
listen 1
disconnect request 0

Remarque : les nombres suivant les commandes représentent des offsets (ici 1 et 2) dans la table d'adresses réelle ainsi que la quantité de données transférée (ici 0)

RT_4.1.105 - RT attend des données dans une demande de connexion, attend un disconnect,
RT_4.1.106 - RT envoie des données dans une demande de connexion, attend un disconnect,
RT_4.1.107 - RT envoie une demande de connexion, un disconnect,
RT_4.1.108 - RT envoie des données dans une demande de connexion, un disconnect,
RT_4.1.109 - Collision de disconnect,
RT_4.1.110 - RT attend des données dans une demande de connexion, collision de disconnect,

Annexe 2 : Scénarios de test

RT_4.1.114 - RT attend une demande de connexion, envoie un disconnect avec des données,
RT_4.1.117 - RT envoie une demande de connexion, attend des données dans la réponse,
RT_4.1.118 - RT attend un disconnect,
RT_4.1.119 - RT attend un disconnect avec des données,
RT_4.1.120 - RT envoie un disconnect,
RT_4.1.121 - RT envoie un disconnect avec des données,

transfert de données (normales)

RT_4.1.200 - RT envoie des données normales,
RT_4.1.201 - RT attend des données normales,
RT_4.1.202 - RT envoie plusieurs demandes de transfert de données,
RT_4.1.203 - RT attend plusieurs indications de données,
RT_4.1.250 - RT envoie des données normales suivies d'un disconnect,
RT_4.1.251 - RT attend des données normales suivies d'un disconnect,
RT_4.1.252 - RT envoie plusieurs demandes de transfert de données, suivies d'un disconnect.

log " --- rt_4.1.252 --- "
connect indication 2 1 0
connect response 0
listen 1
data request 5000
data request 5000
data request 5000
data request 5000
data request 5000
disconnect indication 0

log " --- tc_4.1.252 --- "
connect request 1 2 0
connect confirm 0
data indication 5000
data indication 5000
data indication 5000
data indication 5000
data indication 5000

disconnect request 0

RT_4.1.253 - RT attend plusieurs indications de données, suivies d'un disconnect,

transfert de données (express)

RT_4.1.300 - RT envoie des données express,
RT_4.1.301 - RT attend des données express,
RT_4.1.302 - RT envoie plusieurs demandes de transfert de données,
RT_4.1.303 - RT attend plusieurs indications de données,
RT_4.1.350 - RT envoie des données express suivies d'un disconnect,
RT_4.1.351 - RT attend des données express suivies d'un disconnect,
RT_4.1.352 - RT envoie plusieurs demandes de transfert de données express suivies d'un disconnect,
RT_4.1.353 - RT attend plusieurs indications de données express, suivies d'un disconnect,

transfert de données (normales et express)

RT_4.1.400 - RT envoie des données normales et express,
RT_4.1.401 - RT envoie des données normales et express, envoie des données dans une demande de connexion,
RT_4.1.402 - RT attend des données normales et express
RT_4.1.403 - RT attend des données normales et express,attend des données dans une demande de connexion,
RT_4.1.404 - RT envoie des données normales, attend des données express,
RT_4.1.405 - RT envoie des données normales, attend des données express, envoie des données dans une demande de connexion,
RT_4.1.406 - RT attend des donnees normales, envoie des express,
RT_4.1.407 - RT attend des données normales, envoie des express et des données dans une demande de connexion,
RT_4.1.450 - RT envoie des données normales et express, puis un disconnect,
RT_4.1.451 - RT envoie des données normales et express, envoie des données dans une demande de connexion, puis un disconnect,
RT_4.1.452 - RT attend des données normales et express, puis un disconnect,

Annexe 2 : Scénarios de test

RT_4.1.453 - RT attend des données normales et express,attend des données dans une demande de connexion, puis un disconnect,
RT_4.1.454 - RT envoie des données normales, attend des données express, puis un disconnect,
RT_4.1.455 - RT envoie des données normales, attend des données express, envoie des données dans une demande de connexion, puis un disconnect,
RT_4.1.456 - RT attend des données normales, envoie des express, puis un disconnect,
RT_4.1.457 - RT attend des données normales, envoie des express et des données dans une demande de connexion, puis un disconnect,

4.2 Séquences invalides de primitives

demande de connexion (connect request)

RT_4.2.100 - RT envoie une demande de connexion après la réception d'une demande de connexion,
RT_4.2.101 - RT envoie deux demandes de connexion,
RT_4.2.102 - RT envoie une demande de connexion après qu'une connexion soit établie par le centre de test (TC),
RT_4.2.103 - RT envoie une demande de connexion après qu'une connexion soit établie par le site client (RT),
RT_4.2.108 - RT envoie une demande de connexion après une demande de libération (disconnect),

réponse (connect response) à une demande de connexion

RT_4.2.200 - RT envoie une réponse avant la réception d'une demande de connexion,
RT_4.2.201 - RT envoie une réponse après avoir envoyé une demande de connexion,
RT_4.2.202 - RT envoie deux réponses,
RT_4.2.203 - RT envoie une réponse après que la connexion soit établie,
RT_4.2.207 - RT envoie une réponse après avoir reçu un disconnect,
RT_4.2.206 - RT envoie une réponse après avoir envoyé un disconnect,

Annexe 2 : Scénarios de test

transfert de données (normales)

RT_4.2.300 - RT envoie des données normales avant une demande de connexion,
RT_4.2.301 - RT envoie des données normales avant une réponse à une demande de connexion,
RT_4.2.302 - RT envoie des données normales avant la réception de la confirmation de l'établissement de la connexion (connect confirm),
RT_4.2.305 - RT envoie des données normales sur une connexion venant d'être libérée,
RT_4.2.306 - RT envoie des données normales après avoir envoyé un disconnect,

transfert de données (normales)

RT_4.2.400 - RT envoie des données expresses avant une demande de connexion,
RT_4.2.401 - RT envoie des données expresses avant une réponse à une demande de connexion,
RT_4.2.402 - RT envoie des données expresses avant la réception de la confirmation de l'établissement de la connexion (connect confirm),
RT_4.2.405 - RT envoie des données expresses sur une connexion venant d'être libérée,
RT_4.2.406 - RT envoie des données expresses après avoir envoyé un disconnect,

demande de libération (disconnect)

RT_4.2.600 - RT envoie un disconnect avant une demande de connexion,
RT_4.2.601 - RT envoie un disconnect après que la connexion soit libérée,
RT_4.2.602 - RT envoie un disconnect après avoir reçu un disconnect,
RT-4.2.603 - RT envoie deux disconnect,

4.3 Modification des paramètres des primitives

Les variations de parametres concernent exclusivement la quantité de données (1, 16 ou 32 bytes) :

- acheminées au moyen des primitives suivantes : demande de connexion (connect request), réponse à une demande de connexion (connect response), transfert de données normales (data request) et expresses (expedited request), demande de libération de connexion (disconnect request),
- reçues au moyen des primitives suivantes : indication d'une demande de connexion (connect indication), confirmation de l'établissement d'une connexion (connect confirm), indication de la présence de données normales (data indication) et expresses (expedited indication), indication de libération d'une connexion (disconnect indication).

Le paramètre de quantité de données acheminées ou reçues, concernant les primitives de transfert de données normales et expresses spécifiera des valeurs allant de 250 à 10000 bytes.

4.4 Séquences invalides d'éléments de protocole

- CR_TPDU

RT_4.4.000 - L'Implémentation à Tester (IAT) reçoit un CR dupliqué,
RT_4.4.001 - L'IAT reçoit un CR après avoir envoyé un CC,
RT_4.4.002 - L'IAT reçoit un CR après avoir reçu un CC,
RT-4.4.003 - L'IAT reçoit un CR après avoir envoyé un DT,
RT_4.4.004 - L'IAT reçoit un CR après avoir reçu un DT,
RT_4.4.005 - L'IAT reçoit un CR après avoir envoyé un XPD (expedited data),
RT_4.4.006 - L'IAT reçoit un CR après avoir reçu un XPD,
RT_4.4.009 - L'IAT reçoit un CR apres avoir envoyé un DR,
RT_4.4.010 - L'IAT reçoit un CR après avoir reçu un DR,
RT_4.4.011 - L'IAT reçoit un CR après avoir envoyé un DC,
RT_4.4.012 - L'IAT reçoit un CR après avoir reçu un DC,

- CC_TPDU

RT_4.4.100 - L'IAT reçoit un CC avant qu'un CR ne soit produit,
RT_4.4.101 - L'IAT reçoit un CC après avoir reçu un CR,
RT_4.4.102 - L'IAT reçoit un CC après avoir envoyé un CC,
RT_4.4.103 - L'IAT reçoit un CC dupliqué,
RT_4.4.104 - L'IAT reçoit un CC après avoir envoyé un DT,
RT_4.4.105 - L'IAT reçoit un CC après avoir reçu un DT,

Annexe 2 : Scénarios de test

RT_4.4.106 - L'IAT reçoit un CC après avoir envoyé un XPD,
RT_4.4.107 - L'IAT reçoit un CC après avoir reçu un XPD,
RT_4.4.110 - L'IAT reçoit un CC après avoir envoyé un DR,
RT_4.4.111 - L'IAT reçoit un CC après avoir reçu un DR,
RT_4.4.112 - L'IAT reçoit un CC après avoir envoyé un DC,
RT_4.4.113 - L'IAT reçoit un CC après avoir reçu un DC,

- DR_TPDU

RT_4.4.200 - L'IAT reçoit un DR avant qu'un CR ne soit produit,
RT_4.4.201 - L'IAT reçoit un DR après avoir envoyé un DR,
RT_4.4.202 - L'IAT reçoit un DR dupliqué,
RT_4.4.203 - L'IAT reçoit un DR après avoir envoyé un DC,
RT_4.4.204 - L'IAT reçoit un DR après avoir reçu un DC,

- DC_TPDU

RT_4.4.300 - L'IAT reçoit un DC avant qu'un CR ne soit produit,
RT_4.4.301 - L'IAT reçoit un DC après avoir envoyé un CR,
RT_4.4.302 - L'IAT reçoit un DC après avoir reçu un CR,
RT_4.4.303 - L'IAT reçoit un DC après avoir envoyé un CC,
RT_4.4.304 - L'IAT reçoit un DC après avoir reçu un CC
RT_4.4.305 - L'IAT reçoit un DC après avoir envoyé un DT,
RT_4.4.306 - L'IAT reçoit un DC après avoir reçu un DT,
RT_4.4.307 - L'IAT reçoit un DC après avoir envoyé un XPD,
RT_4.4.308 - L'IAT reçoit un DC après avoir reçu un XPD,
RT_4.4.311 - L'IAT reçoit un DC apres avoir envoyé un DR,
RT_4.4.312 - L'IAT reçoit un DC apres avoir envoyé un DC,
RT_4.4.313 - L'IAT reçoit un DC dupliqué,

- DT_TPDU

RT_4.4.500 - L'IAT reçoit un DT avant qu'un CR ne soit produit,
RT_4.4.501 - L'IAT reçoit un DT après avoir envoyé un CR,
RT_4.4.502 - L'IAT reçoit un DT après avoir reçu un CR
RT_4.4.504 - L'IAT reçoit un DT après avoir envoyé un DR,
RT_4.4.505 - L'IAT reçoit un DT après avoir reçu un DR,
RT_4.4.506 - L'IAT reçoit un DT après avoir envoyé un DC,
RT_4.4.507 - L'IAT reçoit un DT après avoir reçu un DC,

Annexe 2 : Scénarios de test

- XPD_TPDU

RT_4.4.600 - L'IAT reçoit un XPD avant qu'un CR ne soit produit,
RT_4.4.601 - L'IAT reçoit un XPD après avoir envoyé un CR,
RT_4.4.602 - L'IAT reçoit un XPD après avoir reçu un CR,
RT_4.4.604 - L'IAT reçoit un XPD après avoir envoyé un DR,
RT_4.4.605 - L'IAT reçoit un XPD après avoir reçu un DR,
RT_4.4.606 - L'IAT reçoit un XPD après avoir envoyé un DC,
RT_4.4.607 - L'IAT reçoit un XPD après avoir reçu un DC,

- AK_TPDU

RT_4.4.700 - L'IAT reçoit un AK avant qu'un CR ne soit produit,
RT_4.4.701 - L'IAT reçoit un AK après avoir envoyé un CR,
RT_4.4.702 - L'IAT reçoit un AK après avoir reçu un CR,
RT_4.4.704 - L'IAT reçoit un AK après avoir envoyé un DR,
RT_4.4.705 - L'IAT reçoit un AK après avoir reçu un DR,
RT_4.4.706 - L'IAT reçoit un AK après avoir envoyé un DC,
RT_4.4.707 - L'IAT reçoit un AK après avoir reçu un DC,

4.5 Modifications des champs des éléments de protocole

Les principaux champs d'un élément de protocole sont :

- la longueur de celui-ci,
- le type (CR, CC, DR, etc ...)
- la valeur de crédit (pour les mécanismes de fenêtres),
- les références source et destination (pour identifier les deux entité d'une connexion),
- la classe choisie (pour le protocole de transport),
- la raison (pour un disconnect),
- le numéro de séquence ,
- l'indication de fin de transfert de données (pour des longs transferts nécessitant plusieurs TPDUs),
- la version du protocole choisie,
- les valeurs choisies pour les timers,

L'ensemble des tests entrant dans cette catégorie ont pour objectif d'affecter à ces divers champs des valeurs alternatives prévues dans les spécifications du protocole ou incorrectes (inexistantes dans les spécifications).

4.6 Tests relatifs aux timers

RT_4.6.201 - Supprimer un CR entrant,
RT_4.6.202 - Supprimer cinq CRs entrant,
RT_4.6.203 - Supprimer tous les CRs entrant,
RT_4.6.204 - Supprimer un CC entrant,
RT_4.6.205 - Supprimer cinq CCs entrant,
RT_4.6.206 - Supprimer tous les CCs entrant,
RT_4.6.207 - Supprimer un DT entrant,

log " --- rt_4.6.207 --- "
connect indication 2 1 0
connect response 0
data request 100
disconnect indication 0

log " --- tc_4.6.207 --- "
eg inlist "1 suppress DT"
connect request 1 2 0
connect confirm 0
data indication 100
disconnect request 0

RT_4.6.208 - Supprimer cinq DTs entrant dont les numéros de séquence sont identiques,
RT_4.6.209 - Supprimer tous les DTs entrant dont les numéros de séquence sont identiques,
RT_4.6.211- Supprimer cinq DTs entrant et possédant des numéros de séquence différents,
RT_4.6.212 - Supprimer tous les DTs entrant et possédant des numéros de séquence différents,
RT_4.6.216 - Supprimer un DR entrant,
RT_4.6.217 - Supprimer cinq DRs entrant,

log " --- rt_4.6.217 --- "
connect indication 2 1 0
connect response 0
listen 2
disconnect request 0

Annexe 2 : Scénarios de test

log " --- tc_4.6.217 --- "
eg inlist "5 suppress DR"
connect request 1 2 0
connect confirm 0
disconnect indication 0

RT_4.6.218 - Supprimer tous les DRs entrant,
RT_4.6.219 - Supprimer tous les AKs dans les deux directions,
RT_4.6.220 - Supprimer un XPD ,
RT_4.6.221 - Supprimer cinq XPDs,
RT_4.6.222 - Supprimer tous les XPDs,

ANNEXE 3

ALGORITHMES RELATIFS A LA GENERATION AUTOMATIQUE DE SEQUENCES DE TEST

Cette annexe décrit et commente trois algorithmes relatifs à la génération automatique de séquences de test (point 7.3) à savoir :

- **un algorithme de parcours des transitions**
- **un algorithme de construction de l'arbre de test**
- **un algorithme de construction d'une séquence de reconnaissance des états**

1. *Algorithme A : Parcours des transitions*

Objectif : parcourir chaque arc d'un graphe au moins une fois.

Considérons un graphe dont les noeuds constituent les états d'un système FSM et les arcs les transitions de ce système. On appelera chemin une séquence d'arcs tel que le noeud terminal d'un arc coïncide avec le noeud initial de l'arc suivant. Un chemin sera qualifié de simple s'il n'utilise pas deux fois le même arc. D_i^+ représentant le nombre d'arcs quittant un état E_i et D_i^- représentant le nombre d'arcs arrivant en un noeud E_i, on définira R, F, P et E comme :

R : { E_i appartenant à $E \mid D_i^+ = D_i^-$}

F : { E_i appartenant à $E \mid D_i^+ > D_i^-$}

P : { E_i appartenant à $E \mid D_i^+ < D_i^-$}

E : { ensemble de tous les états du système}

Si $D_i^- = D_i^+$ pour tout état du graphe, celui-ci est dit eulérien (R = E ; F et P vide). Dans ce cas, la théorie des graphes permet de montrer qu'il est possible de trouver un chemin parcourant une et une seule fois chaque arc. Dans les autres cas, le théorème suivant est d'application :

"Si un graphe orienté D n'est pas eulérien, un parcours de longueur minimale des transitions de D consiste en K chemins (chacun de ceux-ci joignant un noeud de F à un noeud de P) où

$K = \Sigma_F \ (D_i^+ - D_i^-) = \Sigma_P \ (D_i^- - D_i^+)$ "

Annexe 3 : algorithmes

Reprenant le graphe de la figure 7.1 représentant le protocole de transport classe 0, les valeurs des différentes variables sont les suivantes :

* E1 : $D_i^+ = 5$; $D_i^- = 15$;
* E2 : $D_i^+ = 6$; $D_i^- = 1$; $R = \{\}$;
* E3 : $D_i^+ = 3$; $D_i^- = 1$; $F = \{ E2, E3, E4 \}$
* E4 : $D_i^+ = 7$; $D_i^- = 4$; $P = \{ E1 \}$

* $K = 15 - 5 = (6 - 1) + (3 - 1) + (7 - 4) = 10$
  ==> 10 séquences de transfert dont 3 seront de longueur 2

Sachant qu'un isthme est un arc dont la suppression augmente le nombre de composantes d'un graphe, ceci permet de décrire l'algorithme suivant :

Etape 1 : Partir d'un état quelconque de F.

Etape 2 : Chaque fois qu'un arc est suivi, le supprimer.

Etape 3 : Ne pas utiliser un arc qui est un isthme, s'il en existe un autre qui peut être suivi.

Etape 4 : Quand on est bloqué, reprendre un autre état de F de telle manière que pour tout état, $D_i^+ - D_i^-$ départs soient réalisés

Algorithme A : parcours des transitions

Dans l'étape 4 se pose le problème du choix d'un nouvel état de départ. Désignons par V l'ensemble des noeuds qui sont, à un moment donné, origine d'un arc non parcouru. Lorsque l'on arrive en un état E_i qui ne permet plus d'aller plus loin (il n'existe aucun arc non parcouru dont E_i est origine), on doit considérer l'état le plus proche de E_i ; E_j appartenant à $(F \cap V)$. L'état le plus proche étant celui qui requiert le moins de transitions pour être atteint à partir de l'état courant E_i. En procédant de la sorte, on évite de choisir un état E_j si un autre état E_k appartenant à V est situé sur le chemin E_i --> E_j et on minimise la longueur de la séquence de transfert (= séquence d'événements nécessaire pour rejoindre l'état choisi).

2. *Algorithme B : Construction de l'arbre de test*

Objectif : construire un arbre de test qui sera utilisé pour la génération d'une partie de la séquence caractéristique.

Considérant un système modélisé selon la méthode FSM, l'arbre de test T peut être construit selon l'algorithme récursif suivant :

Etape 1 : associer à la racine de l'arbre T le label correspondant à l'état initial du système. On constitue ainsi le niveau 1 de l'arbre T.

Etape 2 : Supposant que l'on ait déjà construit l'arbre T jusqu'au niveau K, le niveau K+1 peut être obtenu en examinant les noeuds du niveau K de gauche à droite. Un noeud du niveau K est terminal si le label qui lui est associé est le même que le label associé à un noeud de niveau J (J<K) ou que celui d'un noeud situé plus à gauche. Sinon, si lors de la survenance d'un événement X, le système passe de l'état Ei à l'état Ej, on crée un nouvel arc et un nouveau noeud. Ceux-ci se voient associer respectivement les labels X et Ej dans l'arbre de test.

Algorithme B : construction de l'arbre de test

Le procédé décrit se termine toujours étant donné que par définition, le système est composé d'un nombre fini d'états. On peut constater que l'ordre de rangement des noeuds dans l'arbre est arbitraire (ici, le rangement s'est fait de gauche à droite) ; l'algorithme pourrait donc fournir différents arbres de tests (mais ceux-ci resteraient sémantiquement équivalents).

3. *Algorithme C : Construction de la séquence de reconnaissance des états*

Objectif : construire la séquence de reconnaissance des états de telle façon que la DS (Distinguish Sequence) soit appliquée deux fois à chaque état.

Sachant

- que l'état Ei est une **source** pour l'événement X si et seulement si il n'existe aucun état Ej tel que Ej --X-->Ei ;
- qu'un **(DS)-diagamme** est un diagramme de transition inter-états pour l'événement DS uniquement,

l'algorithme est basé sur les principes suivants :

- L'objectif de l'algorithme est d'appliquer deux fois la DS à chaque état.

- Si on atteint un état pour lequel on a déjà appliqué deux fois la DS, il est nécessaire de faire passer le système dans un autre état en utilisant une séquence de transfert (étape 7 ou 9). Etant donné qu'une source nécessite, par définition, une séquence de transfert pour être atteinte, on choisira comme état initial une source afin d'économiser une séquence de transfert (S'il n'existe aucune source, on devra néanmoins utiliser un autre état initial).

- L'algorithme consiste à appliquer la DS aussi longtemps que le système ne passe pas dans un état reconnu (auquel on a déjà appliqué deux fois la DS). Si le système passe dans un état reconnu, on applique une séquence de transfert pour rejoindre un autre état non reconnu.

- Il existe deux régles qui régissent l'application d'une séquence de transfert :

* l'état initial d'une séquence de transfert doit avoir été reconnu. Ceci est rendu possible par l'application de la DS (étape 5) lorsque le système passe dans un état déjà reconnu.
* l'état terminal d'une séquence de transfert doit être une source non encore reconnue s'il en existe une ou un état non encore reconnu sinon.

Annexe 3 : algorithmes

L'algorithme suivant est basé sur les principes venant d'être énoncés :

Etape 1 : S'il existe plusieurs DS pour le système considéré, choisir la plus courte. Tracer le (DS)-diagramme. Identifier les sources s'il en existe.

Etape 2 : Choisir une source comme état initial et opérer les transitions nécessaires pour rejoindre cet état. Si aucune source n'existe, choisir un état quelconque comme état initial.

Etape 3 : Appliquer la DS à l'état courant.

Etape 4 : Si l'état atteint suite à l'application de la DS est un état non encore reconnu, aller à l'étape 3. Sinon, aller à l'étape 5.

Etape 5 : Appliquer la DS à l'état courant.

Etape 6 : S'il existe des sources non encore reconnues, aller à l'étape 7. Sinon, aller à l'étape 8.

Etape 7 : Appliquer les transitions nécessaires pour rejoindre une source non reconnue à partir de l'état courant. Aller à l'étape 3.

Etape 8 : S'il existe encore des états non reconnus, aller à l'étape 9. Sinon, arrêter.

Etape 9 : Appliquer les transitions nécessaires pour rejoindre un état non encore reconnu à partir de l'état courant. Aller à l'étape 3.

Algorithme C : construction de la séquence de reconnaissance des états