ANNEXE
B

La spécification de protocole

LES MÉTHODES DE SPÉCIFICATION DE PROTOCOLES

Nous allons décrire trois modèles sur lesquels reposent les techniques de description formelle : les automates d'états finis, les réseaux de Pétri et l'ordonnancement temporel. Ensuite, deux langages normalisés seront décrits : Estelle et Lotos ; Estelle utilisant les automates et Lotos l'ordonnancement temporel. Ces langages de description de protocoles ont été développés par l'ISO. Ils possèdent tous les deux une sémantique et une syntaxe formelle.

La dernière partie de ce chapitre s'attachera d'une part à présenter une approche d'évaluation de ces deux langages, d'autre part à approfondir les comparaisons par un choix de critères plus sélectifs.

Les automates d'états finis

Un automate d'états finis est composé :

- d'un ensemble Q fini d'états,
- d'un état distingué q_0 appelé état initial,
- d'une partie F incluse dans Q, et composée d'états terminaux notés F,
- d'une fonction de transition S :
  $S : Q \times X \rightarrow Q$
  $(q,a) \rightarrow q' = S(q,a)$.

L'automate $A = (Q, q_0, F, S)$ est appliqué un alphabet.

Exemple

L'automate pour reconnaître l'ensemble des mots contenant au moins deux « a » sera représenté sous la forme décrite dans la figure B.1.

FIGURE B.1 • *Automate de reconnaissance*

avec X = {a, b}

l'état 1 : initial

l'état 3 : terminal

Lorsque l'on applique un tel automate aux mécanismes d'un protocole de communication, on omettra les états initial et terminal, car ils possèdent les mêmes fonctions que les états intermédiaires. De même, l'alphabet représentera l'arrivée d'un événement ou d'une interaction.

Dans le cas de machines à automates d'états finis étendus, deux notions nouvelles apparaissent : les notions de prédicats et d'actions. La notion de prédicat permet, pour chaque arc, une valuation par une condition. Le franchissement d'une transition donnée demandera une nouvelle condition en plus de l'arrivée d'un événement et d'un état de départ associé. La notion d'action introduit le fait qu'à chaque franchissement de transition, un traitement pourra être effectué.

En conclusion, un automate d'états finis à prédicats est un ensemble de transitions représentées par le quintuplet :

<interaction, état de départ, prédicat, état d'arrivée, action>.

Toutes ces extensions (prédicats et actions) ont l'avantage par rapport aux automates simples (sans prédicats ni actions) de réduire le nombre d'états, donc le volume des automates.

Les réseaux de Pétri

Les réseaux de Pétri (RdP) ont été introduits au début des années 60 par C.A. Pétri, puis développés au MIT autour de 1972. Ils permettent, en particulier, de modéliser et d'analyser des systèmes de processus concurrents. La spécification des protocoles de communication en RdP a mené à des recherches et des analyses très importantes.

Un RdP est composé de deux types d'objets : les places et les transitions. L'ensemble des places permet de représenter les états du système ; l'ensemble des transitions représente alors l'ensemble des événements dont l'occurrence provoque la modification de l'état du système. Les places jouent donc le rôle de variables d'états du système et sont à valeur entière. Ces valeurs entières sont représentées par autant de marques affectées à une place. L'état du système est alors associé à un marquage définissant pour toute place le nombre de marques qui lui sont affectées (ou bien qu'elle contient). À l'occurrence d'un événement correspond le franchissement d'une transition.

Un réseau de Pétri est un quadruple R = <P,T ; Pré, Post> où :

- P est un ensemble fini de places et est représenté graphiquement par des cercles ;
- T est un ensemble fini de transitions et est représenté graphiquement par des barres ;
- Pré : PxT $\rightarrow\!|$N est l'application d'incidence avant ;
- Post : PxT $\rightarrow\!|$N est l'application d'incidence arrière.

Exemple

Soit l'exemple décrit dans la figure B.2.

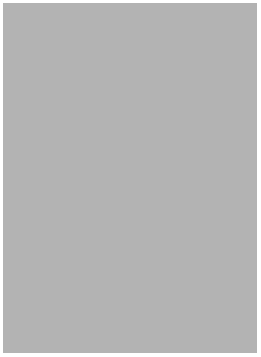

FIGURE B.2 • *Exemple de réseau de Petri*

• $P = \{P_1, P_2, P_3, P_4, P_5\}$ et $T = \{t_1, t_2\}$

Un marquage d'un réseau de Pétri est une application de P dans N. Si M est un marquage d'un RdP, M(P) est le marquage de la place P et est représenté par la présence de M(P) points ou marques à l'intérieur du cercle symbolisant la place P. Ces points sont aussi appelés jetons. Un réseau marqué est donc le couple :

• $N = <R ; M>$

• où R est un RdP et M un ensemble fini non vide de marques.

Un RdP peut être considéré comme un graphe orienté biparti dont les arcs sont valués : le graphe d'un réseau est un quadruplet :

• $G = <P, T ; \Upsilon, V>$

où :

• P est un ensemble fini de places,

• T est un ensemble fini de transitions.

• Υ est défini tel que :

$\forall p \in P, \quad \Upsilon(p) = \{t \in T \mid Pré(p,t) > 0\}$;

$\forall t \in T, \quad \Upsilon(t) = \{p \in P \mid Post(p,t) > 0\}$;

• V est la valuation de tous les arcs tel que :

$\forall p \in P, \forall t \in T, V(p,t) = Pré(p,t)$ et

$V(t,p) = Post(p,t)$.

Une transition t est dite déclenchable pour un marquage M si, et seulement si, pour tout p d'entrée : P(p) _ Pré(p,t), c'est-à-dire si toute place d'entrée de cette transition possède un nombre de marques au moins égal à la valeur de l'arc reliant chacune de ces places à la transition t.

Exemple

Soit l'exemple décrit dans la figure B.3 de graphe associé à un réseau.

FIGURE B.3 • *Exemple de graphe associé à un réseau*

Ainsi, t_1 n'est pas déclenchable tandis que t_2 l'est. En effet, dans t_1, le nombre de marques de la deuxième place est de 2 et non de 3.

Le déclenchement de la transition induit la soustraction de jetons aux places d'entrées de la transition. On enlève un nombre de jetons à la place p égal au nombre porté sur l'arc reliant la place à la transition. Lors du déclenchement de la transition, on ajoute dans chaque place de sortie un nombre de jetons égal à la valeur de l'arc reliant la transition à la place de sortie.

Exemple

L'ordre temporel est une notation et un calcul avec lequel on peut construire des expressions qui décrivent le séquencement des actions. Les séquences d'actions peuvent être vues comme des séquences temporelles d'événements. Les événements peuvent être identifiés avec des fonctions définies par les utilisateurs et l'état des variables.

FIGURE B.4 • *Exemple avant et après déclenchement*

La logique temporelle est en fait un outil de construction d'assertions et de preuves de théorèmes sur les séquences. Elle peut être utilisée pour spécifier et analyser des protocoles.

LE LANGAGE ESTELLE

Introduction

Paru en 1983, Estelle est devenu une norme. Ce langage est fondé sur le concept de machine d'états finis étendus. L'automate d'états finis est étendu par des variables, des paramètres et des priorités. Dans le principe, pour la description d'un protocole, l'état principal (celui de l'automate d'états finis sous-jacent) sert à décrire les phases du protocole, c'est-à-dire l'aspect contrôle, alors que l'aspect données sera représenté par des variables Pascal. Pour l'aspect contrôle, Estelle définit une syntaxe originale, tandis que pour l'aspect données il reprend intégralement le langage Pascal.

Le langage Estelle a deux objectifs principaux :

- le comportement du système ne doit pas être ambigu, ou alors ses ambiguïtés sont explicitées ;
- la spécification de chaque morceau du système doit être assez complète pour qu'il soit possible de la décrire isolément et l'intégrer ensuite au système complet.

Ainsi le langage apporte un soin particulier à la description d'interfaces (des canaux, des modules, des interactions), et toutes les finesses d'un comportement non déterministe (partie processus, avec des transitions, comportement des gardes variées).

Modules

Une spécification en Estelle est composée d'un ensemble de modules communicants, interconnectés par un mécanisme de transfert de message appelé canal (channel). L'objectif d'une spécification de modules est de définir le comportement de modules visibles au travers de ces points d'interaction. Le module est d'abord défini en tant que type, et dans un deuxième temps il sera instancié. La définition du type « module » comprend la définition de son en-tête (header) et celle de son corps (body) ; plusieurs corps peuvent être définis avec le même en-tête. Une instance de module n'existera qu'après l'exécution de la primitive INIT ; cette primitive crée une instance de module en associant l'en-tête du module à un de ses corps précédemment définis ; cette même instance cessera d'exister lorsqu'une primitive de destruction RELEASE sera appliquée.

Une spécification Estelle est organisée selon une structure hiérarchique où chaque module peut être structuré en sous-modules appelés modules fils. Deux classes de modules sont disponibles :

- les modules processus (process),
- les modules activités (activity).

Parmi les règles de structuration, on a :

- une activité qui ne peut pas être sous-structurée ;
- un processus qui peut être sous-structuré, soit en processus fils, soit en activités filles, mais pas les deux simultanément ;

- une instance de module qui ne peut jamais s'exécuter en parallèle avec ses instances filles ni avec aucune de ses instances descendantes ; une instance de module peut accéder aux variables exportées par ses instances filles ;
- des instances de processus qui sont filles de la même instance de module père peuvent s'exécuter en parallèle les unes par rapport aux autres dès qu'elles sont créées ; elles ne peuvent pas accéder mutuellement aux variables qu'elles exportent à l'instance de leur module père ;
- des instances d'activités qui sont filles de la même instance de module père ne peuvent jamais s'exécuter en parallèle les unes par rapport aux autres ; elles peuvent, par contre, partager des variables qu'elles exportent.

La figure B.5 décrit la structuration hiérarchique de la spécification.

Les instances de modules sont considérées comme étant au même niveau hiérarchique quand elles sont filles d'une même instance : elles sont ainsi connectées au même nœud de l'arbre hiérarchique des instances de modules.

FIGURE B.5 • *Hiérarchie des modules*

L'en-tête d'un module définit la visibilité externe de chaque instance de ce module en précisant :

- les points d'interaction permettant l'échange d'interactions avec l'environnement extérieur ;
- l'ensemble des variables exportées par le module vers son module père et vers ses activités sœurs si le module est lui-même une activité ;
- les paramètres passés à une instance de module lorsqu'elle est créée et qui sont utilisés dans son corps ;
- classe du module définie par les attributs process (si le module est de la classe processus) et activity (si le module est de la classe activité).

La forme générée de l'en-tête d'un module est :

```
module identificateur-type classe de module
  (liste de points d'interaction ;
    param liste de paramètres
        export liste des variables exportées)
```

(où chaque liste et mot-clé sont optionnels).

Le corps d'un module décrit le comportement interne du module au moyen de transitions. Estelle permet la description de plusieurs corps pour un même en-tête, permettant au module de posséder des comportements internes distincts pour une même visibilité externe.

La définition d'un corps de module comprend les parties optionnelles suivantes :

- la partie déclaration spécifie les constantes, les types, les en-têtes et les corps de modules fils, les points d'interaction internes, les procédures et fonctions et les variables locales ;
- la partie initialisation comporte un ensemble de transitions dont l'une sera exécutée lors de l'initialisation d'une instance de module ;
- la partie déclaration des transitions spécifie les transitions qui décrivent la machine à états finis correspondant au comportement du module ;
- la partie terminaison déclare la procédure à exécuter lorsqu'une instance de module est détruite.

Une transition commence par le mot-clé trans. Les transitions d'un module sont définies par la spécification d'un certain nombre de types de transitions. Chaque type de transition est caractérisé par des conditions de franchissement qui incluent :

- l 'état majeur courant (clause from) ;
- l'interaction d'entrée (clause when) ;
- le prédicat de franchissement (clause provided) ;
- la condition de délai (clause delay) ;
- la priorité (clause priority).

Une transition est franchie si les conditions de franchissement sont remplies. La transition qui ne possède pas une condition d'entrée (le cas de délai par exemple) s'appelle une transition spontanée. Le choix d'une transition est fait d'une manière non déterministe si les transitions sont de la même priorité.

L'opération effectuée par la transition inclut :

- le changement de l'état courant à l'état majeur prochain (clause to) ;
- l'action comportant des sorties.

Dans un modèle, l'exécution d'une transition est atomique (ininterruptible) si l'automate n'est pas dans un état défini et ne peut pas recevoir d'autres interactions (que celles qu'il est en train de traiter).

La clause from est une garde qui signifie que l'automate doit être dans l'état majeur indiqué (ESTAB) pour effectuer cette transition. La clause to indique l'état d'arrivée de la transition. La clause when indique la condition d'entrée qui doit être présente pour franchir la transition.

« Output » indique l'émission d'une interaction sur un point d'interaction. Output est suivi du nom du point d'interaction, suivi de paramètres effectifs (si des paramètres formels ont été déclarés pour cette interaction). Ici, DATA-request (B) est envoyé à travers N et TIMER-request à travers S.

À chaque niveau de la hiérarchie, une instance de module peut créer des sous-modules et détruire des sous-modules existants. Les primitives qui servent à effectuer ces actions sont :

- la primitive INIT permettant la création et l'initialisation de nouvelles instances du module ;

- la primitive RELEASE permettant la destruction des instances des modules dont le module fils est racine.

Les points d'interaction sont les interfaces auxquelles les instances de module peuvent envoyer des interactions et en recevoir. Ce sont des interfaces « full duplex » qui permettent l'échange asynchrone des interactions. Deux catégories de points d'interaction sont définies :

- les points d'interaction externes, qui permettent à une instance de module d'échanger des interactions avec son environnement externe, par exemple avec l'instance de son module père ou avec des instances de ses modules frères. Les points d'interaction externes sont déclarés dans la liste de points d'interaction inclus dans la définition de l'en-tête du module ;
- les points d'interaction internes qui permettent à une instance de module d'échanger des interactions avec son environnement interne, par exemple, avec elle-même ou avec des instances de ses modules fils. Les points d'interaction internes peuvent être définis dans la partie de déclaration des variables, incluse dans la définition du corps du module.

La définition d'un point d'interaction fait référence à trois attributs :

1. le type de canal qui lui est associé – les points d'interaction sont déclarés par rapport à un type de canal. Une définition de type de canal est introduite par le mot-clé channel ; elle comporte l'ensemble des interactions (et des paramètres qui leur sont associés) qui pourront être échangées à travers des points d'interaction déclarés par rapport à ce type de canal ;
2. le rôle que le point d'interaction joue – le concept de rôle sert à déterminer quelles sont parmi les interactions déclarées pour le canal celles qu'une instance de module peut envoyer ;
3. la « discipline de file » – elle définit si le point d'interaction sera associé à une file qui lui est propre (individual queue) ou bien s'il devra partager une file commune (common queue) avec d'autres points d'interaction de la même instance de module. Ces fils sont des FIFO de taille illimitée.

Établir les liens entre les instances d'un module pour qu'ils puissent communiquer est une opération qui est effectuée par le module père dans tous les cas, sauf si le module est connecté à lui-même.

Les primitives qui servent à établir et à défaire les liens entre les points d'interaction associés à différentes instances de module sont les suivantes :

- CONNECT permet de lier des points d'interaction (internes ou externes) d'instances de module de même niveau hiérarchique (de deux fils par exemple), ou de lier des points d'interaction internes d'une instance d'un de ces modules fils. Les points d'interaction liés par la primitive CONNECT doivent nécessairement correspondre au même type de canal et jouer des « rôles » opposés par rapport à ce type de canal ;
- DISCONNECT permet de défaire les liens qui ont été établis par la primitive CONNECT ;
- ATTACH permet de lier un point d'interaction d'un module avec celui de son fils. Ainsi, les interactions destinées au père seront présentées directement au fils. Ces deux points doivent avoir le même rôle par rapport au canal ;
- DETACH défait le lien établi par la primitive ATTACH.

LE LANGAGE LOTOS

Le langage Lotos a été développé par le sous-groupe C à l'ISO (ISO/SC21/WG16-1/FDT), entre 1981 et 1984. L'idée de base de LOTOS provient de la possibilité de décrire un système en définissant les relations temporelles entre événements externes observables du système. Le modèle formel de Lotos a deux composantes :

1. la première concerne la description des comportements et des interactions des processus, fondée sur l'algèbre de Milner : C.C.S. (A Calculus of Communicating Systems) modifié ;
2. la seconde composante concerne la description des structures de données et les expressions de valeurs. Elle provient de ACT-ONE, un langage de types de données abstraites.

Concepts de base de Lotos

Un processus Lotos est une entité abstraite capable d'accomplir des actions observables ou non observables et de communiquer avec d'autres processus dans son environnement. Une interaction peut se dérouler entre deux ou plusieurs processus chaque fois qu'ils sont prêts à exécuter la même action observable. Une action observable exécutée par un processus P peut être vue comme une manifestation du processus P à communiquer avec l'environnement offrant la même action.

Considérons l'abstraction de processus suivante :

MAX3 [in1, in2, in3, out] = noexit :
= (MAX2 [in1, in2, out'] || MAX2 [out', in3, out]) \ [out']

où est défini un processus appelé MAX3. Il est défini comme la composition parallèle (||) de deux instances de processus appelées MAX2. MAX2 est capable d'exécuter les actions (in1, in2, out'), (out', in3, out).

Les deux chaînes d'actions peuvent évoluer indépendamment, sauf l'action out' qui est partagée par les deux processus (synchronisation sur out'). L'action out' est rendue non observable à MAX3 grâce à l'opérateur « \[out'] ». Seules les actions in1, in2, in3 et out sont observables par MAX3. Donc, une interaction est considérée comme l'exécution d'une même action observable par un ensemble de processus. Une action observable est assimilée à un « point d'interaction », ou « porte » (gate), partagé par les processus au moment de l'interaction.

Commentaires

Le comportement du processus MAX3 se compose de séquences d'actions exécutées à ses portes externes in1, in2, in3, out (ses interactions potentielles avec l'environnement). À l'intérieur de la boîte, la porte out' est interne et non accessible de l'extérieur.

Tout processus Lotos est défini en termes de <behaviour Exp>. Syntaxiquement, une <behaviour Exp> est obtenue en appliquant un opérateur à d'autres <behaviour Exp>.

Un processus est défini par son comportement et ce dernier se compose de transitions. On voudrait avoir un mécanisme formel pour décrire toutes les transitions possibles. On utilisera pour cela la syntaxe de la <behaviour Exp> ; c'est l'approche par la sémantique opérationnelle. On applique les axiomes et les règles d'inférence de la sémantique opérationnelle.

On notera B1 α B2. On dira que B1 se transforme en B2 en exécutant « α ».

Une inaction sera notée : stop. Stop est un processus Lotos prédéfini qui est incapable de faire une action ou de communiquer.

FIGURE B.6 • *Représentation du processus MAX3*

COMPARAISON DES LANGAGES ESTELLE ET LOTOS

Un groupe de recherche hollandais a proposé une approche pour l'évaluation des techniques de descriptions formelles. L'approche se fonde sur le modèle de référence d'une entité OSI. Ce modèle va permettre d'observer et de comparer le comportement des FDT. Toute l'évaluation repose sur le fait que les FDT seront comparés en fonction de leur capacité de décrire une application conforme au modèle de référence et non d'après des phénomènes abstraits. Le modèle se compose de la façon suivante :

- P contient ce qui est visible d'une autre entité (N). C'est l'entité abstraite du protocole ;
- Q permet les échanges entre les entités N et N + 1 ;
- U permet les échanges entre les entités N et N-1 ;
- P, Q et U composent une entité complète de protocole.

Étude de la démarche conceptuelle d'une entité de protocole

Dans la conception d'un protocole, on distingue trois étapes :

1. la première description provient directement des documents normalisés en faisant référence aux échanges logiques entre l'entité et son environnement ;
2. la deuxième description (spécification) provient de la première en rajoutant des fonctions et des mécanismes nécessaires représentant les échanges réels rentrant ou sortant d'une entité ;
3. la troisième description est associée à l'implémentation et provient de la deuxième spécification en rajoutant des contraintes propres au système d'exploitation de la machine cible.

Pour chacune de ces trois étapes on peut dégager trois activités. La première et la deuxième étape incluent implicitement les activités suivantes :

- une spécification de l'architecture de l'entité du protocole,
- une spécification de conception comportementale de l'entité de protocole,
- une validation de l'entité de protocole.

La troisième étape contient :

- une implémentation,
- une implémentation complète,
- une validation.

À partir de ces neuf activités, on peut établir une table dans laquelle les colonnes indiquent les objets de description qui peuvent être représentés grâce à une FDT. Les lignes seront les aspects des objets de description tels que l'architecture, le comportement et la validation.

Soit la figure B.7 décrivant les paramètres.

FIGURE B.7 • *Description des paramètres*

Ainsi, l'entrée :

- I définit la structure décrite par les modules, ports, etc. ;
- II définit le comportement des modules indépendamment des aspects de gestion ;
- III validation par rapport aux contrôles des propriétés fonctionnelles spécifiques ;
- IV définit la structure fonctionnelle et l'intégration des aspects de gestion ;
- V définit le comportement des modules plus les aspects de gestion ;
- VI validation de cohérence par rapport au contrôle de sûreté et « deadlocks » ;
- VII installation des structures fonctionnelles décrites dans l'environnement cible incluant les choix des allocations et de configuration ;
- VIII installation des comportements des modules ;
- IX installation et simulation, tests et vérification ;

Nous venons de définir la table de classification des aspects et objets des descriptions. Chaque entrée de la table est un point d'observation. Dans une deuxième phase, il va falloir définir des critères d'évaluation permettant de comparer les comportements des FDT sur ces points d'observation. Ces critères d'évaluation sont les besoins auxquels doivent répondre les FDT.

Les besoins de descriptions formelles afin de classer les FDT

Ces critères vont permettre de classer les différentes FDT pour chaque case (entrée) de la table des descriptions. Chaque critère d'évaluation défini sera affecté à une case de la table. Nous construirons ainsi une nouvelle table appelée « table des besoins ». Soit la table des besoins décrite dans la figure B.8.

FIGURE B.8 • *La table des besoins*

La troisième et dernière phase dans l'évaluation va être de prendre une à une chaque

FDT et de regarder son comportement vis-à-vis de chaque critère d'évaluation appartenant aux neuf entrées de la table des besoins. Les résultats vont donner naissance à une nouvelle table appelée « table d'évaluation ».

FIGURE B.9 • *La table d'évaluation*

Les résultats sur les comportements sont binaires (oui/non), c'est-à-dire que l'on répondra par oui/non à la question : « Est-ce que tous les critères de l'entrée sont vérifiés pour la FDT étudiée ? »

Étude de cas

L'évaluation des langages Estelle et Lotos a été effectuée en prenant comme application (conforme au modèle de référence) le protocole Transport classe 2 (plus précisément la phase d'établissement de connexion). On peut, par exemple, dresser la table d'évaluation pour le critère « disponibilité ». La table d'évaluation du protocole transport pour les FDT Estelle et Lotos est présentée dans la figure B.9.

Observations

1. Seul Lotos est disponible pour la validation des spécifications abstraites et concrètes, car Lotos possède des opérateurs propres pour faire des preuves. Les autres n'ont pas ces outils de validation.
2. L'entrée IX indique un point d'intersection intéressant entre Estelle et Lotos : le test de l'implémentation peut être fait par simulation (Estelle) et par l'application de séquences de tests générés par Lotos.
3. Les trois FDT sont disponibles pour l'architecture des spécifications abstraites et

concrètes (entrées I, IV). Cela est dû à la capacité de ces FDT de décrire les éléments structurels.

4. Les trois FDT sont disponibles pour le comportement des spécifications abstraites et concrètes (entrée II,V) : c'est le but principal de toute FDT.

Un résumé des opérations effectuées sur les tables est présenté dans la figure B.10.

FIGURE B.10 • *Opérations effectuées sur les tables*

Une comparaison sélective des FDT

L'évaluation des langages FDT fondée sur un modèle de référence souligne les contraintes importantes auxquelles doit se plier un langage. Cette méthode, qui étudie le comportement des langages par le biais des descriptions d'un protocole donné, utilise des critères peu sélectifs. On remarque ainsi que les cases I, II, IV, V et XI de la table d'évaluation regroupent nos deux langages.

Nous rappelons les résultats dans la figure B.11 (pour le protocole transport).

FIGURE B.11 • *La table d'évaluation*

5. RÉFÉRENCES

Excellent article de base sur les méthodes formelles.

G.V. BOCHMANN, C. SUNSHINE – Formal methods in communication protocol design, *IEEE Transactions on Communications*, vol. 28, 4, pp. 624-631, avril 1980.

Livre donnant une très bonne introduction aux réseaux de Petri.

G.W. BRAMS – *Réseaux de Pétri*, Tome 1 : *Théorie et Analyse*, Masson, Paris, 1983.

Une extension de Lotos en y ajoutant des temporisateurs.

J.P. COURTIAT, M.S. de CAMARGO, D.E. SAIDOUNI – RT-Lotos : LOTOS temporisé pour la spécification de systèmes temps réel, *CFIP'93 Ingénierie des Protocoles*, R. Dessouli & G.V. Bochmann Éditeurs, Hermès 1993.

Article de base pour la représentation de protocoles par des machines à états finis.

A. DANTHINE – Protocol representation with finite state machine, *IEEE Transactions on Communications*, vol. 28, 4, pp. 632-644, avril 1980.

Article à la base de la normalisation par le CCITT.

G.J. DICKSON, P.E. de CHAZAL – Status of CCITT description techniques and application to protocol specification, *Proceedings of the IEEE*, vol. 71, 12, pp. 1346-1355, décembre 1983.

Description d'une technique avancée pour la validation de protocoles.

O. RAFIQ, L. CACCIARI – La validation réduite des protocoles de communication, *CFIP'93 Ingénierie des Protocoles*, R. Dessouli & G.V. Bochmann Éditeurs, Hermès, Paris, 1993.

Très bon état de l'art sur la spécification de protocoles.

H. RUDIN – An informal overview of formal protocol specification, *IEEE Communications Magazine*, vol. 23, 3, pp. 46-52, mars 1985.