REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE
ECOLE NATIONALE POLYTECHNIQUE

DEPARTEMENT D'ELECTRONIQUE

Ecole Doctorale « GENIE ELECTRIQUE »
Spécialité : Electronique
Option : Signal et communications

Mémoire de Magister

Présenté par : M. BOUDJEMA Mohammed
Ingénieur d'état en électronique
Ecole Nationale Polytechnique d'Alger

Sujet

Conception d'un agent SNMP sous forme d'IP CORE pour la supervision des SoC

Jury d'examen

Président	**M A.BELOUCHRANI**	**Professeur**	**ENP**
Rapporteur	**Mr R.SADOUN**	**Maitre Assistant**	**ENP**
Examinateurs	**Mme L. HAMAMI**	**Professeur**	**ENP**
	Mlle M.GUERTI	**Professeur**	**ENP**

Année universitaire : 2009 - 2010

الملخص

إنِ تعقيدات الأنظمة الإلكترونية على الرقائق (ما يصطلح عليه في التراجم الإنجليزية باسم : System on Chip « SoC ») يتطلب حتما نظاما لتسييرها. إن دراستنا هذه تندرج تحت هذه الفكرة، و من تم ما هي الأدوات و الوسائل التي يلزم تطبيقها لتجسيد هذا الهدف. استوحينا و استفدنا من الشبكات الحاسوبية لنقل الوسائل المستعملة لهذا الغرض.

قمنا بإنشاء وحدة قابلة لإعادة الاستعمال (ما يصطلح عليه في التراجم الإنجليزية باسم :Intellectual Property "IP")، مصممة على معالج MicroBlaze، من أجل إدماج منفذ SNMP لهذه الأنظمة. يستند هذا التطوير على تصميم مسبق بلغة SDL.

<u>الكلمات المفتاحية</u>: **MicroBlaze, LwIP, Co-design, SDL, SNMP, IP CORE, SoC.**

RÉSUMÉ

La complexité des SoCs induit impérativement la nécessite de leurs supervisions. C'est derrière cette idée que s'est inscrit notre développement, à savoir quels sont les outils et méthodes à mettre en œuvre pour atteindre cet objectif. Nous nous sommes inspirés des réseaux informatiques pour transposer les outils utilisés dans ce but.

Un IP core, architecturé autour d'un soft processeur, a été développé dans ce sens réalisant l'implémentation d'un agent SNMP pour un SoC. Ce développement s'est basé au préalable sur une modélisation SDL.

<u>MOTS CLÉS:</u> SoC, IP CORE, SNMP, SDL, Co-design, MicroBlaze, LwIP.

ABSTRACT

The complexity of SoCs induces imperatively the requirement of their supervision. Behind this idea was guided our development, know what tools and methods to implement to achieve this goal. We are inspired from computer networks to transpose the tools used for this purpose.

An IP core, designed around a soft processor, was developed in this way achieving the implementation of an SNMP agent for SoCs. This development was based on a prior SDL modeling.

<u>KEY WORDS:</u> SoC, IP CORE, SNMP, SDL, Co-design, MicroBlaze, LwIP.

Table des matières

Table des matières 1

Table des figures 4

Liste des tableaux 6

Nomenclature 7

Introduction générale 9

1 Le Protocole SNMP 12

Introduction . . . 12

1.1 SNMP . . . 13

1.1.1 Présentation générale . . . 13

1.1.2 Fonctionnement . . . 14

1.1.3 Les requêtes SNMP . . . 14

1.1.4 Les MIBs . . . 15

1.1.4.1 Structure de la SMI . . . 15

1.1.4.2 Structure de la MIB . . . 16

1.2 Les RFCs et les versions SNMP . . . 17

1.3 SNMPv1 et v2 . . . 18

1.3.1 Les faiblesses de SNMPv1 . . . 20

1.3.2 Les améliorations de SNMPv2c . . . 20

1.4 SNMP v3 . . . 21

1.4.1 User Security Module (USM) . . . 21

1.4.1.1 L'authentification . . . 21

1.4.1.2 Le cryptageLe recepteu . . . 22

1.4.1.3 L'estampillage de temps . . . 23

1.4.2 VACM (View Access Control Model) . . . 24

1.4.3 La trame de SNMPv3 . . . 24

1.4.3.1 Version SNMP . . . 24

1.4.3.2 Identificateur de message . . . 24

1.4.3.3 Taille maximale . . . 24

1.4.3.4 Drapeaux . . . 24

1.4.3.5 Le modèle de sécurité . . . 25

1.4.3.6 Les informations de sécurité 25
1.4.3.7 Les identificateurs de contextes 25
Conclusion 26

2 Modélisation SDL d'un agent SNMP 27
Introduction 27
2.1 Nécessité d'une modélisation formelle 28
2.2 Cas de SDL 28
2.3 Présentation du langage SDL 29
2.3.1 Principes de SDL 30
2.3.1.1 Structuration 30
2.3.1.2 Communication 30
2.3.1.3 Comportement 31
2.3.1.4 Les données 31
2.3.1.5 Concurrence, protection et synchronisation des données 32
2.3.1.6 Signalisation temporelle et priorités 32
2.4 Présentation de l'environnement de développement TauSDL 32
2.5 Modèle SDL d'un agent SNMPv1 33
2.5.1 Choix de la version 33
2.5.2 L'agent SNMPv1 34
2.5.2.1 Bloc « *CONFIGURATION* » 34
2.5.2.2 Bloc « *ENTREE* » 37
2.5.2.2.1 Utilisation d'ASN.1 dans la modélisation de l'agent SNMP . . 38
2.5.2.3 Bloc « *MIB* » 39
2.5.2.3.1 Fichier MIB.txt 40
2.5.2.4 Bloc « *SORTIE* » 42
2.5.2.5 Bloc « *INTERFACE* » 42
Conclusion 44

3 IP CORE Agent SNMP 45
Introduction 45
3.1 Le Co-design 46
3.2 Environnement de développement utilisé 47
3.2.1 Bibliothèque IP cores existante (MP, bus, E/S, Mémoires) 49
3.2.2 Génération d'IP core dans EDK 49
3.3 Architecture IP core (Agent SNMP) 50
3.3.1 Architecture hardware 50
3.3.1.1 Justification des constituants 50
3.3.1.2 Architecture hardware complète 51
3.3.1.2.1 MicroBlaze : 52
3.3.1.2.2 Bus PLB : 53
3.3.1.2.3 Bus LMB : 53
3.3.1.2.4 Xps_ethernetlite : 53
3.3.1.2.5 Xps_TIMER : 55
3.3.1.2.6 xps_INTC [33] : 55

3.3.1.2.7 Mdm : Module Debug : 55
3.3.1.2.8 xps_uartlite : 56
3.3.2 Architecture Software 56
3.3.2.1 Architecture de l'application 56
3.3.2.2 Pile protocolaire LwIP [35] 56
3.3.2.2.1 Architecture de la pile : 57
3.3.2.2.2 Le modèle de fonctionnement : 58
3.3.2.2.3 La couche émulateur système d'exploitation : 58
3.3.2.2.4 La gestion de buffer et de mémoire : 58
3.3.2.2.5 Le fonctionnement IP : 60
3.3.2.2.6 Le fonctionnement UDP : 60
3.3.2.3 Module SNMP 61
3.3.2.4 Initialisation du software 66
3.3.2.4.1 Application dans la BRAM : 66
3.3.2.4.2 Bootloop : 66
3.3.2.4.3 Bootloader : 67
3.4 Résultat de mise en œuvre 68
3.4.1 Première Partie 68
3.4.2 Deuxième partie 72
Conclusion 74

Conclusion générale 75

Bibliographie 76

A Symboles graphiques de SDL 79

B Description SDL de l'Agent_SNMP 82

Table des figures

1.1 Environnement de gestion SNMP . . . 13
1.2 Architecture de SNMP[11] . . . 15
1.3 Structure de la MIB II . . . 16
1.4 SNMP v1- N octets (variable) . . . 18
1.5 PDU –N octets (variable) . . . 18
1.6 Datagramme IP d'une requête SNMP . . . 19
1.7 Datagramme IP d'un TRAP SNMP . . . 19
1.8 Opération d'authentification . . . 22
1.9 Opération de cryptage . . . 23

2.1 Eléments syntaxiques du langage SDL graphique [20]. . . . 30
2.2 Exemple de modélisation SDL . . . 31
2.3 Le flot de conception dans TauSDL . . . 33
2.4 Système SDL de l'agent SNMP . . . 34
2.5 Bloc configuration . . . 35
2.6 Procédure "char_str_2_oct_str" . . . 36
2.7 Fichier snmpd.conf . . . 36
2.8 Datagramme IP d'un message SNMP . . . 37
2.9 Bloc d'ENTREE . . . 38
2.10 Bloc MIB . . . 39
2.11 Différentes réponses dans le modèle SNMP . . . 40
2.12 Structure des données de la MIB . . . 41
2.13 Bloc Sortie . . . 42
2.14 Bloc Interface . . . 43

3.1 Flot de conception d'un système sur silicium . . . 47
3.2 Flot de conception sous EDK . . . 48
3.3 Schéma bloc de la configuration hardware . . . 51
3.4 Bloc Diagramme détaillé . . . 51
3.5 L'architecture de MicroBlaze . . . 52
3.6 Emplacement de la sous-couche MAC dans le modèle OSI. . . . 54
3.7 Architecture complète de l'application . . . 56
3.8 Architecture de la pile LwIP . . . 57
3.9 Pbuf PBUF_RAM avec une mémoire contrôlée par le sous-système pbuf . . . 59

3.10 Une chaine *pbuf*, dont la premiere est de type ***PBUF_ RAM*** et la deuxième est de type ***PBUF ROM*** 59
3.11 Une chaine pbuf de type PBUF POOL 59
3.12 Structure udp_pcb 60
3.13 Le fonctionnement UDP 61
3.14 Génération d'un fichier exécutable sous SDL 62
3.15 Schéma d'environnement d'un système SDL 63
3.16 Structure du code généré 64
3.17 Application MicroBlaze avec un bootloader 67
3.18 Illustration de la communication établie entre un PC contenant le Manager SNMP et la carte Spartan-3E. 68
3.19 Capture d'écran de l'interface graphique du simulateur 69
3.20 Capture d'écran de l'interface graphique du simulateur 69
3.21 Trame équivalente 70
3.22 Trace de fonctionnement à travers le diagramme MSC 71
3.23 Processus du système SDL 72
3.24 Commande « get-request » à partir du manager SNMP 72
3.25 Capture des trames réseaux par l'outil « Wireshark » 73

A.1 Symboles graphiques de SDL (1) 80
A.2 Symboles graphiques de SDL (2) 81

B.1 Interface graphique de TuaSDL 83
B.2 Processus « CONF » 84
B.3 Procédure « char_str_2_oct_str » 85
B.4 Processus « VALIDATION » 86
B.5 Procédure « compare_IP » 87
B.6 Procédure « convert_IPOS_2_int » 87
B.7 Procédure « extraire_IP » 88
B.8 Processus « RECHERCHE » (page1/3) 89
B.9 Processus « RECHERCHE » (page2/3) 90
B.10 Processus « RECHERCHE » (page3/3) 90
B.11 Procédure « get_MIB » 91
B.12 Procédure « extraire » 92
B.13 Procédure « set_MIB » 93
B.14 Procédure « getnext_MIB » (page1/2) 94
B.15 Procédure « getnext_MIB » (page2/2) 94
B.16 Procédure « convert_OID_to_CharString » 95
B.17 Procédure « 10_puiss » 95
B.18 Processus « MesSortie_Proc » 96
B.19 Processus « P_INTERFACE » 97

Liste des tableaux

1.1 Type PDU 19
1.2 Réponses possibles pour « Error Status » 19

Nomenclature

API : Application Program Interface

ARP : Address Resolution Protocol

ASICs : Application Specific Integrated Circuits

ASN.1 : Abstract Syntax Notation 1

BER : Basic Encoding Rule

BOOTP : Bootstrap Protocol

DES : Data Encryption Standard

DHCP : Dynamic Host Configuration Protocol

DNS : Domain Name System

EDK : Embedded Development Kit

ELF : Executable and Linkable Format

FPGA : Field Programmable Gate Array

FTP : File Transfer Protocol

HDL : Hardware Description Language

IAB : Internet Activities Board

ICMP : Internet Control Message Protocol

IGMP : Internet Group Management Protocol

ISO : International Standard Organization

IP : Intellectual Property

IP : Internet Protocol

ITU : International Telecommunication Union

LOTOS : Language Of Temporal Ordering Specification

MIB : Management Information Base

MSC : Message Sequence Chart

NoC : Network on Chip

OSI : Open System Interconnection

OID : Object Identifier

PCB : Printed Circuit Board

PDU : Packet Data Unit

PAD : Process Activity Definition

QoS : Quality of Service

RFC : Request for Comments

RTL : Register Transfer Level

SDL : Specification and description Language

SDK : Software Development Kit

SNMP : Simple Network Management Protocol

SMI : Structure of Management Information

SoC : System on Chip

TCP : Transport Control Protocol

UDP : User Datagramme Protocol

UML : Unified Modelling Language

USM : User-based Security Model

VACM : View- based Access Control Model

VHDL : Vhsic Hardware Description Language

XPS : Xilinx Platform Studio

Introduction générale

Avec l'évolution incessante de la technologie des semi-conducteurs, il est désormais possible de placer des centaines de millions de transistors sur une seule puce de silicium [1].

Ainsi, un système complexe peut être réalisé. On peut imaginer intégrer sur un même circuit des dizaines de microprocesseurs pilotant des circuits annexes. On parlera alors de système sur puce ou SoC (System on Chip).

La spécificité des SoCs est qu'ils sont conçus pour une application particulière. Contrairement aux systèmes communs qui peuvent s'adapter à de très nombreuses et diverses tâches, les SoCs ciblent une tâche en particulier. La partie matérielle est optimisée pour ne contenir que les composants nécessaires à cela, et elle peut avoir des accélérateurs matériels pour une fonction de calcul spécifique (FFT, DCT, encryptage...) qui serait trop coûteuse en temps de calcul. La partie logicielle est écrite en tenant compte des spécificités particulières de la partie matérielle (vitesse d'exécution, disponibilité mémoire, accélérateurs matériels...). Cette spécialisation a un coût : pour chaque nouvel usage, il faut concevoir un nouveau SoC.

Par rapport à l'utilisation de composants sur circuits séparés et disposés sur une carte électronique, l'usage de SoC apporte de nombreux avantages. D'un point de vue physique, le SoC permet de réduire l'espace utilisé, ce qui est particulièrement recherché lors de la conception des appareils électroniques grand public (téléphone portable, lecteur DVD portable...). Les composants étant beaucoup plus proches les uns des autres, la performance du réseau de connexion inter-composant s'en trouve largement améliorée (car sa fréquence peut être augmentée). L'usage d'une seule puce permet aussi de réduire la consommation énergétique, ce qui est un point important dans tous les systèmes embarqués. Enfin, sur de gros volumes de fabrication, les SoC introduisent un coût de fabrication moindre que l'assemblage sur carte PCB[1]. En raison de ces avantages, on retrouve aujourd'hui les SoCs dans presque tous les systèmes embarqués : les appareils électroniques grand public, les radars, les appareils de contrôle dans les transports...

Les méthodes de conception ont continuellement évolué pour pouvoir gérer la complexité croissante des circuits. Une complexité devenue telle qu'il est impossible de continuer à les concevoir au niveau RTL[2]. L'abstraction permet de concevoir des circuits plus rapidement, ouvrant la voie à l'intégration d'un plus grand nombre de composants.

[1] PCB : Printed Circuit Board, synonyme de Circuit imprimé

[2] RTL : Register Transfer Level est une méthode de description des architectures microélectroniques. Dans la conception RTL, le comportement d'un circuit est défini en termes d'envois de signaux ou de transferts de données entre registres, et les opérations logiques effectuées sur ces signaux.

La complexité des SoCs induit impérativement leur monitoring ou leur supervision à l'échelle du composant ou du système. C'est derrière cette idée que s'inscrit notre développement à savoir quels sont les outils à utiliser pour atteindre cet objectif.

Nous nous sommes inspirés des réseaux informatiques. Le parallèle nous semble évident entre les SoCs et les réseaux cités.

L'ISO, a cerné cinq axes pour la supervision des réseaux informatiques :

- La gestion des anomalies (Fault Management). L'objectif de l'administration réseau est d'avoir un réseau opérationnel sans rupture de service (taux de disponibilité à 99,999 % par exemple soit quelques secondes d'indisponibilité par an), ce qui définit une certaine qualité de service (QoS) offerte par l'opérateur à l'abonné. On doit être en mesure de localiser le plus rapidement possible toute panne ou défaillance. Pour cela, on surveille les alarmes émises par le réseau, on localise un incident par un diagnostic des alarmes, on journalise les problèmes...
- La gestion de la configuration réseau (Configuration Management). Il convient de gérer la configuration matérielle et logicielle du réseau pour en optimiser l'utilisation. Il est important que chaque équipement, chaque compteur... soit parfaitement identifié de façon unique à l'aide d'un nom ou identificateur d'objet OID (Object Identifier).
- La gestion des performances (Performance Management). Il convient de contrôler à tout moment le réseau pour voir s'il est en mesure d'écouler le trafic pour lequel il a été conçu.
- La gestion de la sécurité (Security Management). On gère ici les contrôles d'accès au réseau, la confidentialité des données qui y transitent, leur intégrité et leur authentification.
- La gestion de la comptabilité (Accounting Management). L'objectif est de gérer la consommation réseau par abonné en vue d'établir une facture.

L'ISO a proposé dans les années 80 la norme CMIS/CMIP (Common Management Information Service ISO 9595, Common Management Information Protocol ISO 9596) comme protocole d'administration de réseau et définit un cadre général au niveau architecture (ISO 7498).

En parallèle, l'IAB (Internet Activities Board) approuve le protocole SNMP (Simple Network Management Protocol) comme solution à cours terme et CMOT (CMIP Over TCP) à plus long terme. Au début des années 90, SNMP, plus simple, devient alors standard de fait et est adopté par de nombreux constructeurs. C'est le protocole d'administration de réseau des réseaux IP mais aussi des réseaux des opérateurs comme pour les réseaux ATM !

Nous pensons que les cinq axes cernés par ISO peuvent être exploité non seulement pour les réseaux informatiques mais peuvent être étendus aux systèmes sur puce, et chaque système sur puce est considéré comme un nœud réseau, et de même, c'est le protocole SNMP qui va être utilisé.

Avec le protocole SNMP, chaque nœud de réseau est considéré comme un agent SNMP. D'où l'idée est de concevoir un IP[3] CORE (**IP_SNMP**) réutilisable qui va représenter un agent SNMP à l'intérieur du SoC.

[3]IP : Intellectual Property est un bloc fonctionnel également appelé coeur. Chaque IP dédié ou reconfigurable est composé de processeurs, de mémoires et/ou d'interfaces.

L'autre volet qui en découle est comment l'implémenter.

Plusieurs solutions sont envisagées :

– **Une description purement matérielle**

Cette méthode consiste à concevoir un IP CORE par un langage de description matériel bas niveau comme VHDL ou Verilog.

– **Une application SNMP soft autour d'un système d'exploitation :**

Cette méthode consiste à installer un paquet déjà conçu d'une application SNMP (exemple : Net-SNMP [2]) sur un système d'exploitation comme «uClinux» [3] le tout autour d'un processeur.

– **Description haut niveau et implémentation type standalone :**

Cette méthode consiste à implémenter un agent SNMP modélisé à l'aide d'un langage de description autour d'un microprocesseur dédié.

Comme le système que nous avons à décrire se trouve être assez complexe pour être modélisé à l'aide de langages de description matériel (VHDL ou Verilog), nous avons décidé d'élever le niveau d'abstraction, en utilisant un langage de spécification système et de surcroît formel. Mais les concepts que manipulent ce genre de langage ne sont pas aisément transposables sur ceux des langages de description de matériels, et donc un moyen de passage direct de ces spécifications vers un code VHDL par exemple, n'est pas à notre portée.

De ce fait, nous avons choisi d'utiliser l'autre approche pour la synthèse des spécifications système, autrement dit le passage par un processeur intégré (solution 2 et 3).

L'utilisation d'une application autonome (***standalone***) est plus performante que l'utilisation d'un système d'exploitation, ce qui nous a poussé à opter pour la troisième solution.

Notre développement sera décrit par trois chapitres :

Le premier traitera du protocole SNMP. On y retrouvera les éléments clefs pour la supervision des réseaux et par extension, des SoCs.

Le deuxième chapitre abordera la modélisation haut niveau du protocole ainsi choisi.

Le troisième chapitre abordera l'implémentation de notre IP CORE tenant compte de la modélisation précédemment introduite. On y trouvera les détails de l'implémentation et les résultats de sa mise en œuvre.

Chapitre 1

Le Protocole SNMP

Introduction

Les RFC (Request For Comments) sont un ensemble de documents qui font référence auprès de la Communauté Internet et qui décrivent, spécifient, standardisent et débattent de la majorité des normes, standards, technologies et protocoles liés à Internet et aux réseaux en général. Le Fonctionnement du SNMP est décrit en détails dans les RFCs : RFC 1157 [4], RFC 1155 [5], RFC 1441 [6], RFC 1452 [7], RFC 3411 [8], et RFC 3418 [9], et les descriptions qui suivent, ne font que les résumer.

1.1 SNMP

1.1.1 Présentation générale

SNMP (Simple Network Management Protocol) est le protocole de gestion de réseaux proposé par l'IETF[1] . Il est actuellement le protocole le plus utilisé pour la gestion des équipements de réseaux [10].

SNMP est un protocole relativement simple. Pourtant l'ensemble de ses fonctionnalités est suffisamment puissant pour permettre la gestion des réseaux hétérogènes complexes. Il est aussi utilisé pour la gestion à distance des applications : les bases de données, les serveurs, les logiciels, etc.

L'environnement de gestion SNMP est constitué de plusieurs composantes : la station de supervision, les éléments actifs du réseau, les variables MIB et un protocole. Les différentes composantes du protocole SNMP sont les suivantes (Figure 1.1).

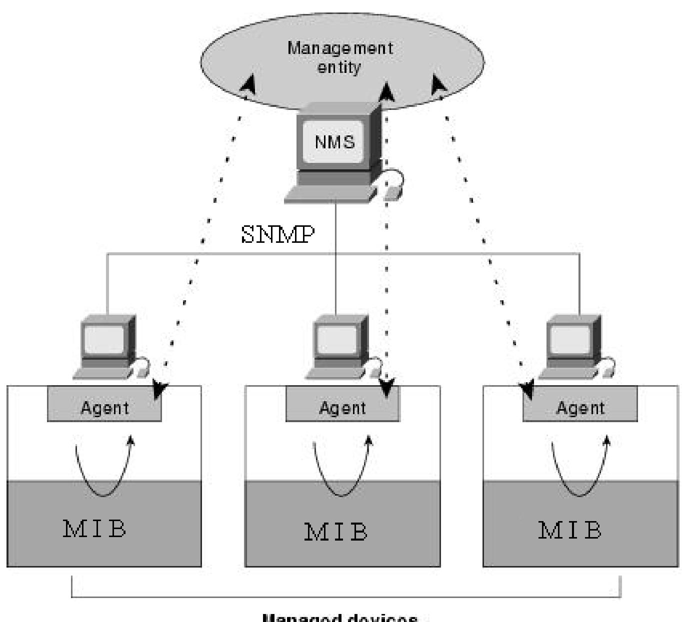

FIG. 1.1 – Environnement de gestion SNMP

- Les éléments actifs du réseau sont les équipements ou les logiciels que l'on cherche à gérer. Cela va d'une station de travail à un concentrateur, un routeur, un pont, etc. Chaque élément du réseau dispose d'une entité dite agent qui répond aux requêtes de la station de supervision. Les agents sont des modules qui résident dans les éléments réseau. Ils vont chercher l'information de gestion comme par exemple le nombre de paquets reçus ou transmis.
- La station de supervision (appelée aussi manager) exécute les applications de gestion qui contrôlent les éléments réseaux. Physiquement, la station est un poste de travail.
- La MIB (Management Information Base) est une collection d'objets résidant dans une base d'information virtuelle. Ces collections d'objets sont définies dans des modules MIB spécifiques.

[1] Internet Engineering Task Force

- Le protocole, qui permet à la station de supervision d'aller chercher les informations sur les éléments de réseaux et de recevoir des alertes provenant de ces mêmes éléments.

1.1.2 Fonctionnement

Le protocole SNMP est basé sur un fonctionnement asymétrique. Il est constitué d'un ensemble de requêtes, de réponses et d'un nombre limité d'alertes. Le manager envoie des requêtes à l'agent, lequel retourne des réponses. Lorsqu'un événement anormal surgit sur l'élément réseau, l'agent envoie une alerte (trap) au manager.

SNMP utilise le protocole UDP. Le port 161 est utilisé par l'agent pour recevoir les requêtes de la station de gestion. Le port 162 est réservé pour la station de gestion pour recevoir les alertes des agents.

1.1.3 Les requêtes SNMP

Cinq types de messages ou requêtes SNMP peuvent être échangés (SNMPv1) entre agent et manager (Figure 1.2) [3] :

- Obtention de la valeur courante d'un objet de la MIB géré par un agent : requête getrequest (GET).
- Obtention de la valeur courante du prochain objet de la MIB géré par un agent à partir d'un objet courant : requête get-next-request (GETNEXT).
- Mise à jour de la valeur courante d'un objet de la MIB géré par un agent : requête setrequest (SET).
- Renvoi de la valeur d'un objet de la MIB géré par un agent : requête get-response. C'est la réponse à un GET, GETNEXT ou SET.

Toutefois si la variable demandée n'est pas disponible, le GetResponse sera accompagné d'une erreur noSuchObject.

On voit que SNMP est un protocole de type commande/réponse sans états.

- Signal émis par un agent en direction d'un manager (pour remonter une alarme par exemple) : message trap (TRAP).

Les alertes (TRAPs) sont envoyées quand un événement non attendu se produit sur l'agent.

Les alertes possibles sont : 'ColdStart', 'WarmStart', 'LinkDown', 'LinkUp', 'AuthentificationFailure'.

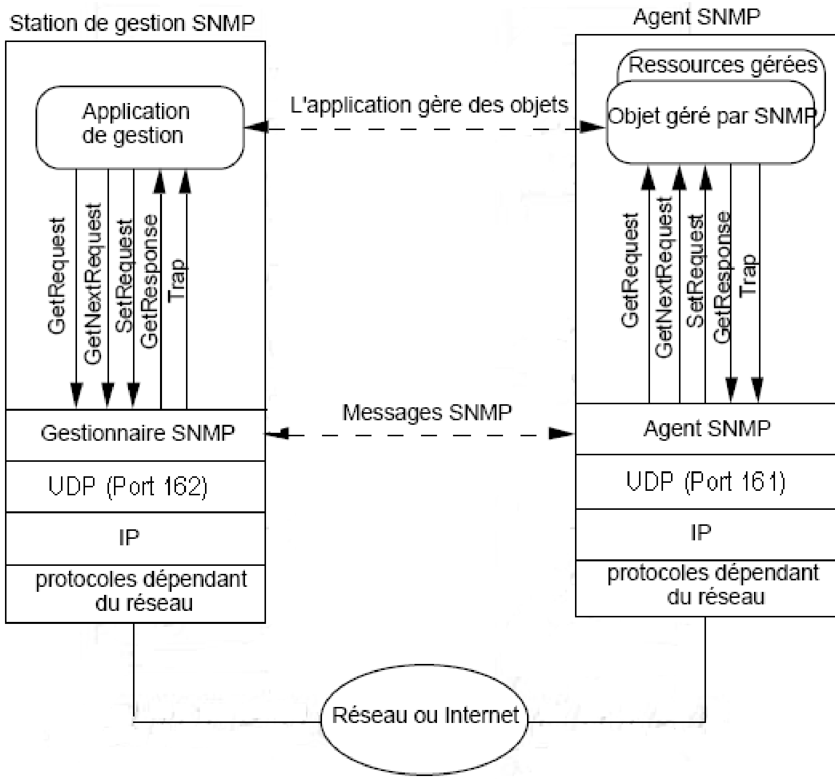

FIG. 1.2 – Architecture de SNMP[11]

1.1.4 Les MIBs

La MIB (Management Information base) est la basé de données des informations de gestion maintenue par l'agent, auprès de laquelle le manager va venir pour s'informer [12].

Deux MIB publics ont été normalisées : MIB I et MIB II (dite 1 et 2)

1.1.4.1 Structure de la SMI

La structure SMI (Structure of Management Information) décrit les règles de description de l'information et permet d'identifier de façon unique un objet de la MIB géré par un agent SNMP. Chaque objet possède donc un identificateur unique ou OID (Object ID).

SMI s'intéresse aussi à la représentation des données (et leur type) pour chaque objet de la MIB. Un objet de la MIB est déclaré et défini en langage ASN.1 (Abstract Syntax Notation 1 : langage de représentation de donnée).

SNMP n'utilise qu'une petite partie du langage ASN.1. Au niveau des types, seuls quelques uns sont utilisés comme [12] :

– INTEGER : valeur entière sur 32 bits en complément à 2.

- OCTET STRING : chaîne de caractères.
- IpAddress : adresse IP.
- PhysAddress : adresse MAC (6 octets pour un réseau de type Ethernet).
- Counter : entier de 32 bits non signé qui s'accroît de 0 à (2exp32 −1) puis revient à 0.
- TimeTicks : compteur de temps sur 32 bits non signé en 1/100 de s.
- . . .

La SMI est décrite dans la RFC 1155 [5].

1.1.4.2 Structure de la MIB

La MIB (Figure 1.3) est une structure arborescente dont chaque nœud est défini par un nombre ou OID (Object Identifier).

Elle contient une partie commune à tous les agents SNMP en général, une partie commune à tous les agents SNMP d'un même type de matériel et une partie spécifique à chaque constructeur. Chaque équipement à superviser possède sa propre MIB. Non seulement la structure est normalisée, mais également les appellations des diverses rubriques.

Ces appellations ne sont présentes que dans un souci de lisibilité. En réalité, chaque niveau de la hiérarchie est repéré par un index numérique et SNMP n'utilise que celui-ci pour y accéder.

Voici un exemple de structure de table MIB II [12] :

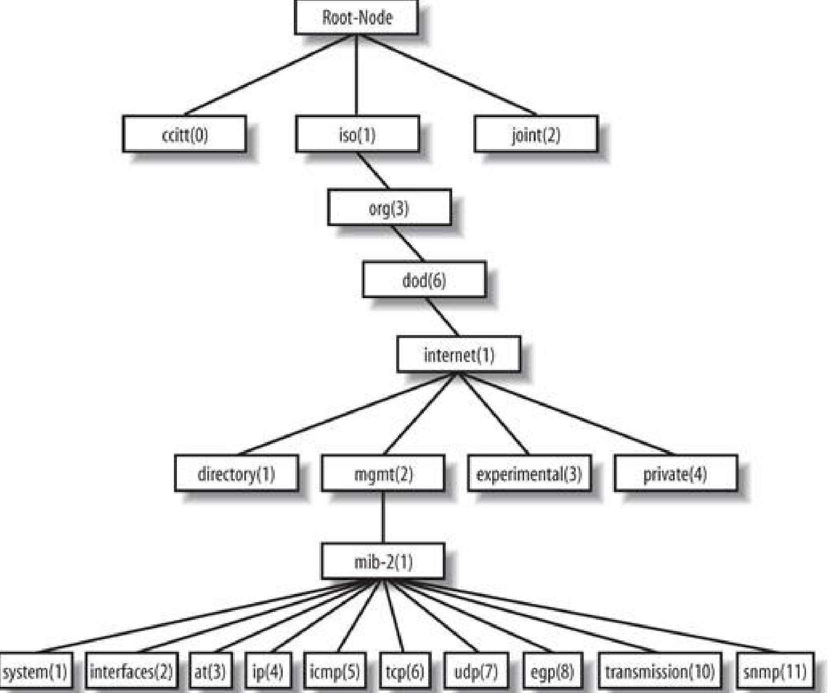

Fig. 1.3 – Structure de la MIB II

Ainsi, pour interroger les différentes variables d'activité sur un appareil, il faudra explorer son arborescence MIB. Celle-ci est généralement fournie par le constructeur mais il est aussi possible d'utiliser un explorateur de MIB tel que « Getif MIB Browser ».

Ensuite, pour accéder aux variables souhaitées, on utilisera l'OID (Object Identification) qui désigne l'emplacement de la variable à consulter dans la MIB.

Lorsqu'une entreprise veut définir son propre ensemble de variables de gestion, elle va enregistrer son numéro d'objet sous le nœud iso.org.dod.internet.private.entreprise. Ces MIB seront dites privées. Elles correspondent à la racine 1.3.6.1.4.1.

Un fichier MIB est un document texte écrit en langage ASN.1 (Abstract Syntax Notation One) qui décrit les variables, les tables et les alarmes gérées au sein d'une MIB.

1.2 Les RFCs et les versions SNMP

Le groupe de travail d'ingénierie de l'Internet IETF[2] , est responsable de définir les protocoles standards qui gèrent le trafic Internet, y compris le SNMP [13].

L'IETF publie des RFCs qui sont des spécifications pour beaucoup de protocoles qui existent dans le monde IP.

Les documents entrent dans les normes d'abord en tant que normes proposées ***proposed***, se déplacent alors au ***draft status***. Quand un projet définitif est par la suite approuvé, le ***RFC*** est donné le statut standard.

Deux autres désignations de norme, ***historique*** et ***expérimentale***, définissent (respectivement) un document qui a été remplacé par un nouveau RFC et un document qui n'est pas encore prêt à devenir une norme.

La liste suivante inclut toutes les versions courantes de SNMP et statut d'IETF de chacun [12] :

- La version 1 de SNMP (SNMPv1) est la version initiale du protocole SNMP. Elle est définie dans RFC 1157 et est une norme historique d'IETF. La sécurité de SNMPv1 est basée sur les communautés, qui ne sont rien d'autre que des mots de passe : suite d'octet qui permet a n'import quelle application basée sur SNMP qui connait cette suite d'octet d'accéder aux informations contenus dans MIB du dispositif. Il y a typiquement trois communautés dans SNMPv1 : read-only, read-write, et trap.

Il convient noter que puisque SNMPv1 est la version historique, c'est toujours l'implémentation primaire que beaucoup de fournisseurs soutiennent.

- SNMP version 2 (SNMPv2), Cette version de SNMP s'appelle techniquement SNMPv2c. Il est défini dans RFC 1441, RFC 1452.
- La version 3 (SNMPv3) c'est la dernière version de SNMP. Sa contribution principale à la gestion de réseau est la sécurité. Elle est améliorée pour une fiable authentification et communication privée entre les entités contrôlées. En 2002, elle a finalement fait la transition à partir du projet de norme à une norme finale. Les RFCs suivants définissent la norme : RFC 3410, RFC 3411, RFC 3412, RFC 3413, RFC 3414, RFC 3415, RFC 3416, RFC 3417, RFC 3418, et RFC 2576.

[2]Internet Engineering Task Force

Malgré que SNMPv3 soit une norme finale, les fournisseurs sont notamment lents à l'adopter. et malgré SNMPv1 a été transférer à une norme historique, la grande majorité du fournisseur qui réalisent du produites SNMP sont basée sur SNMPv1.Certains grands fournisseurs d'équipements de réseaux comme Cisco utilisent SNMPv3 dans les derniers temps, et ensuite d'autres fournisseurs adoptent SNMPv3 puisque les clients insistent sur des moyens plus sécurisées de gestion des réseaux.

1.3 SNMPv1 et v2

La trame SNMPv1 est complètement encodée en ASN.1 [ISO 87]. Les requêtes et les réponses ont le même format (Figure 1.4).

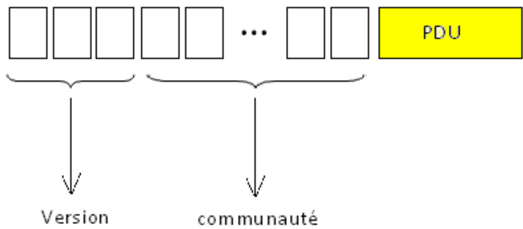

Fig. 1.4 – SNMP v1- N octets (variable)

- **La version** la plus utilisée est encore la version 1. Plusieurs versions 2 ont été proposées par des documents de travail, mais malheureusement, aucune d'entre elles n'a jamais été adoptée comme standard. La version 3 est actuellement en voie d'être adoptée. On place la valeur zéro dans le champ version pour SNMPv1, et la valeur 3 pour SNMPv3.
- **La communauté** permet de créer des domaines d'administration. La communauté est décrite par une chaîne de caractères. Par défaut, la communauté est « PUBLIC».
- **Le PDU** (Packet Data Unit).

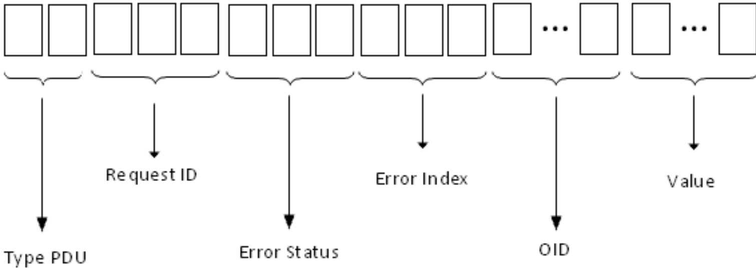

Fig. 1.5 – PDU –N octets (variable)

Le « ***PDU type*** » décrit le type de requête, de réponse ou d'alerte. Le Tableau 1.1 donne les valeurs associées à ces champs.

Type PDU	Nom
0	Get-request
1	Get-next-request
2	Set-request
3	Get-request
4	trap

TAB. 1.1 – Type PDU

Le « **Request ID** » permet à la station de gestion d'associer les réponses à ses requêtes.

Le « **Error Status** » est l'indicateur du type d'erreur. Si aucune erreur ne s'est produite, ce champ est mis à zéro. Les réponses négatives possibles sont décrites dans le tableau suivant :

Reponces	Déscription
NoAccess	Accès non permis
WrongLengh	Erreur de longeur
WrongValue	Valeur erronée
WrongType	type erronée
WrongEncoding	Erreur d'encodage
Nocreation	Objet non créé
ReadOnly	Pas de permission d'écrire
NoWritable	Pas de permission d'écrire
AutorisationError	Erreur d'autorisation

TAB. 1.2 – Réponses possibles pour « Error Status »

Donc La structure générale d'une requête SNMPv1 get-request, get-next-request, set-request et get-response est la suivante (Figure 1.6) :

entête IP 20 o.	entête UDP 8 o.	version	communauté	type PDU (0 à 3)	Id requête	statut erreur (0 à 5)	index erreur	nom (type)	longueur	valeur	T	L	V	…

FIG. 1.6 – Datagramme IP d'une requête SNMP

Et la structure générale d'un trap SNMPv1 est la suivante (Figure 1.7) :

entête IP 20 o.	entête UDP 8 o.	version	communauté	type PDU (4)	entreprise	adresse agent	type TRAP (0 à 5)	code spéc.	time stamping	T	L	V	…

FIG. 1.7 – Datagramme IP d'un TRAP SNMP

Deux remarques sont à faire :

– Le champ communauté (*community*) est un chaîne de caractères qu'il faut voir comme un mot de passe de validation d'une requête SNMP par l'agent (GET, GETNEXT, SET) ou par le manager (TRAP). Si la communauté est incorrecte, la requête est rejetée. La communauté passe en clair sur le réseau. C'est une faille importante de sécurité deSNMPv1. Il faut attendre les versions ultérieures de SNMP pour combler cette lacune.

– Les paramètres d'une requête sont encodés suivant le codage TLV (Type, Longueur, Valeur) ou BER (*Basic Encoding Rules*) défini dans SMI. Ce codage est assez classique en télécommunications et on le retrouve par exemple utilisé pour la signalisation Q.931 du réseau RNIS à l'interface Usager/Réseau. Il faut noter que ce système d'encodage est gourmand en octets car un paramètre sur 1 octet sera encodé en utilisant 3 octets (l'entier 12 est encodé comme $02 $01 $0C) ! On peut en revanche passer un nombre variable de paramètres par ce système d'encodage.

1.3.1 Les faiblesses de SNMPv1

Une des plus grandes faiblesses du protocole SNMPv1 est l'absence d'un mécanisme adéquat pour assurer la confidentialité et la sécurité des fonctions de gestion. Les faiblesses comprennent aussi l'authentification et le cryptage, en plus de l'absence d'un cadre administratif pour l'autorisation et le contrôle d'accès. Ce problème rend la sécurité sur SNMPv1 du type : "SHOW-AND-TELNET", c'est à dire qu'on utilise SNMP pour l'acquisition des données de gestion, mais pas pour effectuer le contrôle on utilise le protocole Telnet.

Le groupe de travail de l'IETF qui a œuvré sur SNMPv2 a voulu inclure la sécurité dans la nouvelle version. Malheureusement, ce groupe n'a pas pu atteindre un consensus sur le fonctionnement du mécanisme de sécurité. Partant de là, deux propositions ont été développées (SNMPv2u et SNMPv2*). La plupart des experts s'entendent pour dire que deux standards SNMP ne peuvent pas coexister, et que ceci n'est pas une solution à long terme.

Tous les consensus du groupe de travail ont été rassemblés (uniquement les améliorations qui ne portaient pas sur la sécurité), et le groupe de travail SNMPv2 de l'IETF a terminé ses travaux en publiant une version de SNMPv2 (on l'appelle SNMPv2c, RFC 1901, RFC 1905 et RFC 1906) sans sécurité.

1.3.2 Les améliorations de SNMPv2c

SNMPv2c a introduit quelques nouveaux types, mais sa nouveauté majeure est l'opération GET-BULK, qui permet à une plate forme de gestion, de demander en bloc de plusieurs variables consécutives dans la MIB de l'agent. Généralement, on demande autant de variables que l'on peut mettre dans un paquet SNMP. Ceci règle un problème majeur de performance dans SNMPv1. Avec la version 1, la plate forme est obligée de faire un GETNEXT et d'attendre la réponse pour chaque variable de gestion.

1.4 SNMP v3

Cette nouvelle version du protocole SNMP vise essentiellement à inclure la sécurité des transactions. La sécurité comprend l'identification des parties qui communiquent et l'assurance que la conversation soit privée, même si elle passe par un réseau public.

Cette sécurité est basée sur 2 concepts :

- USM (User-based Security Model)
- VACM (View- based Access Control Model)

1.4.1 User Security Module (USM)

Trois mécanismes sont utilisés. Chacun de ces mécanismes a pour but d'empêcher un type d'attaque.

- L'authentification : Empêche quelqu'un de changer le paquet SNMPv3 en cours de route et de valider le mot de passe de la personne qui transmet la requête.
- Le cryptage : Empêche quiconque de lire les informations de gestions contenues dans un paquet SNMPv3.
- L'estampillage du temps : Empêche la réutilisation d'un paquet SNMPv3 valide a déjà transmis par quelqu'un.

1.4.1.1 L'authentification

L'authentification a pour rôle d'assurer que le paquet reste inchangé pendant la transmission, et que le mot de passe est valide pour l'usager qui fait la requête.

Pour construire ce mécanisme, on doit avoir connaissance des fonctions de hachage à une seule direction. Des exemples de ces fonctions sont : MD5 et SHA-1. Ces fonctions prennent en entrée une chaîne de caractères de longueur indéfinie, et génèrent en sortie une chaîne d'octets de longueur finie (16 octets pour MD5, 20 octets pour SHA-1).

Pour authentifier l'information qui va être transmise, on doit aussi avoir un mot de passe qui est « partagé ». Le mot de passe ne doit donc être connu que par les deux entités qui s'envoient les messages, et préférablement par personne d'autre.

La figure ci dessous montre le mécanisme d'authentification :

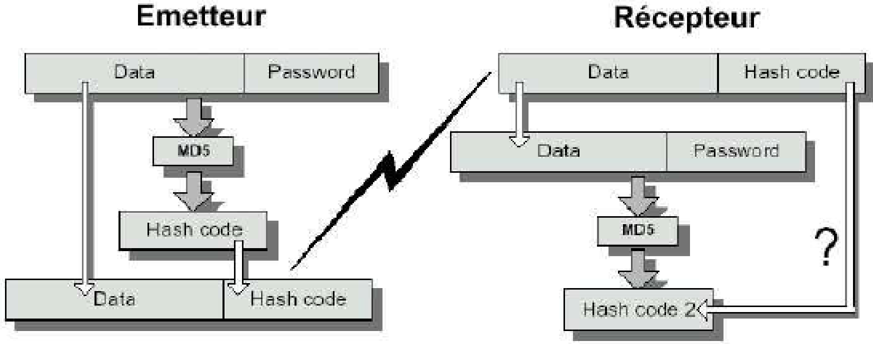

Fig. 1.8 – Opération d'authentification

Les étapes d'authentification sont les suivantes :

– Le transmetteur groupe des informations à transmettre avec le mot de passe.

– On passe ensuite ce groupe dans la fonction de hachage à une direction.

– Les données et le code de hachage sont ensuite transmis sur le réseau.

– Le récepteur prend le bloc des données, et y ajoute le mot de passe.

– On passe ce groupe dans la fonction de hachage à une direction.

– Si le code de hachage est identique à celui transmis, le transmetteur est authentifié.

Avec cette technique, le mot de passe est validé sans qu'il ait été transmis sur le réseau. Quelqu'un qui saisit les paquets SNMPv3 passants sur le réseau ne peut pas facilement trouver le mot de passe.

Pour ce qui est de SNMPv3, l'authentification se fait à l'aide de HMAC-MD5-96 ou de HMAC-SHA-96, qui est un peu plus compliqué que ce qui a été décrit ici. Le résultat de la fonction de hachage est placé dans le bloc paramètres de sécurité du paquet SNMPv3.

L'authentification se fait sur tout le paquet. L'étape d'authentification ne vise pas à cacher l'existence du paquet ou à le crypter. Si l'on utilise uniquement l'authentification, les personnes qui saisissent les paquets passants sur le réseau peuvent encore en voir le contenu. Toutefois, elles ne peuvent pas changer le contenu sans connaître le mot de passe.

1.4.1.2 Le cryptageLe recepteu

Le cryptage a pour but d'empêcher que quelqu'un n'obtienne les informations de gestion en écoutant sur le réseau les requêtes et les réponses de quelqu'un d'autre.

Avec SNMPv3, le cryptage de base se fait sur un mot de passe « partagé » entre le manager et l'agent. Ce mot de passe ne doit être connu par personne d'autre. Pour des raisons de sécurité, SNMPv3 utilise deux mots de passe : un pour l'authentification et un pour le cryptage. Ceci permet au système d'authentification et au système de cryptage d'être indépendants. Un de ces systèmes ne peut pas compromettre l'autre.

SNMPv3 se base sur DES (Data Encryption Standard) pour effectuer le cryptage.

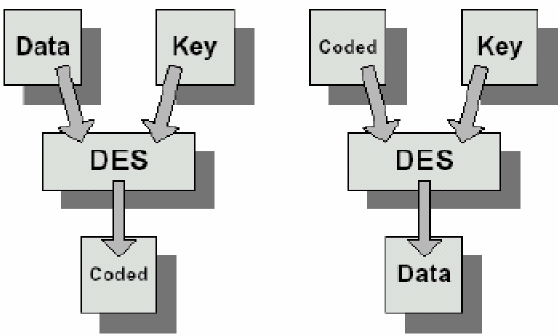

Fig. 1.9 – Opération de cryptage

On utilise une clé de 64 bits (8 des 64 bits sont des parités, la clé réelle est donc longue de 56 bits) et DES encrypte 64 bits à la fois. Comme les informations que l'on doit encrypter sont plus longues que 8 octets, on utilise du chaînage de blocs DES de 64 bits.

Une combinaison du mot de passe, d'une chaîne aléatoire et d'autres informations forme le « Vecteur d'initialisation ». Chacun des blocs de 64 bits est passé par DES et est chaîné avec le bloc précédent avec un XOR. Le premier bloc est chaîné par un XOR au vecteur d'initialisation. Le vecteur d'initialisation est transmis avec chaque paquet dans les « Paramètres de sécurité », un champ qui fait partie du paquet SNMPv3.

Contrairement à l'authentification qui est appliquée à tout le paquet, le cryptage est seulement appliqué sur le PDU.

1.4.1.3 L'estampillage de temps

Si une requête est transmise, les mécanismes d'authentification et de cryptage n'empêchent pas quelqu'un de saisir un paquet SNMPv3 valide du réseau et de tenter de le réutiliser plus tard, sans modification.

Par exemple, si l'administrateur effectue l'opération de remise à jours d'un équipement, quelqu'un peut saisir ce paquet et tenter de le retransmettre à l'équipement à chaque fois que cette personne désire faire une mise à jour illicite de l'équipement. Même si la personne n'a pas l'autorisation nécessaire, elle envoie un paquet, authentifié et encrypté correctement pour l'administration de l'équipement.

On appelle ce type d'attaques le « Replay Attack ». Pour éviter ceci, le temps est estampillé sur chaque paquet. Quand on reçoit un paquet SNMPv3, on compare le temps actuel avec le temps dans le paquet. Si la différence est plus que supérieur à 150 secondes, le paquet est ignoré.

SNMPv3 n'utilise pas l'heure normale. On utilise plutôt une horloge différente dans chaque agent. Ceux-ci gardent en mémoire le nombre de secondes écoulées depuis que l'agent a été mis en circuit. Ils gardent également un compteur pour connaître le nombre de fois où l'équipement a été mis en fonctionnement. On appelle ces compteurs BOOTS (Nombre de fois ou l'équipement a été allumé) et TIME (Nombre de secondes depuis la dernière fois que l'équipement a été mis en fonctionnement).

La combinaison du BOOTS et du TIME donne une valeur qui augmente toujours, et qui peut être utilisée pour l'estampillage. Comme chaque agent a sa propre valeur du BOOTS/TIME, la plate-forme de gestion doit garder une horloge qui doit être synchronisée pour chaque agent qu'elle contacte. Au moment du contact initial, la plateforme obtient la valeur du BOOTS/TIME de l'agent et synchronise une horloge distincte.

1.4.2 VACM (View Access Control Model)

Permet le contrôle d'accès au MIB. Ainsi on a la possibilité de restreindre l'accès en lecture et/ou écriture pour un groupe ou par utilisateur.

1.4.3 La trame de SNMPv3

Le format de la trame SNMPv3 est très différent du format de SNMPv1. Ils sont toutefois codés tous deux dans le format ASN.1 [ISO 87]. Ceci assure la compatibilité des types de données entres les ordinateurs d'architectures différentes.

Pour rendre plus facile la distinction entre les versions, le numéro de la version SNMP est placé tout au début du paquet. Toutefois, le contenu de chaque champ varie selon la situation. Selon que l'on envoie une requête, une réponse, ou un message d'erreur, les informations placées dans le paquet respectent des règles bien définies dans le standard. Voici comment les champs d'un paquet SNMPv3 sont remplis :

1.4.3.1 Version SNMP

Pour SNMPv3, on place la valeur 3 dans ce champ. On place 0 pour un paquet SNMPv1.

1.4.3.2 Identificateur de message

Ce champ est laissé à la discrétion du moteur SNMP. On retrouve souvent des algorithmes, où le premier message de requête est envoyé avec un nombre aléatoire et les suivants avec les incréments de 1. Les paquets qui sont émis en réponse à une requête portent la même identification que le paquet de la requête.

1.4.3.3 Taille maximale

Le moteur choisit la taille maximale d'une réponse à une requête selon ses capacités en mémoire tampon et ses limites à décoder de longs paquets. Quand on envoie une réponse à une requête, on doit veiller à ne pas dépasser la taille maximale.

1.4.3.4 Drapeaux

Trois bits sont utilisés pour indiquer :

- Si une réponse est attendue à la réception de ce paquet. (Reportable Flag)
- Si un modèle de cryptage a été utilisé (Privacy Flag)
- Si un modèle d'authentification a été utilisé (Authentification Flag)

1.4.3.5 Le modèle de sécurité

Ce module identifie le type de sécurité qui est utilisé pour encrypter le reste du paquet. Cet identificateur doit identifier de façon unique chaque module de sécurité. Actuellement, l'algorithme de cryptage DES (Data Encryption Standard) et l'algorithme d'authentification HMAC-MD5-96 ont été choisis comme algorithmes utilisés dans SNMPv3. HMAC-SHA-96 est optionnel.

1.4.3.6 Les informations de sécurité

Ces informations ne sont pas décrites dans le standard SNMPv3, ce bloc est laissé au soin des modules de sécurité. D'un module de sécurité à un autre, ces informations seront différentes. Le module de sécurité DES a standardisé le contenu de ce bloc.

1.4.3.7 Les identificateurs de contextes

Avec SNMPv1, il était possible d'avoir une seule base d'informations (MIB) par agent. Ceci n'est pas suffisant pour certains équipements qui peuvent contenir plusieurs fois les mêmes variables. Par exemple, un commutateur ATM contient plusieurs cartes qui ont chacune leurs propres bases d'informations. Le contexte permet de distinguer entre plusieurs bases d'informations et même plusieurs agents. On distingue entre des agents, par exemple, quand on a un moteur qui agit comme passerelle entre SNMPv3 et du SNMPv1. On envoie donc un paquet SNMPv3 en identifiant à quel agent SNMPv1 on désire que le paquet soit retransmis.

Conclusion

On soulignera le manque de sécurité évident qui subsiste sur les premières versions de SNMP (v1 et v2). C'est dans ce but qu'a donc été développée la dernière version (v3) de SNMP. Depuis 2002, celle-ci a été décrétée comme standard pour ce protocole. Pourtant la version 1 reste encore beaucoup utilisée et peu d'entreprises évoluent en passant à la version 3.

On s'intéressera dans notre cas à la version 1 pour des raisons qui vont être précisées dans le chapitre suivant.

Chapitre 2

Modélisation SDL d'un agent SNMP

Introduction

Comme décrit dans l'introduction, nous nous intéressons au développement d'un IP CORE dédié à la supervision d'un SoC. Pour ce faire, la recherche d'une méthodologie adaptée est plus que nécessaire. Nous avons opté au co-design dans lequel la partie software va être déduite d'une modélisation préalable conduisant à un niveau d'abstraction élevé. L'intérêt est multiple comme il va être expliqué dans le présent chapitre. On parlera aussi des outils adaptés à cet effet.

2.1 Nécessité d'une modélisation formelle

Une description formelle est une description claire et parfaitement précise. Elle permet d'obtenir des spécifications système prouvées et des programmes prouvés conformes aux exigences fonctionnels.

Les langages formels reposent sur des fondements mathématiques qui permettent d'effectuer des vérifications formelles sur les systèmes décrits. Différentes approches ont été développées, parmi elles, les automates d'états finis et les types de données abstraites [14].

Il existe deux classes de méthodes formelles :

La première traite de spécifications orientées propriétés, qui sont des descriptions données dans un langage permettant d'énoncer les propriétés attendues d'un système.

La seconde traite de spécifications orientées modèles ; c'est à dire la construction d'un système à partir d'objets fondamentaux préétablis.

Les raisons pour lesquelles les méthodes formelles sont utilisées sont les suivantes :

- Elles permettent un approfondissement de la connaissance du cahier des charges du système à développer ;
- Elles permettent de décrire ce que l'on veut faire sans expliquer comment le faire ;
- Pour différer l'étude des aspects algorithmiques ;
- Pour exprimer complètement un problème ;
- Pour diminuer la complexité ;
- Pour avoir un contrôle sur le processus de développement (qualité, fiabilité et la découverte d'erreurs) ;
- Elles ont souvent l'avantage d'être associées à des outils qui les implémentent.

Une spécification formelle se concentre sur les propriétés du système décrit plutôt que sur les détails d'implémentation. Cette spécification formelle sert de base pour une réalisation. Elle doit donc faire abstraction des détails d'implémentation afin de retarder les décisions concernant la réalisation et de permettre de n'exclure aucune implémentation valide.

Nous pouvons citer comme exemples de langages de description formelle : Le langage Z, le langage B, ESTEREL, LUSTRE, SDL (Specification and Description Language), UML (Unified Modeling Language), LOTOS (Language Of Temporal Ordering Specification), ESTELLE. . . etc

2.2 Cas de SDL

Un langage spécifique est choisi en fonction du domaine d'application, de son pouvoir d'expression, et de la disponibilité de méthodes et d'outils autour de ce langage.

Dans notre cas, les langages les mieux adaptés sont ceux dédiés à la spécification de protocoles de communication et aux systèmes distribués, c'est-à-dire : LOTOS, ESTELLE, SDL [14] et UML. Aussi,

SDL et UML présentent un avantage sur LOTOS et ESTELLE, celui de la richesse d'expression ainsi que la facilité de compréhension et d'apprentissage.

L'UML se situe en réalité, à un niveau d'abstraction plus élevé que SDL et ses principes de modélisation sont applicables à n'importe quel concept. En contrepartie l'UML n'est pas assez précis pour décrire de manière détaillée les applications. De ce fait, on l'utilise souvent dans les phases en amont d'un développement pour les opérations d'analyse et de spécifications système, mais rarement durant les opérations de spécifications détaillées et de conception. Et par conséquent, un autre langage est rapidement nécessaire pour enrichir le modèle décrit (avec du C ou SDL) [15].

Pour toutes ces raisons, SDL se présente comme étant le mieux adapté et le plus pratique, pour nos spécifications système et la modélisation du protocole SNMP.

Il présente également, plusieurs spécificités avantageuses telles que le fait que ce soit un langage normalisé, possédant une riche grammaire. SDL permet des spécifications textuelles mais également graphiques, et ses diagrammes sont faciles à comprendre.

Une autre caractéristique d'SDL est qu'il est un langage Orienté Objet, supportant des encapsulations et des polymorphismes, étendant le concept de classe de données Orientées Objet pour des applications techniques et des objets actifs (exemples : systems, blocks. . . etc).

Les outils qui supportent SDL permettent le passage vers des langages de niveau inférieur comme C/C ++. Cela signifie que le système SDL peut être traduit en une application exécutable.

Pendant le processus de modélisation une description SDL peut être rapidement vérifiée. Ce qui permet la correction des erreurs à un niveau très avancé.

2.3 Présentation du langage SDL

Le langage SDL (Specification and Description Language) est un langage formel et normalisé (ITU-T) particulièrement utilisé dans le domaine des télécommunications dédié à la description des systèmes discrets [16]. C'est un langage de haut niveau qui peut être utilisé en mode graphique (Figure 2.1) ou en mode textuel. SDL est basé sur les automates à états finis étendus communicants et les types abstraits de données. Il inclut les notions de typage et de généricité. Sa sémantique étant définie formellement, il est possible de simuler et de valider une application décrite en SDL. Il est aussi possible de générer le code correspondant pour des exécutifs temps réel du marché. En complément, le langage MSC (Message Sequence Chart) [17], normalisé par l'ITU-T, assure la description des diagrammes de séquence associés à SDL.

D'un point de vue historique, une première version informelle de SDL a été proposée en 1976. Depuis, une nouvelle version est produite tous les 4 ans. Le premier standard formel est arrivé en 1988. Les versions 92 et 96 [18] sont assez proches et sont à la base de cette présentation. Enfin, la version 2000 [19], non encore outillée, introduit des concepts Orienté Objet (OO) au sein de SDL.

Au niveau des outils, ObjectGéodeTM de Verilog$^{(R)}$ et TauTM de Telelogic$^{(R)}$ ont pour objectif d'assurer une continuité dans la couverture du cycle de développement [20]. Ces outils s'appuient sur SDL et les MSC. Ils offrent des possibilités d'édition, de simulation, de test et de génération de code.

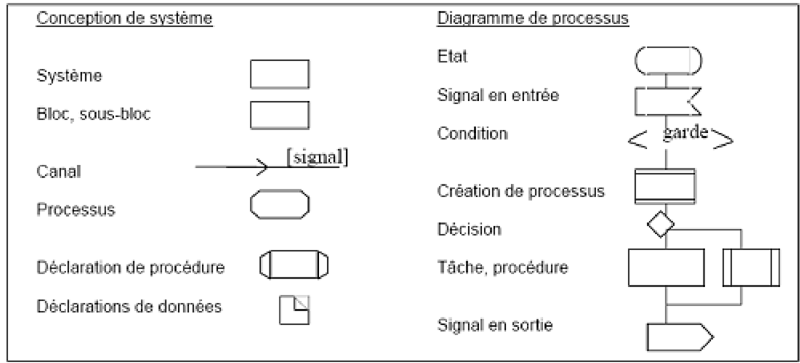

Fig. 2.1 – Eléments syntaxiques du langage SDL graphique [20].

2.3.1 Principes de SDL

2.3.1.1 Structuration

En SDL, dans un premier niveau, le système modélisé est composé de blocs liés les uns aux autres au moyen de canaux. Les blocs sont composés de sous-bloc et de processus. Un processus, au sens SDL, est une machine à états finis communicante [21]. Les instances de processus sont créées statiquement ou dynamiquement. Chaque instance possède son identifiant ou PID (Processus Identifier). Pour exemple, le système présenté à la Figure 2.2 il est composé de trois blocs : Capteur, Regulation, Actionneur. Ces blocs sont reliés par les canaux can1 et can2. Le canal can3 permet de relier le bloc Actionneur à l'environnement du système. Enfin, le bloc Regulation est lui-même composé des processus ICapteur, Element et IActionneur.

2.3.1.2 Communication

Les processus communiquent via des signaux de manière asynchrone (pas d'attente de l'émetteur). Les signaux sont véhiculés par les canaux. Quand un signal est émis, il est transmis, via son canal, au bloc de destination. Sur un canal, l'ordre d'émission entre signaux est conservé. Par contre, si deux signaux arrivent en même temps ils sont traités arbitrairement. Quand un signal arrive sur un processus, il est mis en attente dans une file gérée selon le principe FIFO (premier arrivé/premier servi). Si le destinataire du signal n'est pas explicité via son PID, le signal est consommé par la première des instances du processus pouvant le consommer. Dans l'exemple présenté à la Figure 2.2, par exemple, le signal IT est transmis du bloc Capteur vers le bloc Regulation via le canal can1. Il est ensuite consommé par le processus Icapteur. Ce dernier produit à son tour le signal Mesure.

Le deuxième mode de communication proposé par SDL est un mode synchrone (attente de l'émetteur) basé sur l'appel distant de procédure. Il est possible de déclarer une procédure dans un processus et de l'appeler dans un processus distinct. L'appelé considère l'appel comme la réception d'un signal implicite déclenchant l'appel à la procédure.

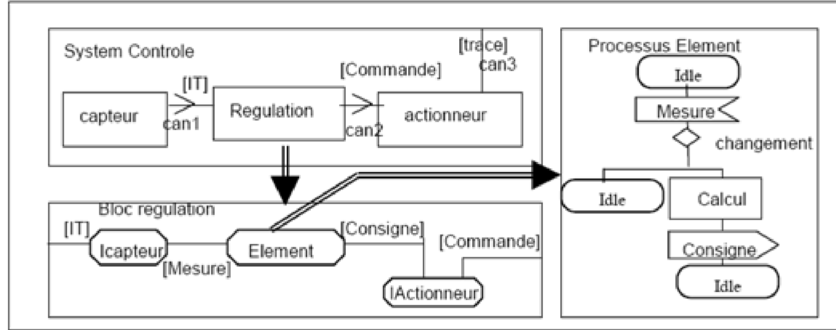

Fig. 2.2 – Exemple de modélisation SDL

2.3.1.3 Comportement

Un processus SDL est une machine à états caractérisée par une file d'attente FIFO, des données, des signaux entrants ou sortants et un ensemble d'états reliés par des transitions. Un état particulier du processus est l'état de départ où se place l'instance de processus à sa création. Une transition est caractérisée par une condition de tir et un corps. La fin de la transition aboutit à un nouvel état (pouvant être le même que l'état précédent).

Une transition est tirée soit sur réception d'un signal, soit de manière aléatoire (transition spontanée). Ce dernier aspect est utile pour modéliser des systèmes non déterministes. Il est possible de lier une condition (basée sur un calcul) de validation à une transition. Dans un état donné, le signal pouvant être consommé (signal présent et condition de validation ok) est absorbé. Si le signal n'est pas attendu dans l'état courant, il est éliminé sauf s'il est explicitement sauvegardé (mot clé save). Si le signal est attendu mais que la condition de validation n'est pas validée, le signal reste dans la file d'attente.

Le corps de la transition est caractérisé par une séquence d'actions (tâche au sens SDL, appel de procédure ou création d'autres instances de processus) et l'émission de plusieurs signaux. Les décisions permettent d'exprimer plusieurs chemins possibles dans une même transition.

Dans l'exemple de la Figure 2.2, lorsque le processus Element est dans l'état Idle et qu'il reçoit un signal Mesure, si les données véhiculées par le signal respectent une certaine condition changement, une procédure Calcul est exécutée afin de produire un nouveau signal Consigne. Sinon le processus n'effectue aucune action et revient à l'état idle.

2.3.1.4 Les données

La structuration des données se fait via des types (prédéfinis ou construits) déclarés au niveau des blocs ou du système, selon la syntaxe ADT (Abstract Data Types). Les données sont, elles, exclusivement déclarées au sein d'un processus. Les données et les types ne sont pas visibles à l'extérieur de leur zone de déclaration.

L'échange de données entre processus, se fait soit par signal soit par partage des données. A cet effet, il existe deux mécanismes : l'import/export et le revealed/view. Ces mécanismes de partage sont limités aux données déclarées au niveau des processus (et non par exemple au niveau des procédures).

Dans le premier mécanisme, une donnée D1 déclarée partagée (mot clé exported) possède une copie D2 en variable globale. Le processus doit alors explicitement (mot clé export) signifier la mise à jour de la copie D2 à partir de la donnée D1. Un processus lecteur de cette donnée partagée déclare une donnée locale D3 (mot clé imported). Puis il doit explicitement (mot clé import) signifié la mise à jour de la donnée locale D3 à partir de la copie D2. Ce mécanisme de copie est créé pour chaque instance de processus. Pour éviter tout conflit, lors de la consultation, le processus lecteur doit indiquer le PID du processus émetteur de la donnée.

Dans le deuxième mécanisme, une donnée est déclarée visible de l'extérieur (mot clé revealed), et tous les processus appartenant au même bloc peuvent consulter (mot clé viewed) cette donnée (la modification est bien sur interdite !). Le mode revealed/view permet d'avoir l'accès en lecture à une donnée « en continu », alors qu'avec le mode export/import, l'accès porte sur une donnée pouvant être ancienne.

2.3.1.5 Concurrence, protection et synchronisation des données

Les instances de processus s'exécutent en parallèle. Pour une instance, une seule transition est exécutée à la fois : il n'existe pas de concurrence intra-« instance de processus » (c'est l'hypothèse RTC « Run To Completion »). Par contre, si deux transitions (de 2 instances de processus) sont tirables au même instant, aucune relation d'ordre n'est donnée a priori.

En SDL, les données étant locales aux processus, elles sont inaccessibles de l'extérieur. Comme leur modification a lieu dans une transition et que cette dernière ne peut être interrompue, les données sont implicitement protégées car manipulées en exclusion mutuelle. Enfin, les opérations de partage se font en mode atomique pour assurer la protection des données.

2.3.1.6 Signalisation temporelle et priorités

La signalisation temporelle se fait par l'activation ou la désactivation de timers. Un timer est déclaré au niveau d'un processus. Il peut être armé (set) et interrompu (reset). Lors de son initialisation, on associe au timer une date relative ou absolue d'expiration (la primitive NOW de SDL renvoie l'heure courante). On peut alors lui associer des données. S'il n'est pas désactivé, à la date d'échéance, le processus reçoit dans sa file d'attente un signal portant le nom du timer. Ce signal véhicule les données associées au timer.

Enfin, il est possible de rendre un signal plus prioritaire pour un état donné (mise en cause du principe premier arrivé/premier servi). Cet aspect n'est pas réellement lié à un aspect temporel. Mais il peut permettre de rendre un message plus prioritaire, et donc de prendre en compte des contraintes temporelles d'une application.

Au-delà des ces informations, les notions de Qualité de Service sont absentes du modèle SDL.

2.4 Présentation de l'environnement de développement TauSDL

On se propose dans cette section de faire une petite présentation de l'environnement de développement TauSDL 4.4 développé par la société Telelogic [22], avec lequel nous avons pu modéliser, simuler notre modélisation du SNMPv1.

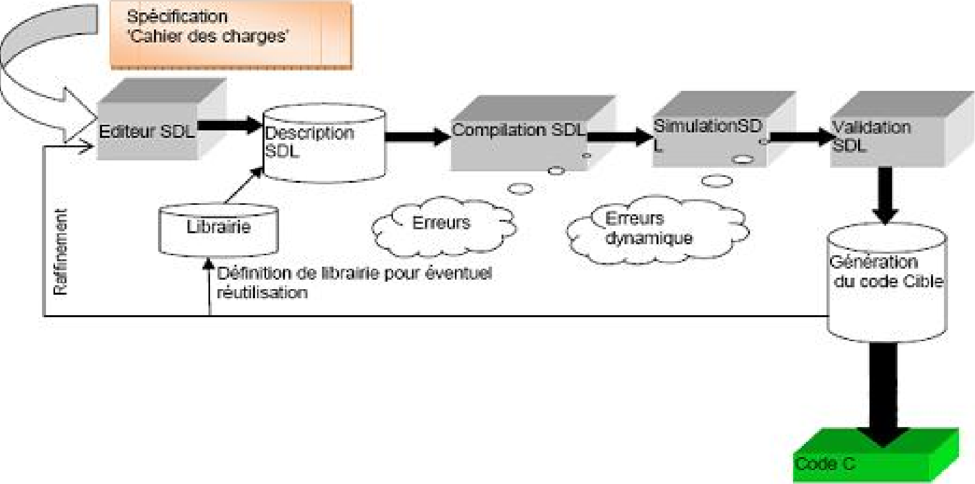

FIG. 2.3 – Le flot de conception dans TauSDL

La figure 2.3 expose clairement la suite logique de la conception assistée par TauSDL. Ainsi, l'approche de création de tout projet se résume aux étapes suivantes :

1. On commence par décrire le programme en utilisant l'éditeur SDL,
2. Ensuite, avec le compilateur SDL, on prendra connaissance des erreurs de conception,
3. Puis, il y a le processus de validation simulée, en d'autres termes, le programme va simuler tous les cas possibles et va détecter les erreurs liées à l'utilisation du système,
4. L'avant dernière étape, est le Targeting (ciblage en français). Elle consiste à cibler l'environnement dans le quel, on veut que notre programme tourne,
5. Enfin, lorsque toutes les étapes précédentes se sont bien déroulées, le code est alors généré, l'application est ainsi créée, un fichier exécutable est prêt à être utilisé.

2.5 Modèle SDL d'un agent SNMPv1

2.5.1 Choix de la version

La version (SNMPv1) est choisie parce que :

– elle satisfait pleinement à toutes les exigences d'une solution de gestion et de tests.
– la version 1 est la plus utilisée (comme déjà énoncé dans le chapitre1).
– le but de notre travail est de valider l'idée de l'implémentation du SNMP sur SoC. La version 3 a le module « ***sécurité*** » en plus. C'est le principal avantage qui n'a pas une importance capitale lors de la phase de modélisation.

Comme on a choisi la version SNMPv1 pour notre application, la RFC1157 [4] est utilisée pour définir le protocole SNMP et la RFC1155 [5] est utilisée pour définir la SMI (Structure of Management Information).

2.5.2 L'agent SNMPv1

La figure 2.4 est la description de notre système (agent SNMP). Comme présenté dans cette figure, notre système comporte cinq (05) blocs, chacun jouant un rôle bien déterminé et dont trois (03) communiquant avec l'environnement extérieur. Nous avons également défini des signaux qui peuvent être émis de/et vers leurs blocs respectifs. En quelque sorte, ces signaux représentent un protocole de communication entres les cinq blocs.

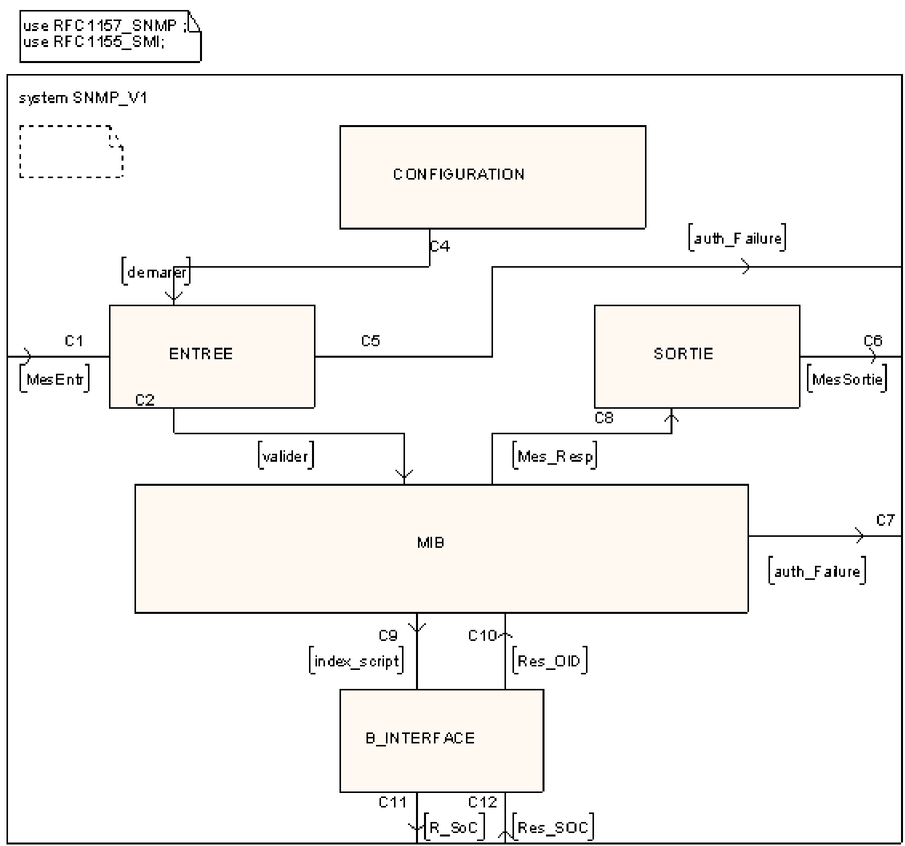

Fig. 2.4 – Système SDL de l'agent SNMP

Maintenant nous allons présenter le rôle de chaque bloc en commençant par le bloc de configuration « CONFIGURATION », en suite le bloc d'entrée « ENTREE », le bloc « MIB » qui est le cœur de notre système, et les blocs de sortie « SORTIE » et « INTERFACE ».

2.5.2.1 Bloc « *CONFIGURATION* »

Le bloc configuration a pour rôle d'extraire les informations concernent la configuration de l'agent SNMP comme par exemple l'adresse IP des entités de protocole, la communauté et le type d'accès aux

objets de la MIB à partir d'un fichier enregistré dans l'agent SNMP.

Les figures 2.5 et 2.6 représentent la modélisation SDL de ce bloc, les informations de configuration sont extraites, interprétées et transformées sous forme d'octets utilisables par les autres blocs (bloc d'« ENTREE», « SORTIE» et « MIB»). Le signal ***'demarer'*** véhicule cette information de configuration.

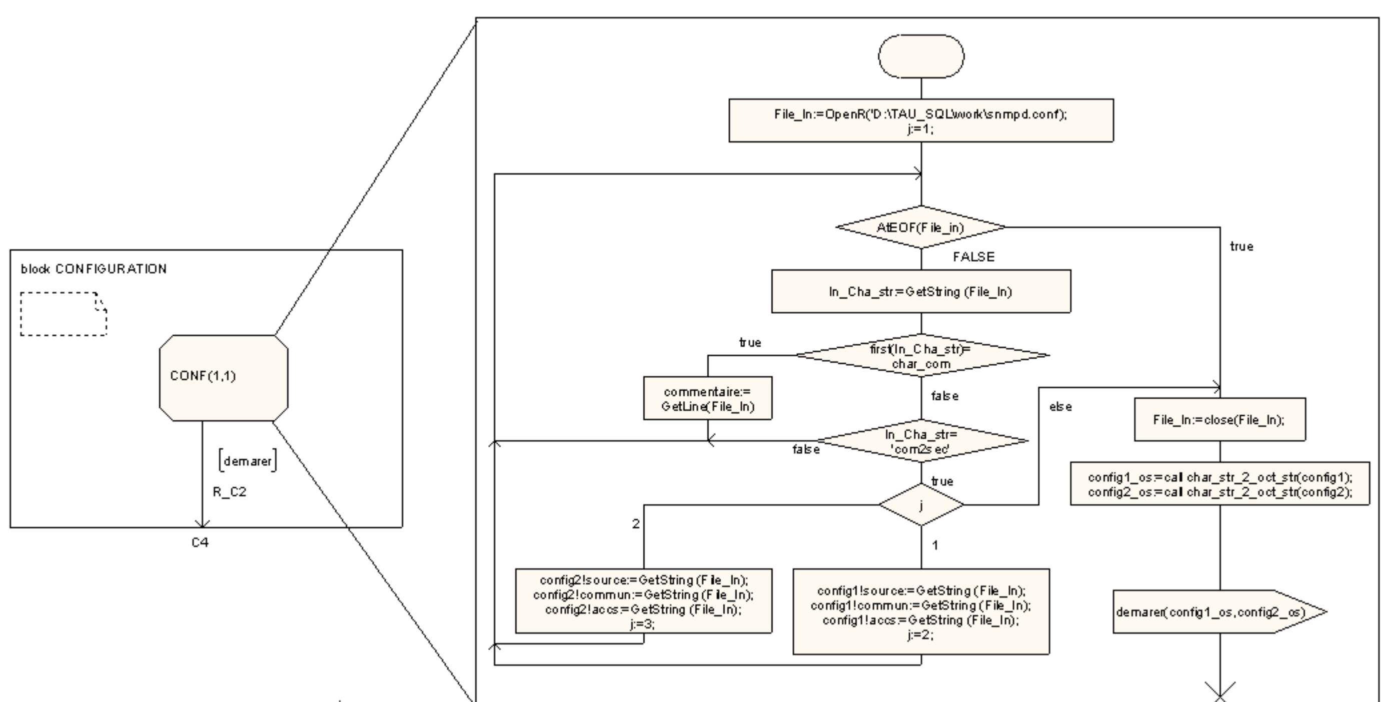

FIG. 2.5 – Bloc configuration

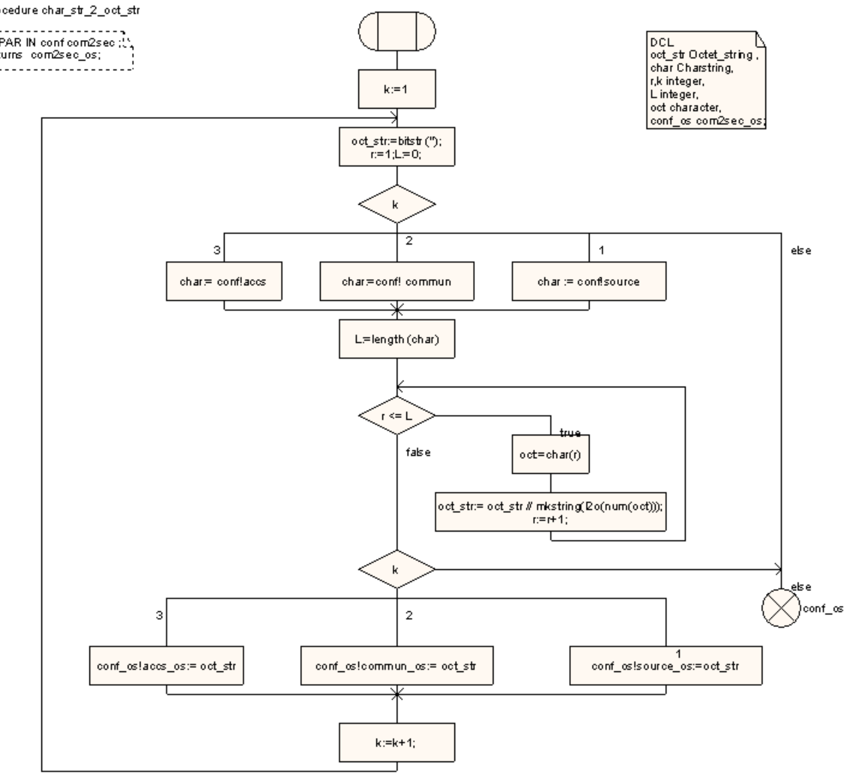

FIG. 2.6 – Procédure "char_str_2_oct_str"

La structure de chaque profil d'authentification est enregistré dans un fichier *"snmpd.conf"* (figure 2.7).

Dans ce fichier les lignes qui commencent avec ' # ' sont des commentaires.

Pour chaque profil, on définit trois champs :

- *source* : définit l'adress IP ou les adresses IP (un group) des entitiés d'application SNMP.
- *community* : définit les communautés pour chaque IP ou pour chaque groupe.
- *access* : définit les privilèges d'accès spécifiés à les variables dans les branches de la MIB qui peuvent être soit en lecture-seul (READ-ONLY,RO) soit en lecture-ecriture (READ-WRITE,RW).

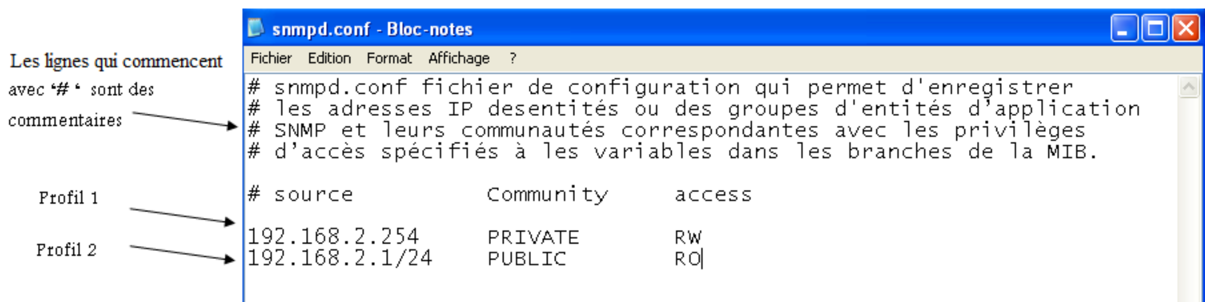

FIG. 2.7 – Fichier snmpd.conf

2.5.2.2 Bloc « *ENTREE* »

Puisque le message SNMP lui même est décomposé en deux parties, version et nom de la communauté comme une première partie, et le PDU comme une deuxième partie (Figure 2.8), on va traiter les messages reçus en deux parties. On commence tout d'abord par authentifier les messages et après envoyer le données PDU pour la suite de traitement dans le cas où ils sont acceptés. Seuls les messages de la première version sont acceptés.

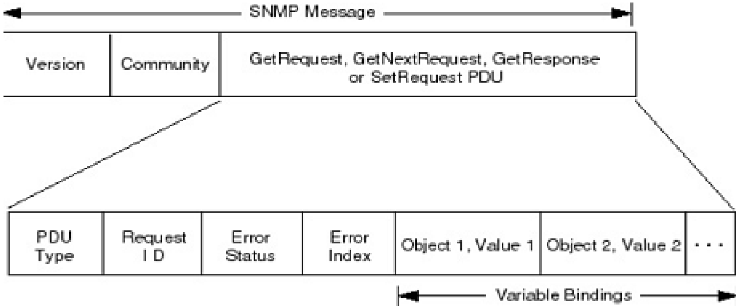

FIG. 2.8 – Datagramme IP d'un message SNMP

Ce bloc va recevoir de l'environnement extérieur le message SNMP de la couche UDP et l'adresse IP de l'entité émettrice à partir de la couche IP.

L'agent juge un message authentique si l'adresse IP et le nom de la communauté du message reçu sont identiques aux profils déjà enregistrés. Il envoie un signal ***'valider'*** au boc « MIB » pour la suite de traitement. Ce signal contient les données PDU (Figure 2.8)

Dans le cas où l'adresse IP et le nom de la communauté ne conviennent à aucun profil, le message est jugé non-authentique et l'agent envoi un signal *'**auth_ Failure**'* au bloc de sortie pour générer une trap avec un champ *'generic_trap'* de type *authenticationFailure*.

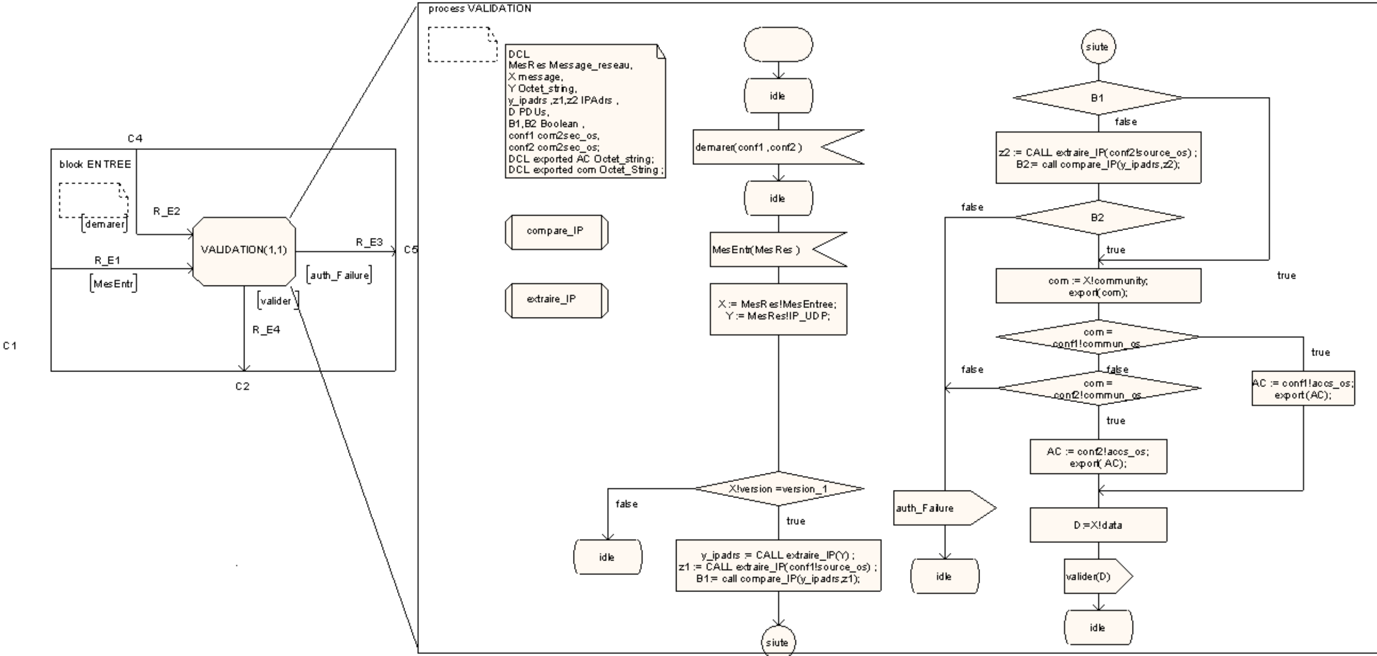

FIG. 2.9 – Bloc d'ENTREE

2.5.2.2.1 Utilisation d'ASN.1 dans la modélisation de l'agent SNMP Comme déjà expliqué dans le chapitre 1, les données du message SNMP sont définies en ASN.1 et codés en BER (Basic Encoding Rule) [24], (qui donne la configuration des bits émis ou reçus sur le réseau)

Et pour optimiser le processus de développement, on doit implémenter les types de données ASN.1 dans la modélisation du protocole SNMP.

En important des types de données ASN.1 vers le système SDL, on doit translater les définitions ASN.1 à SDL. L'environnement TauSDL fait cette tâche à l'aide d'un outil appelé "ASN.1 tools". Cet outil est automatiquement appelé lors de l'analyse du système SDL [25].

Cependant, avoir les types de données ASN.1 translatées en SDL ne suffit pas pour les inclure dans l'application. Lorsqu'on veut transférer les informations produites sur le réseau informatique, les valeurs des types de données doivent être codées. En transférant des signaux vers ou hors le système SDL, on doit également créer l'interface entre l'environnement et le système SDL.

Ainsi, le processus d'implémentation de types de données ASN.1 en SDL peut être divisé en trois étapes séparées :

- Créer la syntaxe abstraite ,
- créer la syntaxe de transfert ,
- compilation de l'application.

Pour créer la syntaxe abstraite on doit accomplir les tâches suivantes [25] :

- Ajouter les modules ASN.1 dans le projet, dans notre cas les modules ASN.1 sont récupérés directement à partir des RFC 1157 et RFC1155. On a utilisé les même types de données définies dans les RFC1157 et RFC1155.

– Importation des modules ASN.1 dans nos diagrammes SDL.

– Assigner des valeurs aux types de données.

TauSDL offre plusieurs manières pour créer la syntaxe de transfert. Les interfaces d'accès codage disponibles sont :

– Interface de base de SDL,

– interface avancée de SDL,

– interface de code de C.

On a choisi la troisième méthode.

2.5.2.3 Bloc « *MIB* »

Le bloc « *MIB* » est le cœur de notre système. Il communique avec tous les autres blocs. Il va recevoir les données PDU envoyés par le bloc d'« *ENTREE* » après que le message soit validé. Il traite le message selon leur type (get-request, get-next-request, set-request), et récupère les données recherche à partir du bloc « *INTERFACE* » et ensuite génère un PDU de sortie et l'envoie au bloc *« SORTIE ».*

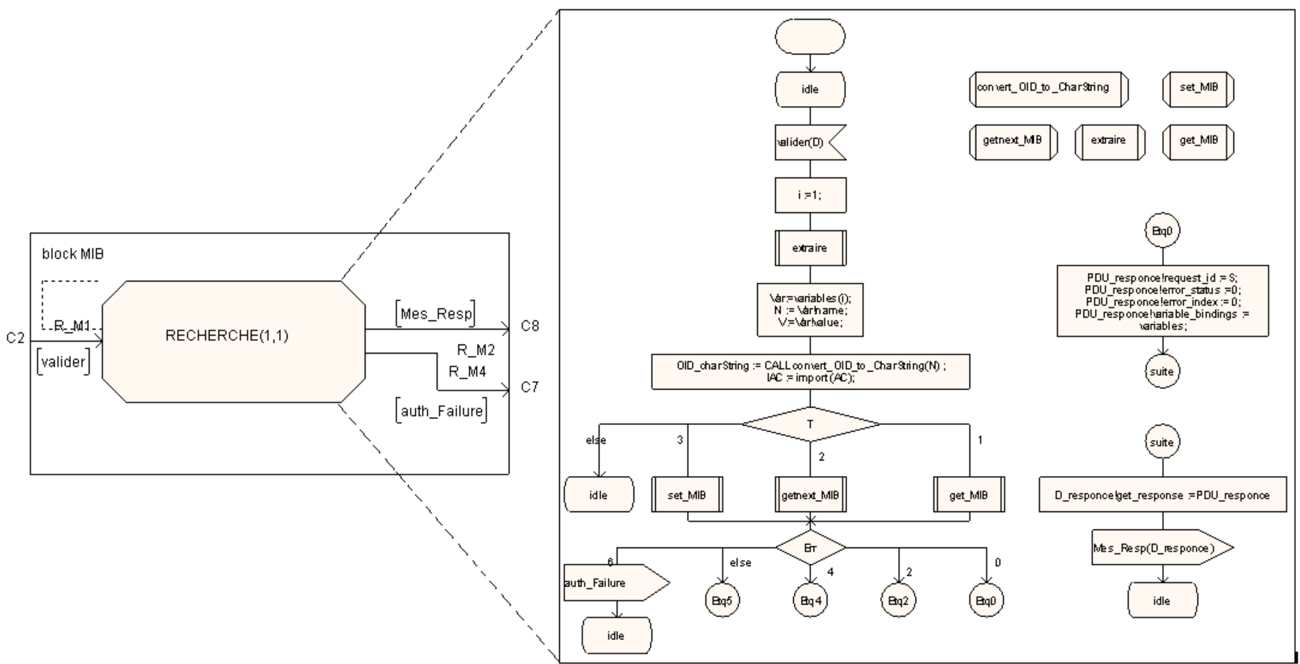

FIG. 2.10 – Bloc MIB

A la réception du signal ***'valider'***, le processus *'Recherche'* traite les données PDU champ par champ. (Figure 2.10).

Les champs '*ErrorStatus*' et '*ErrorIndex*' (Figure 2.8) sont généralement mis à zéro pour les messages d'entrée (les requêtes *: get-request, get-next-request et set-request).* Le champ '*RequestID*' est

enregistré pour qu'il soit retransmit dans le message de sortie (Corrélation entre les messages d'entrée avec les messages de sorties). Les variables du champ 'VariableBindings' sont extraites variable par variable. Pour chaque variable, on appelle la procédure *"convert_ OID_ to_ CharString"* qui convertie le type de l'OID de OBJECT IDENTIFIER au type *CharString*. Selon la valeur du champ '*PDUType*' (get-request(0), get-next-request(1) et set-request(3)), on appelle une procédure associée qui va rechercher le nouveau OID dans un fichier MIB.txt (la base de données dans notre cas). Une fois trouvé, on transmet l'index de la fonction associé au bloc interface.

Selon la réponse reçue du bloc interface, l'agent répond à l'entité émettrice en respectant les règles de réponse définies dans la RFC1157.

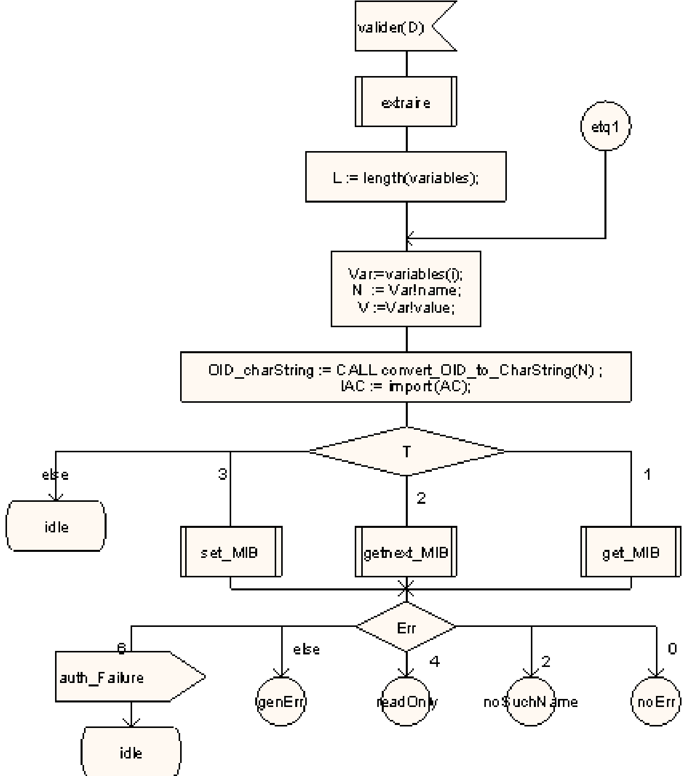

FIG. 2.11 – Différentes réponses dans le modèle SNMP

2.5.2.3.1 Fichier MIB.txt Tous les objets de la MIB (RFC1213-MIB) (Figure 2.12a), sont de type OBJECT TYPE définie par le MACRO TYPE d'ASN.1(Figure 2.12b), Comme le concept de MACRO TYPE est remplacé depuis 1994 par le concept de INFORMATION OBJECT CLASSES et INFORMATION OBJECTS, et SDL ne support pas les opérations sur ces types (pour extraire les champs des données définies par ces types, comme le *status* et *access*) on a créé un fichier 'MIB.txt' qui va représenter notre MIB en gardant tous les champs définis par OBJECT TYPE (Figure 2.12c). Ce fichier donne la possibilité de recherche les OIDs par l'agent en récupérant l'index de la fonction associée dans le bloc interface.

RFC1213-MIB.txt - Bloc-notes

Fichier Edition Format Affichage ?

```
sysDescr OBJECT-TYPE
    SYNTAX  DisplayString (SIZE (0..255))
    ACCESS  read-only
    STATUS  mandatory

    ::= { system 1 }

sysObjectID OBJECT-TYPE
    SYNTAX  OBJECT IDENTIFIER
    ACCESS  read-only
    STATUS  mandatory

    ::= { system 2 }

sysUpTime OBJECT-TYPE
    SYNTAX  TimeTicks
    ACCESS  read-only
    STATUS  mandatory

    ::= { system 3 }

sysContact OBJECT-TYPE
    SYNTAX  DisplayString (SIZE (0..255))
    ACCESS  read-write
    STATUS  mandatory

    ::= { system 4 }

sysName OBJECT-TYPE
    SYNTAX  DisplayString (SIZE (0..255))

    ::= { system 5 }

sysLocation OBJECT-TYPE
    SYNTAX  DisplayString (SIZE (0..255))
```

(a) Structure des données de la MIB

```
OBJECT-TYPE MACRO::=
BEGIN
TYPE NOTATION::= "SYNTAX" type (TYPE ObjectSyntax)
"ACCESS" Access
"STATUS" Status
VALUE NOTATION::= value (VALUE ObjectName)

Access ::= "read-only"
| "read-write"
| "write-only"
| "not-accessible"
Status ::= "mandatory"
| "optional"
| "obsolete"
END
```

(b) Structure de MACRO TYPE

MIB.txt - Bloc-notes

Fichier Edition Format Affichage ?

```
# Value(OID)        Access      Ref_Script    Syntax        Status

1.3.6.1.2.1.1.1.0   read-only      1          sysDescr      mandatory
1.3.6.1.2.1.1.2.0   read-only      2          sysObjectID   mandatory
1.3.6.1.2.1.1.3.0   read-only      3          sysUpTime     mandatory
1.3.6.1.2.1.1.4.0   read-write     4          sysContact    mandatory
1.3.6.1.2.1.1.5.0   read-write     5          sysName       mandatory
1.3.6.1.2.1.1.6.0   read-write     6          sysLocation   mandatory
1.3.6.1.2.1.1.7.0   read-only      7          sysServices   mandatory
```

(c) Fichier MIB.txt

FIG. 2.12 – Structure des données de la MIB

2.5.2.4 Bloc « ***SORTIE*** »

Nous passons maintenant à la présentation du bloc qui se trouve à l'autre extrémité de la chaîne de traitement, car il joue le même rôle que le bloc « *ENTREE* » sauf que la fonction s'inverse. En effet, ce bloc, après réception du signal ***"Mes_ Resp"*** avec comme paramètre une variable de type *PDUs* (voir module RFC1157) du bloc « MIB », aura pour tâche de construire un message de sortie en assignant la version et la communauté au PDU reçu qu'il soit de type *Get-responce* ou bien de type *Trap*. Il est ensuite envoyé à l'environnement extérieur.

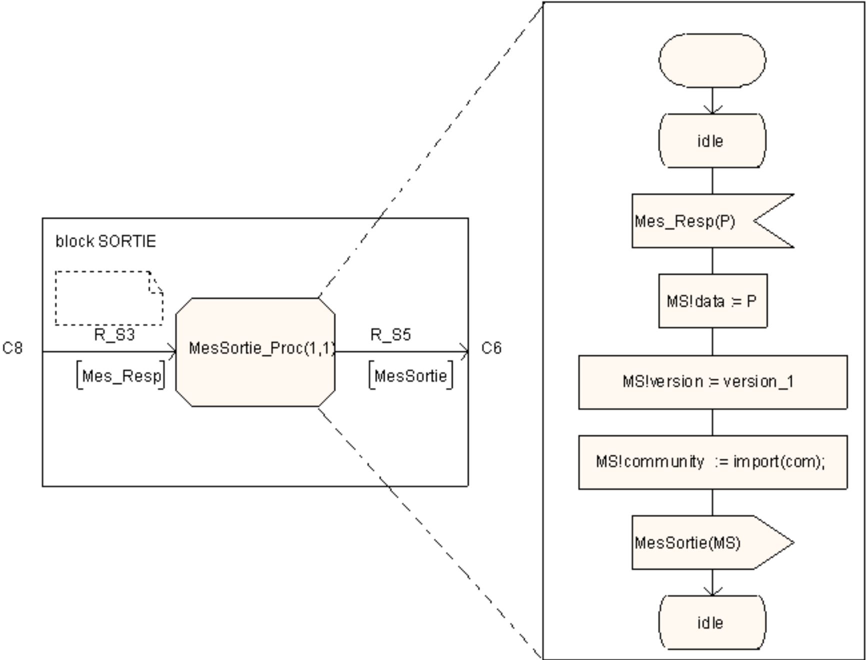

FIG. 2.13 – Bloc Sortie

2.5.2.5 Bloc « ***INTERFACE*** »

Comme son nom l'indique, ce bloc joue le rôle d'interface entre l'agent SNMP et les éléments à l'intérieur du SoC. Ce bloc reçoit un signal **«** ***index_ scrip*** **»** contenants l'index du script de la valeur OID recherchée. Selon cette valeur, une procédure est appelée. Cette procédure exécute un script pour intéragir avec les éléments du SoC en envoyant le signal ***"Val_ recherche_ env"*** à l'environnement. La valeur retournée par l'environnement doit être convertie pour s'adapter aux types de donnée définies par SDL. Enfin la valeur OID recherchée est extraite et envoyée au bloc MIB (Figure 2.14).

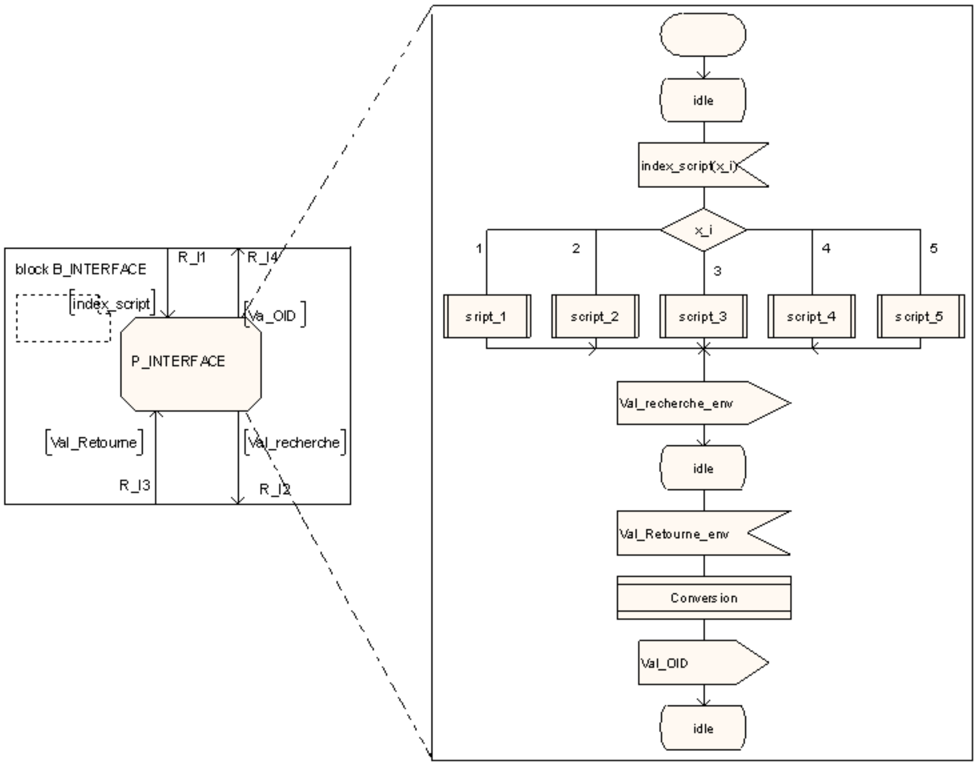

Fig. 2.14 – Bloc Interface

Conclusion

Nous avons montré dans ce chapitre que le langage formel SDL épaulé par l'environnement de développement TauSDL de Telelogic se trouve tout à fait approprié pour la spécification de notre système, autrement dit la modélisation de l'agent SNMP. Une production automatique du code est alors un atout. Ce code sera l'élément de base dans la conception de la partie software du système, qui sera étudiée dans le chapitre suivant.

Chapitre 3

IP CORE Agent SNMP

Introduction

Dans ce chapitre nous présentons la deuxième partie de notre développement qui concerne la réalisation de notre IP core. Nous commençons par introduire quelques notions théoriques de l'aspect co-design des éléments réutilisables (IP cores). Le flot de conception suivi sera décrit. Il sera suivi par le détail de notre développement. On le conclura par la présentation des résultats de sa mise en œuvre sur une carte de développement Spartan-3 E.

3.1 Le Co-design

Devant les possibilités offertes par les technologies de fabrication, les techniques de conception des systèmes électroniques vont évoluer vers l'intégration de systèmes de plus en plus complexes avec des durées d'obsolescence de plus en plus courtes. Des blocs fonctionnels déjà validés (y compris sur Silicium) appelés généralement IP (pour Intellectual Property) devront être de plus en plus utilisés. De ce fait, les outils classiques de Conception Assistée par Ordinateur (CAO) dans les domaines de la Microélectronique sont évolués en prenant de plus en plus en compte l'aspect système et le recours au prototypage à des fins de validation [26].

Les classiques bibliothèques utilisées pour la conception des Circuits Intégrés pour Applications Spécifiques (ASICs) sont complétées, et remplacées, par des bases de données de composants virtuels dont la fonctionnalité pourra correspondre à un cœur de processeur ou même à un ordinateur complet avec sa mémoire et ses entrées/sorties. Le concepteur d'ASICs a "manipulé" des transistors pendant les années 80, des blocs fonctionnels pendant les années 90 ; il assemble de plus en plus des composants virtuels complexes ou IPs aujourd'huit.

La conception réutilise ; elle devient incrémentale et flexible. A partir de blocs IPs, les concepteurs adapteront rapidement le système à l'application visée. Dans les années 80, à coté des outils graphiques (schématique, dessin de masques, placement/routage), les concepteurs de circuits intégrés disposaient de simulateurs électriques au niveau transistor pour les fonctions analogiques et de simulateurs au niveau porte pour les fonctions logiques.

A la fin des années 80, l'utilisation du langage VHDL (comportemental et structurel) s'est généralisée. Ce langage, initialement développé pour la modélisation et la simulation, est également utilisé pour la description comportementale ou RTL (transfert de registres) d'une architecture logique pour la synthèse automatique, des environnements de CAO permettent de vérifier la fonctionnalité du système à ce niveau d'abstraction par des simulations de type flot de données. Ces outils permettent également d'assembler et de co-simuler des modèles virtuels décrits dans différents langages à des niveaux d'abstraction différents.

La figure 3.1 représente le flot de conception d'un système sur silicium. La réutilisation de blocs affecte le flot de conception sous plusieurs aspects. Il convient de distinguer plusieurs niveaux de blocs réutilisables ou IP : les " hard-cores " sont des blocs entièrement conçus et optimisés pour une technologie donnée, les " softcores " sont des blocs définissant une architecture au niveau RTL. Du point de vue de l'utilisateur de blocs réutilisables, il est nécessaire de faire le choix des blocs qui permettront d'implanter efficacement le plus grand nombre de fonctionnalités du système à un coût acceptable. Le flot n'est alors pas complètement descendant puisqu'il sera nécessaire de capturer pour les blocs choisis leur comportement, d'obtenir des modèles de simulation, pour finalement intégrer ces blocs dans le système. En revanche, du point de vue du fournisseur de blocs réutilisables, la notion primordiale est celle de la conception en vue de la réutilisation (" Design For Reuse "). Ainsi, un bloc IP pourra être défini pour une application donnée, puis optimisé pour une technologie cible. Le flot est alors vraiment descendant : le comportement du bloc est d'abord décrit, puis son implantation est progressivement raffinée.

Les CAO de SoC se situent à différents niveaux : spécification, modélisation, synthèse, vérification. On parle de technologie de la conception (" Design Technology "), notion qui recouvre les algorithmes, les outils logiciels et matériels, ainsi que les méthodes de conception des systèmes.

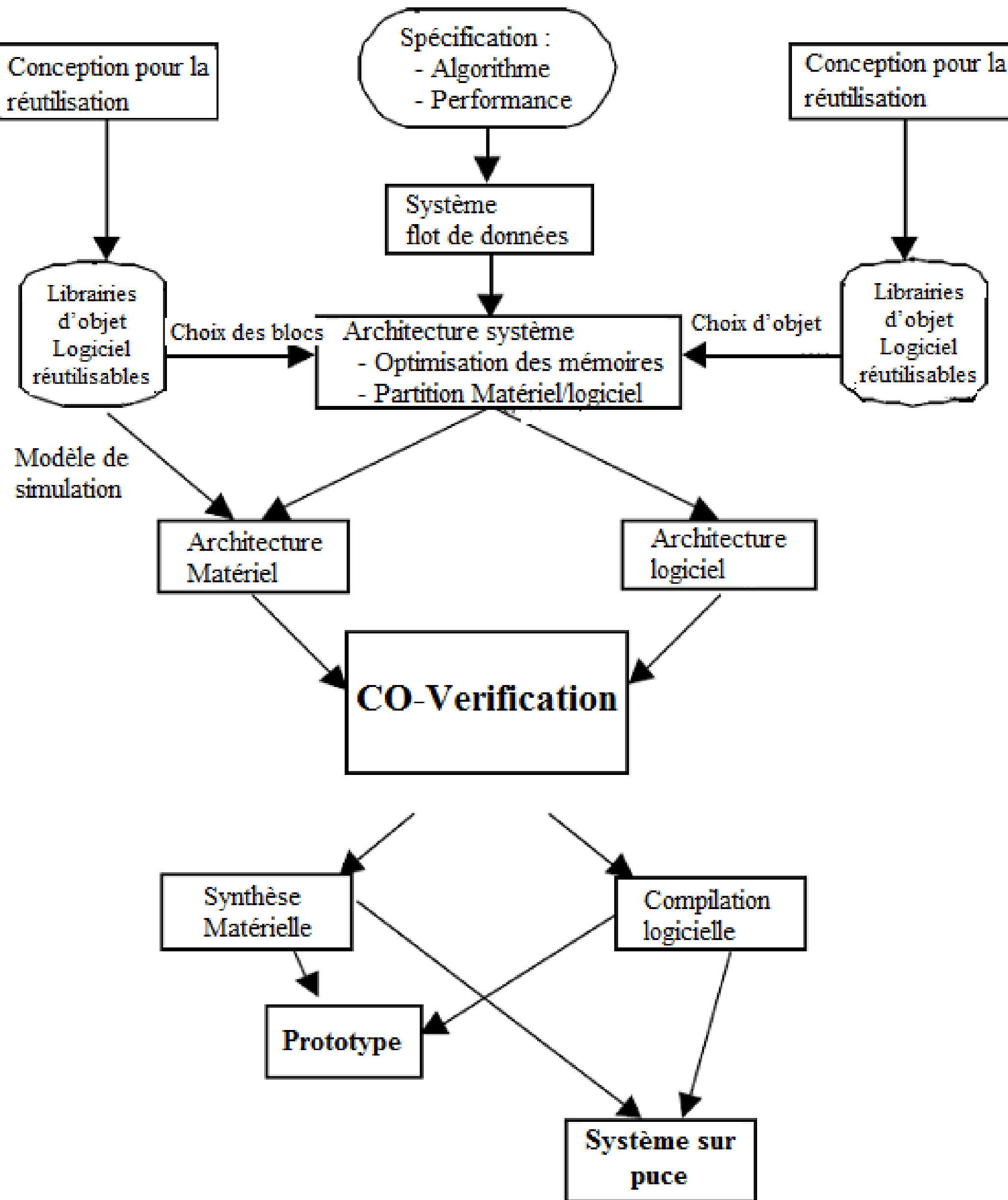

Fig. 3.1 – Flot de conception d'un système sur silicium

3.2 Environnement de développement utilisé

L'environnement de développement fourni par Xilinx permet la configuration des circuits de type FPGA est appelé EDK (Embedded Development Kit). Grâce au principe du Co-Design, cet environnement nous permet d'une part de développer le hardware (configuration de MicroBlaze ou PowerPC et de ces périphériques) et d'une autre part, de développer le software (plateformes et les applications) puis de générer le Bitstream pour des configurations déférentes du FPGA [27].

EDK contient un environnement de développement intégré (IDE) appelé Xilinx Platform Studio (XPS) qui est une interface graphique d'EDK qui permet de définir le matériel et le logiciel qui seront

implémentés sur le circuit FPGA.

EDK contient aussi la plateforme SDK (Software Developement Kit) qui peut être utilisée pour un développement avancée des applications softwares en langage C ou C++.

EDK inclut :

- The Xilinx Platform Studio (XPS).
- The embedded system tool suite.
- Embedded processing IP cores, (comme les processeurs et les périphiriques).
- The Platform Studio SDK (Software Development Kit).

Le flot de conception d'un projet sous l'environnement EDK est représenté par la figure suivante [28] :

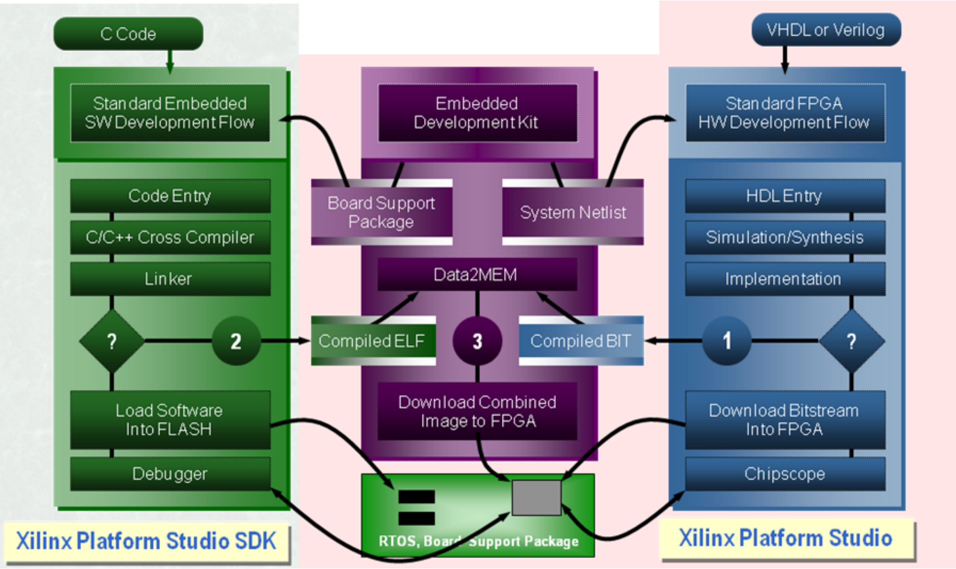

FIG. 3.2 – Flot de conception sous EDK

L'aspect co-design du flot de conception sous EDK nous permet de développer la plateforme hardware en parallèle avec la plateforme software.

Dans le flot de conception de la partie Hardware, on choisi un processeur IP et les différents périphériques qui lui seront connectés puis on exécute l'outil PlatGen de EDK qui les configure en se basant sur les caractéristiques spécifiées dans le fichier MHS. Une fois la plateforme matérielle spécifiée, une Netlist[1] est synthétisée à partir des codes sources des IP cores propriétaires fournis par l'environnement ISE en langage VHDL. La configuration de la carte et les connexions des différents périphériques avec le circuit FPGA spécifiés dans le fichier de contrainte UCF sera ensuite lié avec la Netlist pour former un fichier .bit qui est le fichier de configuration de toute la plateforme hardware d'un projet sous EDK.

[1]Netlist est une forme synthétisée du matériel qui contient les données logiques de la conception et des contraintes

Dans le flot de conception software, le programme de l'application doit être écris soit en langage assembleur soit en langage C ou C++. Le code source obtenu doit être compilé pour avoir des fichiers en code objet qui seront liés avec des librairies spécifiques. Ces dernières sont générées lors de l'exécution de l'outil PlatGen de EDK pour les spécifications du software qui sont données dans le fichier MSS. On obtient à la fin du flot software, un fichier exécutable.

Le fichier .bit obtenu à la fin du flot matériel et le fichier exécutable de l'application obtenu à la fin du flot software sont liés par l'outil DATA2MEM pour former un seul fichier download.bit qui configure tout le système en entier et qui sera chargé sur le circuit FPGA.

3.2.1 Bibliothèque IP cores existante (MP, bus, E/S, Mémoires)

L'environnement EDK contient des IP cores déjà conçues qui peuvent être utilisés pendant la conception des systèmes, voilà la bibliothèque disponible dans EDK11.1 groupé par thème.

Ces IP cores seront utilisés pour développer notre IP core.

o Analog (convertisseur analogique/numérique, numérique/analogique)

o Bus and Bridge FSL (BUS)

o Clock, Reset and Interrupt

o Communication High-Speed

o Communication Low-Speed

o DMA and Timer

o Debug

o General Purpose IO

o IO Module

o Interprocessor Communication

o Memory and Memory Controller

o PCI

o Peripheral Controller

o Processor

o Utility

3.2.2 Génération d'IP core dans EDK

EDK permet aussi de générer des IP cores, une fois le développement de l'application est terminé un Wizard sous EDK est utilisé pour générer notre IP core qui sera une unité réutilisable.

3.3 Architecture IP core (Agent SNMP)

Dans la suite de ce chapitre on donnera une description détaillée de chaque partie du développement Co-design à savoir la partie hardware et software, et après on donne l'approche suivie pour générer notre IP core dans EDK.

3.3.1 Architecture hardware

Notre IP core sera une combinaison des IP cores disponibles dans la bibliothèque d'EDK, et selon les besoins de notre application, on a configuré le processeur MicroBlaze en spécifiant les différents périphériques, les pilotes, les librairies ainsi que les bus de données et d'instructions, qui nous permettent de récupérer, traiter et d'envoyer les trames du réseau. Ceci est réalisé grâce à l'outil disponible sur EDK permettant l'ajout et la modification des IP cores (Add/Edit cores).

3.3.1.1 Justification des constituants

Les composants clés de la configuration hardware incluent :

- **Un processeur :** la plateforme de développement permet l'utilisation soit le processeur MicroBlaze ou PowerPC. Le processeur PowerPC c'est un processeur Hard (implémentation matériel sur circuit) et la carte qu'on a utilisée (Spartan-3E) ne contient pas le processeur PowerPC, donc on a choisir d'utilisé le processeur Soft MicroBlaze qu'est disponible dans la bibliothèque d'EDK.

- **EMAC** : c'est l'IP core qui assure les fonctionnalités réseaux, la pile protocolaire LwIP supporte l'IP core EMAC « *xps_ ethernetlite* ».

- **Timer :** minuterie pour maintenir des temporisateurs TCP, la pile protocolaire LwIP exige que certaines fonctions soient appelées périodiquement par l'application. Une application peut faire ça par appelle à un contrôleur d'interruptions avec un Timer.

- **Gestionnaire d'interruptions** : module pour la gestion des interruptions.

- **MPMC** : contrôleur de mémoire Multi-Port pour la gestion de la mémoire utilisé par la pile protocolaire.

- **Clock_Generator** : c'est le générateur d'horloge pour le fonctionnement du système

La figure 3.3 montre le schéma bloc de notre configuration hardware. Le système comporte un processeur relié à un contrôleur de mémoire Multi-Port (MPMC) et aux autres périphériques (xps_Timer et xps_ethernetlite) sur un bus PLB v4.6. Les interruptions du xps_TIMER et de xps_ethernetlite sont exigées, et doivent se reliées au contrôleur d'interruption xps_INTC.

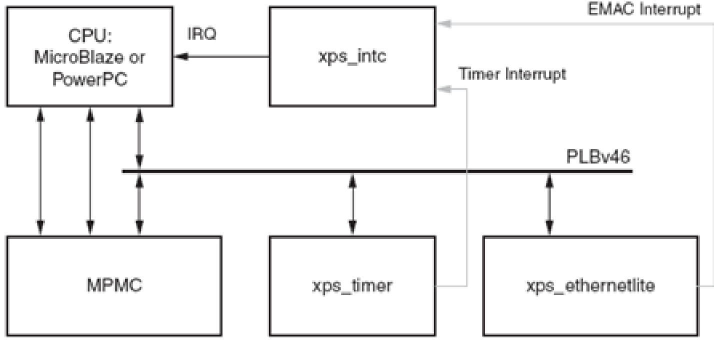

FIG. 3.3 – Schéma bloc de la configuration hardware

3.3.1.2 Architecture hardware complète

La figure suivante donne un schéma détaillé des composants de notre IP core. Et par la suite on donne les spécifications de chaque composant ;

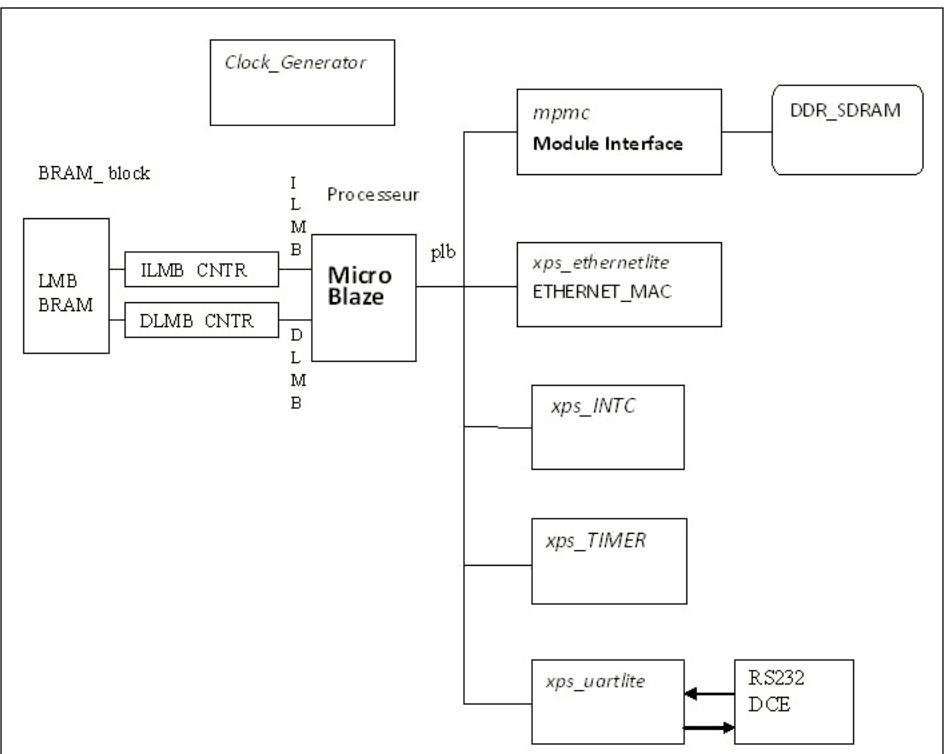

FIG. 3.4 – Bloc Diagramme détaillé

Ce diagramme nous permet aussi de comprendre comment agit MicroBlaze pour gouverner ces différents périphériques, et ce via les deux bus PLB et LMB.

3.3.1.2.1 MicroBlaze : Le processeur soft MicroBlaze embarqué est un processeur à jeu d'instruction réduit (RISC[2]) optimisé pour l'implémentation de Xilinx sur des circuits FPGAs. Développé par Xilinx, le processeur soft MicroBlaze est disponible dans l'environnement EDK de Xilinx sous la forme d'une description faite en HDL (langage de description du matériel : Hardware Description Language). De nombreux périphériques peuvent lui être connectés via le bus PLB tel un contrôleur réseau Ethernet ou un module permettant la gestion du port série.

Conçu selon une architecture dite HARVARD séparant le chemin des instructions de celui des données et permettant un parallélisme dans le transfert des données et des instructions, MicroBlaze réalise l'exécution des instructions en un seul cycle d'horloge grâce à l'utilisation du pipeline. La Figure 3.5 nous donne un aperçu général sur l'architecture de MicroBlaze.

MicroBlaze se caractérise par [29] :

- 32 registres banalisés à 32 bits.
- Un mot d'instruction à 32 bits avec trois opérandes et deux modes d'adressage.
- des bus de données et d'instructions séparés à 32 bits conformiment aux spécifications des bus PLB d'IBM.
- Des bus de données et d'instructions séparés de 32 bits connectant le processeur aux blocs mémoire internes via le bus LMB (iLMB,dLMB).
- Des bus d'adresse à 32 bits.
- Un pipeline qui assure l'exécution d'une instruction à chaque cycle d'horloge.
- Un cache d'instructions et de données qui permet d'augmenter la bande passante entre le processeur et la mémoire.
- Un debugger logique du matériel.
- Un support FSL (Fast Simplex link) qui permet une seule fonction à la fois soit la lecture soit l'écriture

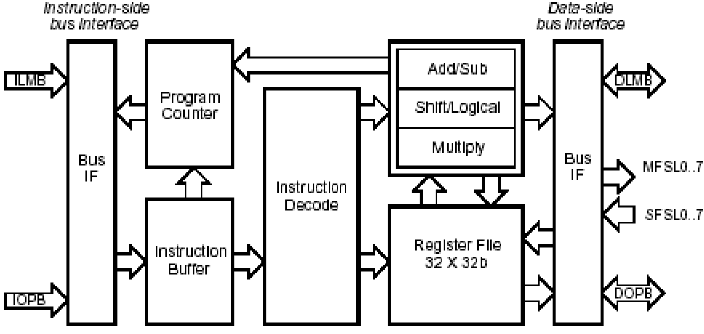

FIG. 3.5 – L'architecture de MicroBlaze

[2]C'est un jeu d'instruction formé par les instructions les plus simples et les plus fréquemment utilisées. Un processeur RISC se caractérise par : une exécution simple (en un cycle d'horloge) des instructions, une longueur d'instruction uniforme, des modes d'adressage simples, et une couche micro programmée inexistante.

3.3.1.2.2 Bus PLB : **Le Processor Local Bus** (**PLB**) est un bus introduit par IBM dans le cadre de l'architecture de bus CoreConnect, qui comprend également les bus OPB (On-chip Peripheral Bus) et DCR (Device Control Register).

Caractéristiques du bus PLB :

- Bus synchrone, non multiplexé.
- Bus de lecture et d'écriture séparés.
- Support de lecture/écriture concurrentes.
- Multimaître à priorité programable et disposant d'un arbitre.
- Adresses sur 32 bits.
- Implémentations sur 32, 64 et 128 bits de données.
- Fréquences supportées : 66, 133 et 183 MHz (respectivement pour les versions 32, 64 et 128 bits).
- Pipeliné, support des interruptions de transfert.
- Support des bursts de taille fixée et variable.
- Support du verrou

Pour EDK, Xilinx fournit le bus (PLB) v4.6 qui assure une infrastructure de bus permettant de connecter un nombre optionnel de maîtres et d'esclaves PLB dans un système global PLB. Il est composé d'une unité de contrôle du bus, une horloge de surveillance. Il contient aussi une interface esclave optionnelle, pour fournir l'accès à l'état de bus d'erreur [30].

3.3.1.2.3 Bus LMB : Le bus LMB est un bus synchrone utilisé principalement pour accéder aux blocks RAM inclus sur le FPGA. Il utilise un minimum de signaux de contrôle et protocole simple pour s'assurer d'accéder à la mémoire rapidement (un front d'horloge).

3.3.1.2.4 Xps_ethernetlite : C'est le module qui assure les fonctionnalités de réseau dans notre IP core.

Certains étapes sont nécessaires (configuration du hardware et du Software) pour la mise en œuvre de réseau (application Ethernet) sous EDK.

Les étapes clés sont :

1. configuration du hardware qui doit contenir un processeur, un IP core Ethernet, et IP core Timer. Les IP cores Timer et Ethernet doivent être connecté au processeur via un contrôleur d'interruption.
2. Insertion et configuration de la bibliothèque lwip130_v1_00_b pour être une partie de la plateforme Software. Cette bibliothèque représente la pile protocolaire TCP/IP.

Cette configuration sera étudiée dans la deuxième partie qui concerne la création de la plateforme Software.

Rappel sur la couche MAC : Le Contrôle d'accès au support (Media Access Control en anglais ou MAC) est une sous-couche, selon les standards de réseaux informatiques IEEE 802.x, de la partie inférieure de la couche de liaison de données dans le modèle OSI (Figure 3.6). Elle sert d'interface entre la partie logicielle contrôlant la liaison d'un nœud (Contrôle de la liaison logique) et la couche physique (matérielle). Par conséquent, elle est différente selon le type de média physique utilisé (Ethernet, Token Ring, WLAN, ...)

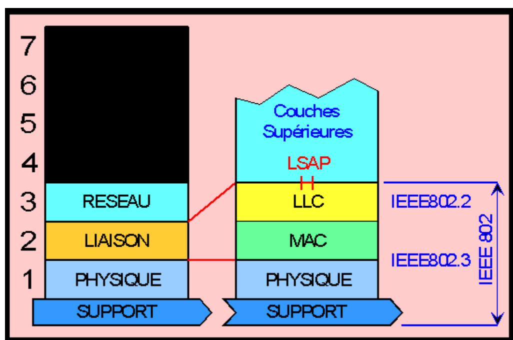

FIG. 3.6 – Emplacement de la sous-couche MAC dans le modèle OSI.

Le rôle de la sous-couche MAC est principalement de :

- reconnaître le début et la fin des trames dans le flux binaire reçu de la couche physique ;
- délimiter les trames envoyées en insérant des informations (comme des bits supplémentaires) dans ou entre celles-ci, afin que leur destinataire puisse en déterminer le début et la fin ;
- détecter les erreurs de transmission, par exemple à l'aide d'une somme de contrôle (checksum) insérée par l'émetteur et vérifiée par le récepteur ;
- insérer les adresses MAC de source et de destination dans chaque trame transmise ;
- filtrer les trames reçues en ne gardant que celles qui lui sont destinées, en vérifiant leur adresse MAC de destination ;
- contrôler l'accès au média physique lorsque celui-ci est partagé.

Une adresse MAC est une suite de 6 octets (souvent représentée sous la forme hexadécimale 01 :23 :45 :67 :89 : ab) qui identifie de façon unique chaque interface réseau.

XPS Ethernet Lite Media Access Controller (v2.01a) [31] : Pour notre application l'IP core disponible dans la bibliothèque d'EDK est XPS Ethernet Lite Media Access Controller (v2.01a)

Cet IP core est conçu pour intégrer les caractéristiques des applications décrites dans la norme IEEE Std. 802,3.

The Ethernet Lite MAC support MII de la standard IEEE Std. 802,3 pour communiquer avec la couche physique (PHY) et communique avec le processeur par le bus BLP (Processor Local Bus).elle est conçue pour fournir une interface de communication 10Mbps et 100Mbps (Fast Ethernet). L'objectif

est de fournir les fonctions minimales nécessaires pour fournir une interface Ethernet avec le moins de ressources utilisées.

Caractéristiques :

- Interface PLB est base sur les spécifications de PLB v4.6.
- Media Independent Interface (MII) pour connecter à un transceiver PHY 10/100 Mbps.
- Mémoire interne indépendante double port 2K octets Tx et Rx pour conservation des données pour un paquet.
- Mémoires tampons double, 4K octets ping-pong, pour Tx et Rx (optional).
- Gestionnaire d'interruption pour la réception et la transmission.

3.3.1.2.5 Xps_TIMER : Le timer est un périphérique on chip indispensable dans le cas de l'utilisation d'IP core xps_ethernetlite car il permet avec le module xps_INTC de générer des interruptions qui sont nécessaire pour le fonctionnement de certaines applications de la pile LwIP.

XPS Timer/Counter (v1.01a) est l'IP core disponible dans EDK11.1. C'est un module Timer 32 bits qui connecte au bus PLB caractérisé par [32] :

- Connexion comme esclave avec le bus de PLB V4.6 (32, 64 ou 128 bits) ;
- organisé en deux modules timers identiques programmables par intervalle avec les interruptions ;
- Sortie PWM (Pulse Width Modulation).

3.3.1.2.6 xps_INTC [33] : C'est le contrôleur d'interruptions nécessaire pour le fonctionnement de la pile LwIP comme décrit précédemment.

3.3.1.2.7 Mdm : Module Debug : Comme on l'a déjà vu, MicroBlaze est un processeur soft configurable selon les besoin de l'utilisateur, l'un des options qu'on peut configurer est le mode *"debug"*.

Un débogage hardware est utilisé pour un meilleur contrôle des ressources du processeur, et un débogage software en utilisant XMDStub (a ROM monitor) à travers UART (Universal Asynchronous Receiver-Transmitter), un cable JTAG, ou bien avec câble RS232 est utilisé pour un meilleur contrôle des applications.

Ce débogeur peut être connecté à MicroBlaze ou PowerPC implémentés sur le circuit FPGA pour exécuter une instruction de simulation. Un autre débuggeur appelé GDB de GNU est aussi nécessaire pour compléter le débogage du logiciel, il permet aux utilisateurs de commencer l'exécution du programme pour poser des conditions initiales, les points d'arrêt et pour examiner l'état du processeur et les composantes des mémoires. Lors de l'exécution du système sur la carte FPGA, XMD (Xilinx Microprocessor Debugge) peut être connecté à MicroBlaze via un module de déboguage en utilisant un câble JTAG.

Le module de débogage dans notre architecture est le module MicroBlaze Debug Module (MDM) (v1.00e) (voir [34] pour plus de détailles).

3.3.1.2.8 xps_uartlite : Utilisé pour une optionnelle sortie sur un Hyper Terminal, pour visualisation des résultats d'exécution.

3.3.2 Architecture Software

Après que la plateforme hardware est créée, et pour terminer le flot de conception de notre IP core il nous reste à créer la plateforme software et générer l'application software qu'est l'agent SNMP.

3.3.2.1 Architecture de l'application

Comme on déjà énoncé dans le chapitre précédent, notre architecture logicielle est autonome et ne nécessite aucun système d'exploitation même la pile protocolaire ne nécessite pas un système d'exploitation, donc la partie software est composée de :

- Plateforme software
  - o Pile protocolaire LwIP
  - o Libraires EDK
- L'application SNMP

D'où l'architecture complète de notre système (Figure 3.7) :

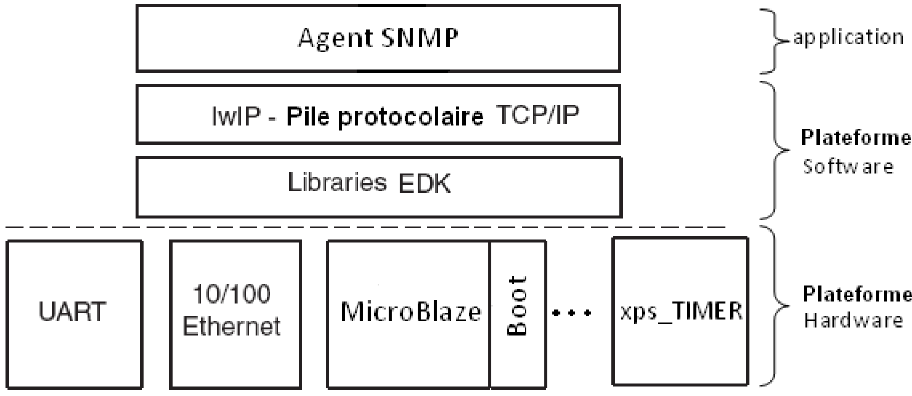

Fig. 3.7 – Architecture complète de l'application

La plateforme software est créée à l'aide de l'outil SDK (Software development Kit) d'EDK, cette outil permet de définir les librairies, les pilotes, les paramètres de personnalisation du processeur, les dispositifs d'entrées/sorties, les routines de traitement d'interruptions et d'autres caractéristiques du software.

La partie la plus importante de la plateforme software est la pile protocolaire LwIP.

3.3.2.2 Pile protocolaire LwIP [35]

LwIP est une implémentation minimale du protocole TCP / IP disponible sous une licence BSD.

LwIP est une pile autonome et ne dépend pas du système d'exploitation, mais peut être utilisée avec les systèmes d'exploitation.

L'objectif de la pile LwIP est de réduire l'utilisation mémoire et la taille du code, rendant LwIP utilisable pour les petits clients avec des ressources très limitées, comme les systèmes embarqués. Afin de réduire le traitement et les exigences de la mémoire, LwIP utilise une interface API[3] qui ne nécessite pas de copie de données.

Le LwIP fournit le support pour les protocoles suivants :

- IP (Internet Protocol) ;
- ICMP (Internet Control Message Protocol) ;
- UDP (User Datagram Protocol) ;
- TCP (Transmission Control Protocol) ;
- ARP (Address Resolution Protocol) ;
- DHCP (Dynamic Host Configuration Protocol).

3.3.2.2.1 Architecture de la pile : Comme dans beaucoup d'autres réalisations de TCP/IP, chaque protocole des couches de la pile est implémenté comme un module séparé, avec quelques fonctions agissant comme points d'entrée dans chaque protocole. Malgré que chaque protocole est implémenté séparément, quelques couches intègrent les fonctionnalités des autres couches de tell sort qu'ils améliorent les performances en terme de temps d'exécution et d'utilisation mémoire. Par exemple lorsqu'on vérifier le *checksum* (la somme de vérification) d'un segment TCP entrant, ou lorsque on démultiplexe un segment, l'adresse IP (source et destination) doivent être connues par le module TCP. Au lieu de les passer ces adresses au TCP par des appelles de fonction, le module TCP prend en compte la structure de l'en-tête IP, et peut donc extraire ces information lui-même.[35]

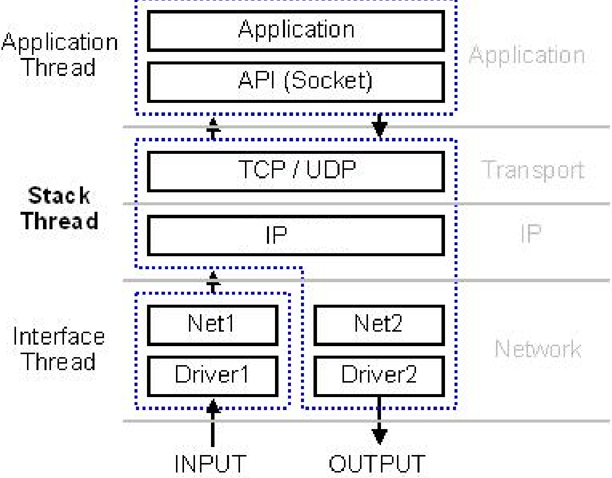

FIG. 3.8 – Architecture de la pile LwIP

[3]Application Program Interface

LwIP est composé de plusieurs modules, en plus des modules qui implémentent les protocoles TCP/IP (IP, ICMP, UDP, and TCP) d'autres modules sont implémentés. Ces modules sont : la couche émulateur de système d'exploitation (operating system emulation), les gestionnaires des buffers et des mémoires, les fonctions d'interface réseau, et les fonctions pour le calcul des sommes de vérification (checksum), LwIP inclue aussi une interface API (Figure 3.8).

3.3.2.2.2 Le modèle de fonctionnement : LwIP utilise un modèle de processus dans lequel tous les protocoles situent dans un processus unique et sont donc séparés du noyau du système d'exploitation. Les programmes d'application peuvent situer dans un processus LwIP, ou dans des processus séparés. La communication entre la pile TCP/IP et les programmes d'application sont effectués soit par des appels de fonction pour le cas où les programmes d'application partage le processus avec la pile LwIP, ou par une façon abstraite avec interface API.

3.3.2.2.3 La couche émulateur système d'exploitation : Pour rendre la pile LwIP portable, les fonctions d'appelle du système d'exploitation et les structures de données ne sont pas directement utilisés. Lorsque ces fonctions sont nécessaires la couche émulateur de système d'exploitation est utilisée. Cette couche fournit une interface uniforme aux services du système d'exploitation tels que les temporisateurs, la synchronisation de processus, et le mécanisme de passage des messages.

3.3.2.2.4 La gestion de buffer et de mémoire : La mémoire et la gestion des buffer doivent être préparés pour qu'ils réponde au besoin des déférentes tailles de buffer allant d'un buffer qui contient une segment TCP complète, a un petit rappelle écho ICMP de quelque octet.

a) Paquets buffers | pbufs : Le *pbuf* c'est une représentation interne des paquets dans la pile lwIP, et il est conçu spécialement pour les besoin d'une implémentation TCP/IP minimale. Leur structure peut support les deux modes soit l'allocation dynamique de la mémoire pour sauvegarder le contenu des paquets, soit l'enregistrement des paquets des données dans une mémoire statique.

la figure 3.9 représente le type ***PBUF_ RAM*****,** dont les paquets sont sauvegardées dans une mémoire contrôlée par le sous-système *pbuf*, la figure 3.10 donne un exemple d'une chaine *pbuf*, ou la première *pbuf* dans la chaine est de type ***PBUF_ RAM*** et la deuxième est de type ***PBUF ROM***, qui veut dire qui a des données situer dans la mémoire non contrôlée par le sous système *pbuf.*, le troisième type ***PBUF POOL*** est dans la figure 3.11 est composé d'une pbuf de taille fixe. Une chaine pbuf est composée de plusieurs type *pbuf.*

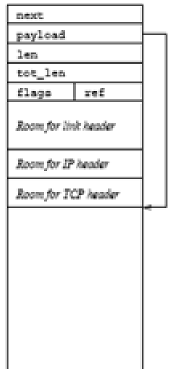

FIG. 3.9 – Pbuf PBUF_RAM avec une mémoire contrôlée par le sous-système pbuf

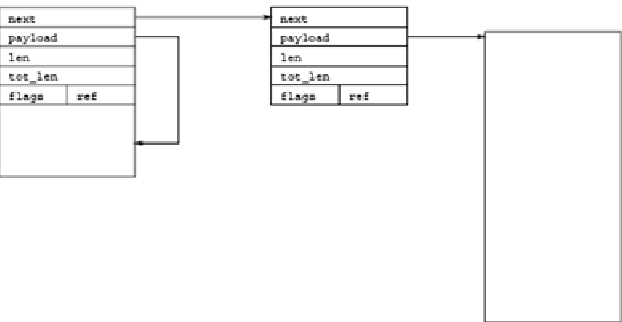

FIG. 3.10 – Une chaine *pbuf*, dont la premiere est de type ***PBUF_ RAM*** et la deuxième est de type ***PBUF ROM***

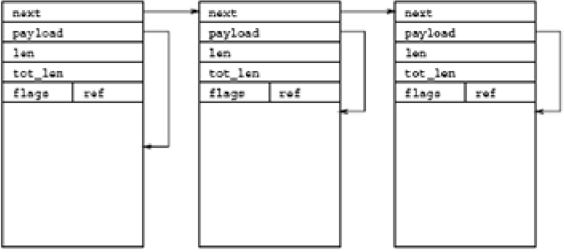

FIG. 3.11 – Une chaine pbuf de type PBUF POOL

Les trois types ont différentes utilisations. Le *pbuf* de type ***PBUF_ POOL*** est principalement utilisé par les pilotes du module réseau puisque les opérations d'allocation d'une seule pbuf est plus rapide, et donc appropriée pour un gestionnaire d'interruption. Le type ***PBUF_ ROM*** est utilisé lorsqu'une application envoie les données d'une mémoire contrôlée par l'application. Ces données ne peuvent être

modifiées après qu'ils sont remis à la pile de TCP/IP par le *pbuf*, donc ce type est principalement utilisé dans le cas où les données sont dans une mémoire ROM (d'où le nom ***PBUF_ROM***). Les en-têtes qui sont ajoutés au début des données d'un ***PBUF_ROM*** sont stockés dans un pbuf de type ***PBUF RAM*** qu'est enchaîné avec le debut d'un pbuf de type ***PBUF ROM***, comme dans la figure 3.10.

b) Gestion de la mémoire : Le gestionnaire de mémoire qui support le schéma pbuf est très simple, il manipule des allocations et dé-allocations des régions mémoires contiguës et peut rétrécir la taille d'un bloc mémoire déjà alloué. Le gestionnaire de mémoire utilise une portion dédie de la mémoire total du système. Ceci s'assure que le système réseau n'utilise pas tous la mémoire disponible, et que l'exécution d'autres programmes n'est pas gêné si le système réseau utilise la totalité de ca mémoire.

Les fonctions de gestion de mémoire sont :

```
void *mem_malloc(mem_size_t size) ;

void mem_free(void *mem) ;

void *mem_realloc(void *mem, mem_size_t size) ;

void *mem_reallocm(void *mem, mem_size_t size) ;
```

3.3.2.2.5 Le fonctionnement IP : LwIP implémente juste les fonctionnalités de base de la couche IP, il peut envoyer, recevoir et « *forwarder* », mais ne peut pas envoyer et recevoir des segments IP fragmentés ou manipule les options IP. Pour la plupart des applications ce ne pose aucun problème.

3.3.2.2.6 Le fonctionnement UDP : UDP est un protocole simple utilisé pour démultiplexer des paquets entre des processus déférents. L'état de chaque session UDP est maintenu dans une structure PCB comme montré dans la figure 3.12.

La structure PCB de UDP contient un pointeur pour la PCB suivante pour assurer une liste globale des liens des PCB UDP.

La session UDP est définie par les adresses IP et les numéros du port des points de communication, qui sont enregistrés dans les champs ***local_ip***, ***dest_ip***, ***local_port*** and ***dest_port***. Le champ flags indique quel type de *chechsum* (somme de vérification) est utilisé dans cette session.

Les deux derniers arguments ***recv*** et ***recv_arg*** sont utilisés lorsqu'un datagramme est reçu dans la session spécifié par la PCB. La fonction pointée par ***recv*** est appelée lorsqu'un datagramme est reçu.

```
struct udp_pcb {
  struct udp_pcb *next;
  struct ip_addr local_ip, dest_ip;
  u16_t local_port, dest_port;
  u8_t flags;
  u16_t chksum_len;
  void (* recv)(void *arg, struct udp_pcb *pcb, struct pbuf *p);
  void *recv_arg;
};
```

FIG. 3.12 – Structure udp_pcb

À cause de la simplicité du UDP, le traitement des entrés et des sorties est simple et suit une ligne assez droite (Figure 3.13). Pour envoyer les données, le programme d'application appelle à la fonction ***udp_ send()*** qui fait appelle a la fonction ***udp_ output().*** A ce niveau les somme de vérification sont calculés (*checksum*) et les en-têtes UDP sont remplis. Puisque le calcule des sommes de vérification utilise l'adresse IP de la source du paquet UDP, la fonction ***ip_ route()*** est appelée dans certain cas pour trouver l'interface réseau au quelle les paquets doit transmis. L'adresse IP de cette interface réseau est utilisé comme l'adresse source du paquet. Finalement le paquet est retourné à ***ip_ output_ if()*** pour la transmission.

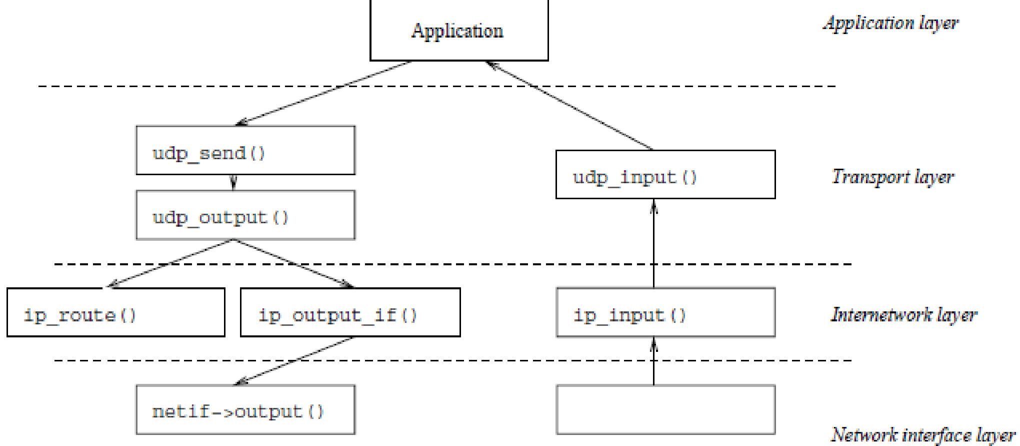

FIG. 3.13 – Le fonctionnement UDP

Lorsque un datagramme UDP arrive, la couche IP appelle la fonction ***udp_ input()***. A ce niveau la somme de vérification est testé et le datagramme est « *demultiplixé* », lorsque la PCB correspondante est trouvé la fonction ***recv()*** est appelé.

3.3.2.3 Module SNMP

Avant de passer a l'environnement EDK il faut valider l'exécution de notre modélisation de l'agent SNMP dans l'environnement TauSDL, l'outil qui va nous permettre de générer une application exécutable est Targeting Expert.

Adéquation SDL-EDK :

Targeting Expert : Les deux fonctions essentielles de l'outil Targeting sont :

- Génération du code C à partir de la modélisation SDL par : Cadvanced SDL to C Compiler.
- Configuration des options du compilateur selon l'environnement et le compilateur utilisé.

Cadvanced SDL to C Compiler : Pour obtenir un programme exécutable qui se comporte selon la description SDL, la description SDL est passée par un analyseur SDL, qui contient le Cadvanced SDL to C Compiler. Si la description de SDL est syntactiquement et sémantiquement correcte, un programme C est généré. Ce programme doit compile en utilisant un compilateur C, et le lies avec une bibliothèque d'exécution prédéfinie de SDL pour former un programme exécutable (Figure 3.14).

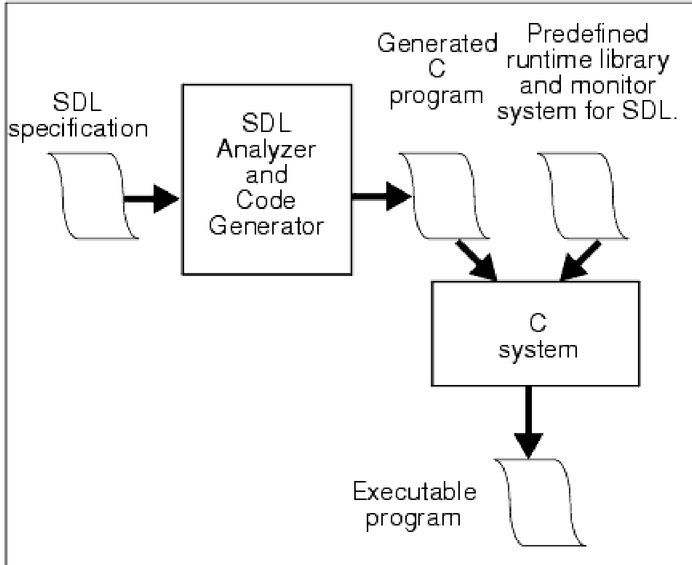

FIG. 3.14 – Génération d'un fichier exécutable sous SDL

Les fichiers générés par TauSDL : Les fichiers qui sont générés et qui seront utilisés pour créer l'application software dans EDK sont :

- ***Component.c*** : est le fichier qui représente la description SDL
- ***Component_ env.c*** : le fichier d'environnement. Ce fichier doit être modifié pour adapter la description SDL à l'environnement cible.
- ***RFC1155_ SMI .c, RFC1155_ SMI.h, RFC1155_ SMI.ifc, RFC1155_ SMI_ asn1coder.c RFC1155_ SMI_ asn1coder.h*** : les fichiers qui contiennent les définitions et les déclarations définies dans la RFC1155 SMI (Structure of Managment Information).
- ***RFC1157_ SNMP.c, RFC1157_ SNMP.h, RFC1157_ SNMP.ifc, RFC1157_ SNMP_ asn1coder.c, RFC1157_ SNMP_ asn1coder.h*** : les fichiers qui contiennent les définitions et les déclarations définies dans la RFC1157.
- ***Sctsdl.h :*** ce fichier représente le fichier main dans la structure d'un programme en C, il contient aussi les opérations SDL tel que : Output, Create, Stop, Nextstate, Set, Reset.
- ***Sctlocal.h*** : Ce dossier contient les définitions et déclarations externes des variables et des fonctions qui sont employées seulement par le noyau. Ce fichier n'est pas inclus dans le code généré.
- ***Sctos.c*** : ce fichier contient quelques fonctions qui représentent la dépendance entre le Hardware, le software, système d'exploitation, les fonctions de lecture de l'horloge, les fonctions de l'allocation mémoire.effectués

– ***Sctpred.h*** : Ce fichier contient les définitions et les déclarations externes manipulant les données prédéfinies par SDL (excepté de type PId, qui est dans scttypes.h). Ce dossier est inclus en code généré par l'intermédiaire de scttypes.h.

– ***Scttypes .h*** : ce fichier contient les définitions et déclarations des variables et des fonctions externes. Le fichier est inclus par sctsdl.c, sctpred.c, sctos.c, et par chaque fichier C généré.

L'environnement de l'application : Pour qu'on puisse adapter le code généré par le Targeting, il faut comprendre la structure de ce code.

Le modèle de fonctionnement du code généré se compose de trois paramètres (Figure 3.15)

1. **Le système SDL** : ce qui représente dans notre cas, le modèle de l'agent SNMP proposé, qui communiquent avec l'environnement par des signaux d'entrée et de sortie ;
2. **L'environnement physique du système** : qu'on peut qualifier par la cible, dans notre cas c'est le processeur MicroBlaze et la pile protocolaire ;
3. **Les fonctions d'environnements** : qui permettent l'adaptation entre le système SDL et l'environnement.

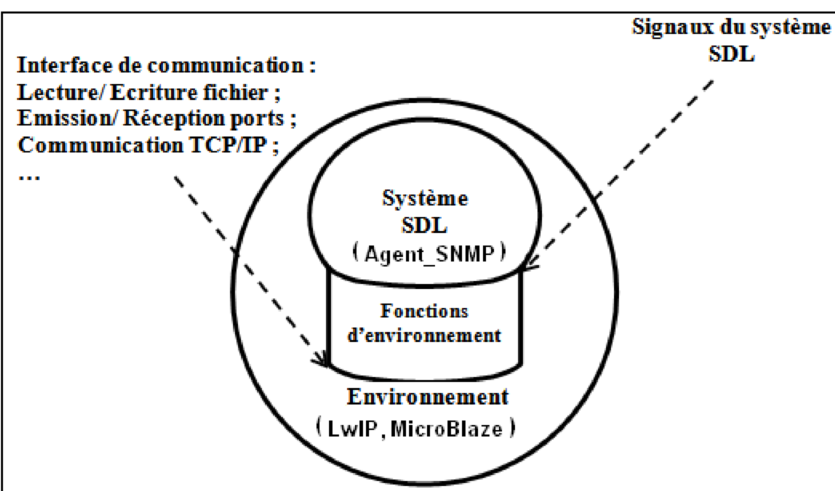

FIG. 3.15 – Schéma d'environnement d'un système SDL

Les fonctions d'environnement se trouvent dans le fichier component_env.c. Chaque fonction joue un rôle spécifique :

- ***xInitEnv*** : appelée lors de l'initialisation de notre système ;
- ***xCloseEnv*** : appelée lorsqu'on veut arrêter l'exécution de notre système ;
- ***xOutEnv*** : appelée à chaque fois qu'un signal sortant de notre système est émis ;
- ***xinEnv*** : appelée à chaque fois qu'un signal est émis de l'environnement vers le système.

Nous devrons, ajouter du code C dans ce fichier, au niveau de chaque fonction, pour adapter notre système a l'environnement qu'est la pile LwIP.

La structure du code généré : Le code généré contient deux types importants des fonctions, des fonctions d'initialisation et des fonctions de PAD[4] .Les fonctions PAD implémentent les actions exécutées par les processus SDL pendant les transitions.

Il y aura une fonction d'initialisation dans chaque fichier C généré. Dans le fichier qui représente le système SDL cette fonction est nommée "yInit". Et la fonction qui initialise l'environnement est la fonction "xInitEnv" définie dans le fichier d'environnement.

Chaque processus dans le système sera représenté par une fonction PAD, appellée quand un processus SDL exécute une transition.

L'exemple ci-dessous (Figure 3.16) montre la structure principale, les fonctions ***MainInit()***, et ***MainLoop()***.

```
void main ( void )
{
xMainInit();
xMainLoop();
}

void xMainInit ( void )
{
xInitEnv();
Init of internal data structures in the
runtime library;
yInit();
}

void xMainLoop ( void )
{
while (1) {
xInEnv(...);
if ( Timer output is possible )
SDL_OutputTimerSignal();
else if ( Process transition is possible )
Call appropriate PAD function;
}
}
```

Fig. 3.16 – Structure du code généré

Adaptation du fichier d'Environnement : Pour adapter le système SDL à l'environnement in faut intégrer les fonctions de la plateforme software aux fonctions d'environnement dans le fichier d'environnement « ***component_ env.c*** ».

Les modifications doit se faire comme suit :

a) Dans la fonction d'initialisation ***xInitEnv***

Les fonctions d'initialisation de la plateforme software et les fonctions d'initialisation de la pile protocolaire doit s'initialisent lors de l'initialisation du système c'est pour ca il faut l'inclurent dans la fonction d'initialisation *xInitEnv*.

[4]Process Activity Definition

C'est fonctions sont :

- ***init_ platform*** : c'est la fonction d'initialisation de la plateforme software

- ***microblaze_ enable_ interrupts*** : cette fonction permet d'activer la gestion des interruptions du processeur MicroBlaze.

- ***lwip_ in***it ;c'est la fonction d'initialisation de la pile LwIP,

L'appelle a cette fonction fait l'initialisation la structure des données de la pile LwIP, elle remplace l'appelle aux fonctions initialisation : de l'état, du système, du mémoire, et des couches ARP, IP, UDP, et TCP.

- ***mac_ ethernet_ address[]*** = { 0x00, 0x0a, 0x35, 0x00, 0x01, 0x02 } ;

Cette fonction permet d'affecter une adresse MAC de l'interface réseau

- ***IP4_ ADDR***(&ipaddr, 192, 168, 1, 10) ;

- ***IP4_ ADDR***(&netmask, 255, 255, 255, 0) ;

- ***IP4_ ADDR***(&gw, 192, 168, 1, 1) ;

Ces fonction permet d'affecter l'adresse IP, les sous masques réseau et l'adresse de routage à l'interface réseau.

- ***udp_ new*** ; Crée un nouvel identificateur de connexion UDP (PCB[5]) qui peut être utilisé pour établir une communication UDP. Le PCB n'est pas activité jusqu'à ce qu'il été connectée à une adresse locale ou une adresse à distance.

- ***udp_ bind*** : connecte le PCB à une adresse locale.

- ***udp_ connect*** : cette fonction détermine l'extrémité distance du PCB. Cette fonction ne produit d'aucun trafic de réseau, mais a seulement place l'adresse distance du PCB.

- ***unsigned source_ port*** *= 162 ;*

- ***unsigned dist_ port*** *= 161 ;*

Ces deux paramètres permettent d'affecter les numéros des protes UDP source et destination, dans notre cas le port 161 est utilisé par l'agent pour recevoir les requêtes de la station de supervision, et le port 162 est utilisé pour répondre.

b) Dans la fonction d'entrée ***xInEnv***

Les fonctions qui permettent de transmetre les datagrammes UDP au système SDL sont installées ici, à savoir ;

- ***udp_ recv*** : cette fonction fait appel à une fonction "udp_recv_callback" qui doit s'exécuter lorsqu'un datagramme UDP est reçu.

- ***udp_ recv_ callback*** : cette fonction s'exécute lorsqu'un datagramme UDP est reçu et doit transmettre les signaux d'entrée au système SDL (agent SNMP). Elle utilise les fonctions d'entrée du système SDL (du code SDL) qui sont :

[5]Connection Contrôle Block

o ***xGetSignal*** : utilisé pour obtenir un secteur de données convenable pour la représentation du signal dans le système SDL.

o ***SDL_ Output*** : transmet le signal au processus récepteur selon les règles sémantique de SDL.

c) Dans la fonction de sortie ***xOutEnv*** ;

Les fonctions d'émission UDP sont incluses ici, ces fonctions envoient les trames réseau après le traitement de l'agent SNMP.

-***udp_ send*** : cette fonction envoie les données qui sont dans le buffer (pbuf), mais la mémoire associée n'est pas déallouée.

d) Dans la fonction de fermeture ***xCloseEnv*** on doit fermer la connexion UDP et quitter l'application.

Les fonctions implémentées sont ;

- ***udp_ disconnect*** : enlever l'extrémité distante du PCB. Cette fonction ne produit d'aucun trafic de réseau, mais enlève seulement l'adresse distante du PCB.

- ***udp_ remove :*** enlève et désactive le PCB.

3.3.2.4 Initialisation du software

Après que la plateforme hardware est créée et le développement de la partie software à intégrer est terminée, on peut choisir les différentes méthodes pour faire fonctionner l'ensemble :

3.3.2.4.1 Application dans la BRAM : On peut intégrer l'application dans le bitstraem généré. Dans ce cas, la BRAM s'initialise avec le fichier exécutable ELF (Executable and Linkable Format). Il sera chargé dans la mémoire du système, et prêt a être exécuté, chaque fois que l'FPGA est configuré.

Cette configuration est possible si le code exécutable est de petite taille (inférieur de la taille de la BRAM).

3.3.2.4.2 Bootloop : Au cours du prototypage, XPS peut dynamiquement télécharger le fichier exécutable dans le FPGA via le câble JTAG connecté à la carte. Dans ce cas, on peut sélectionner un bootloop à intégrer dans le bitstream (initialise la BRAM) pour initialiser la mémoire du système afin que le processeur reste dans un état statique jusqu'à ce que le téléchargement du software soit terminé.

Après que l'FPGA est configuré avec le bitstream, le processeur commence à s'exécuter. Si le système n'est pas initialisé avec aucune application, le processeur pourrait exécuter le code incorrect qu'initialise la mémoire, et met le processeur dans un état à partir duquel on peut pas sortir de l'étape de Reset

Le bootloop est une application logicielle qui maintient le processeur dans un état défini jusqu'à ce que l'application réelle puisse être téléchargée et exécutée. Elle comprend une simple instruction de branchement localisée dans le secteur boot du processeur. XPS contient une simple application bootloop qu'on a utilisée.

3.3.2.4.3 Bootloader : Pour les systèmes de production, le fichier exécutable peut être stocké dans une région mémoire morte comme ROM ou PROM. Dans ce cas, on configure un bootloader exécutable a intégrer dans le bitstream pour initialiser la mémoire du système, et chaque fois le FPGA est configuré ou redémarré le bootloader copie l'application exécutable dans une mémoire vive de le FPGA, et commence a s'exécuter.

La figure 3.17 montre un exemple d'une application MicroBlaze stockée dans une mémoire flash (SPI serial Flash) et qui est téléchargée après le démarrage dans une mémoire DDR et s'exécute. Les étapes sont :

1. Le FPGA est configuré à travers une mémoire Flash (SPI serial Flash) avec le bitstream qui est composé de la plateforme hardware et la BRAM initialisée avec l'application bootloader.
2. Le bootloader accède à un espace mémoire prédéterminé dans la mémoire flash et copie l'application à exécuter de la mémoire flash a la mémoire DDR.
3. Le bootloader se déroule ensuite vers la mémoire DDR et commence à exécuter l'application.

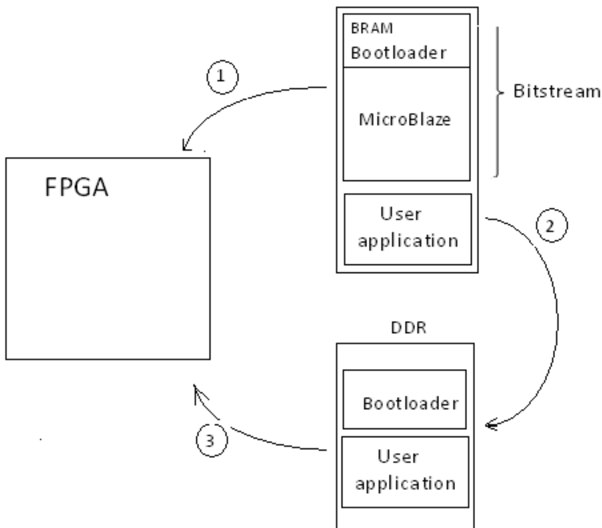

FIG. 3.17 – Application MicroBlaze avec un bootloader

3.4 Résultat de mise en œuvre

Pour mettre en œuvre notre agent SNMP, la carte FPGA doit être configurée avec l'agent SNMP. Une connexion Ethernet doit être établie entre un PC contenant un manager SNMP et la carte Spartan-3E [36] (Figure 3.18).

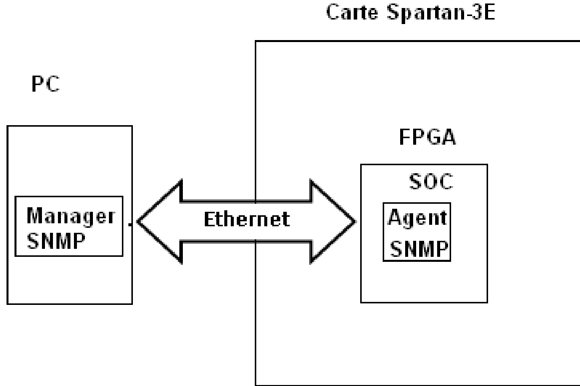

Fig. 3.18 – Illustration de la communication établie entre un PC contenant le Manager SNMP et la carte Spartan-3E.

On a rencontré un problème d'incompatibilité entre le code généré par la version 4.4 de TauSDL et le compilateur «mb-gcc» d'EDK lorsqu'on intègre les fonctions de décodage ASN.1, On n'a donc pas pu implémenter ces fonctions dans notre système. Cette incompatibilité est résolue dans la version 6 ou plus (TausSDL 6.0 ou 6.2). Comme cette version n'est pas à notre portée, on a validé notre développement en deux étapes :

1. Valider le fonctionnement du système compllet de l'agent SNMP.
2. Valider le fonctionnement l'architecture hardware et software tenant compte de la pile protocolaire.

3.4.1 Première Partie

pour mettre en évidence le fonctionnement de notre système complet (agent SNMP), on a utilisé le simulateur de l'environnement de développement TauSDL.

Ce simulateur nous permet de voir d'une manière graphique et très intuitive le comportement de notre système en utilisant l'éditeur MSC (message sequence chart). La simulation consiste à envoyer une requête SNMP de type get-request à partir de l'environnement et récupérer la réponse de l'agent SNMP avec une trame get-response.

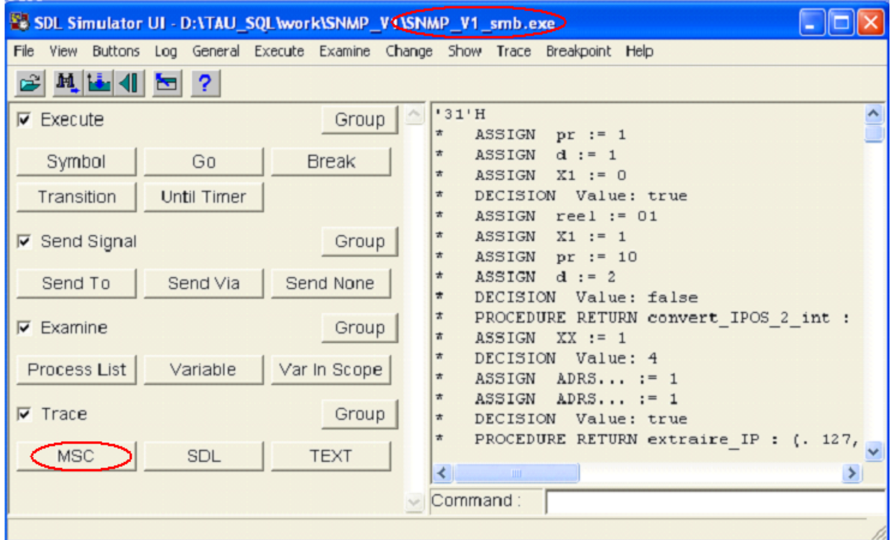

FIG. 3.19 – Capture d'écran de l'interface graphique du simulateur

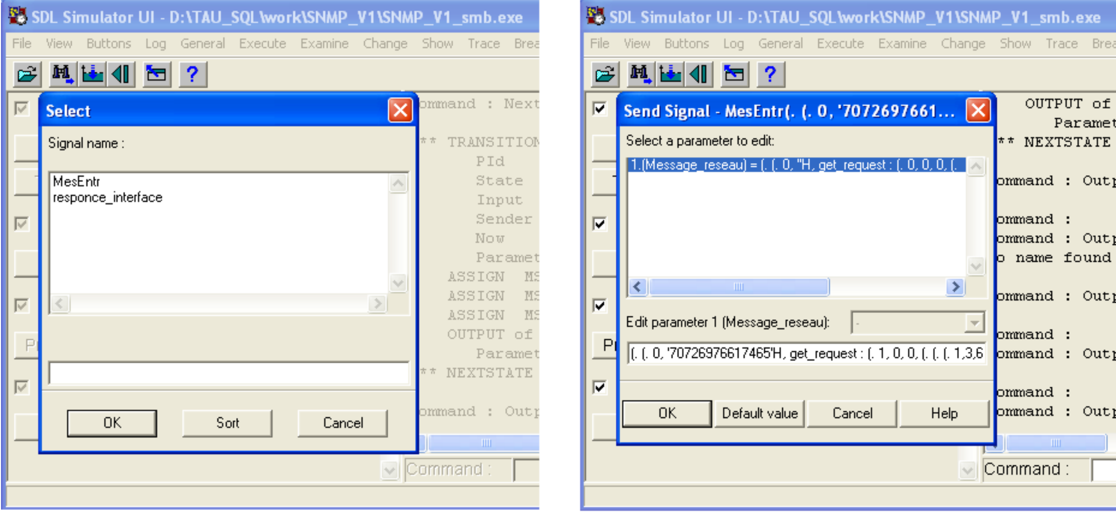

FIG. 3.20 – Capture d'écran de l'interface graphique du simulateur

La figure 3.22 donne les séquences de fonctionnement dérivées du diagramme MSC en ordre synchrone de haut vers le bas.

Ce digramme a été déroulé en envoyant une trame SNMP arbitraire de type get-request suivant :

(. (. 0, '70726976617465'H, get_request : (. 1, 0, 0, (. (. (. 1,3,6,4 .) ,simple : number : 3 .).) .) .), '3132372E302E302E31'H .)

Cette trame est codée en ASN.1 avec la représentation du TauSDL, il est équivalent à :

GetRequest, GetNextRequest, GetResponse or SetRequest PDU

Version	Community	PDU Type	Request I D	Error Status	Error Index	Object 1, Value 1	
						OID	Value
0	70726976617465'H	get -request	1	0	0	1.3.6.4	.3.

FIG. 3.21 – Trame équivalente

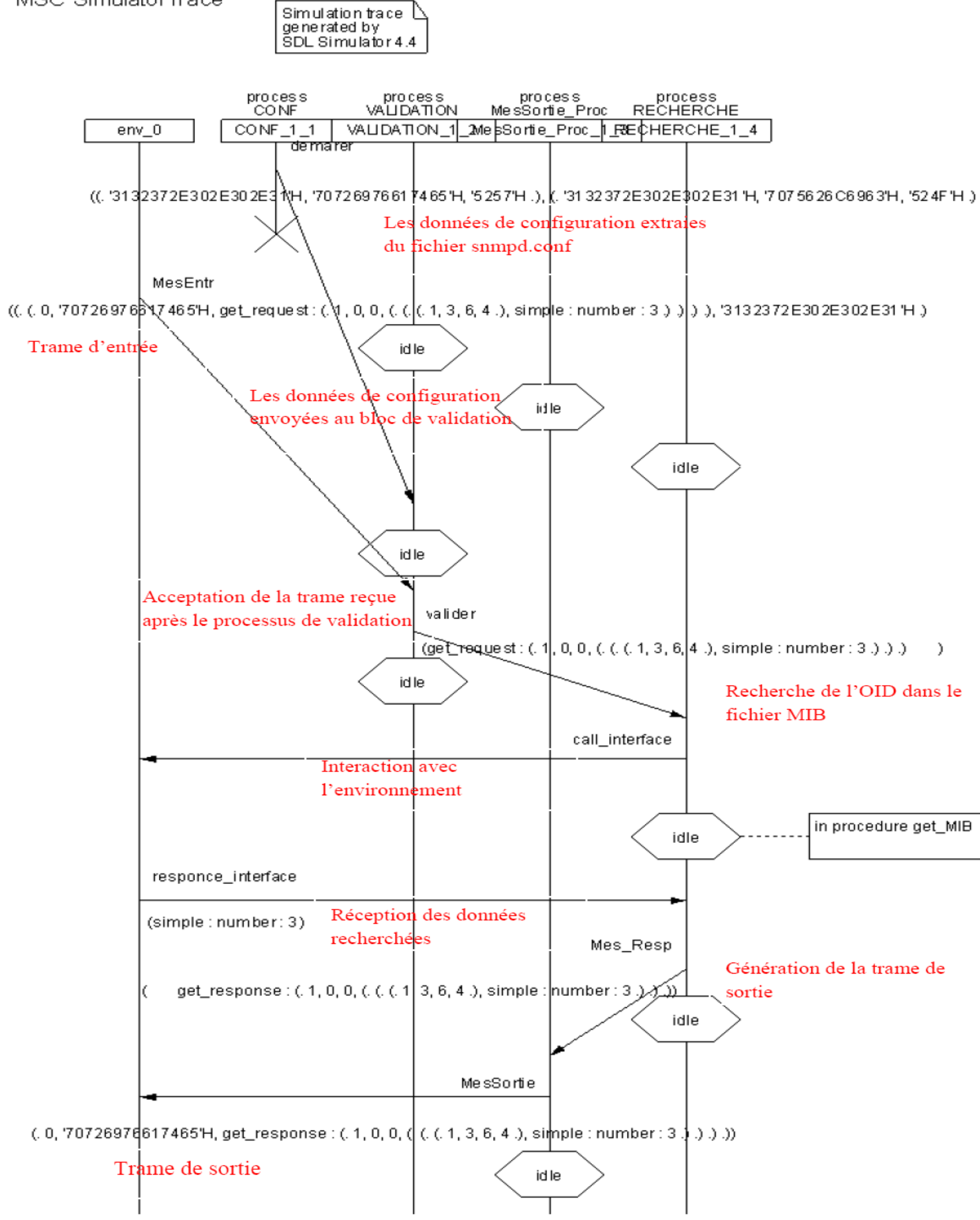

Fig. 3.22 – Trace de fonctionnement à travers le diagramme MSC

3.4.2 Deuxième partie

pour mettre en évidence le fonctionnement de notre architecture hardware et software tenant compte de la pile protocolaire, on a implémenté un simple système modélisé par SDL sans les fonctions de décodage ASN.1, (Figure 3.23).

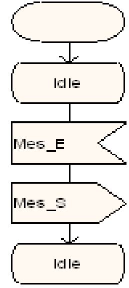

FIG. 3.23 – Processus du système SDL

Le système est composé d'un simple processus. A la réception d'un signal d'entrée qui est dans notre cas une simple trame SNMP (get-request par exemple), le système répond avec une trame avec les mêmes données de la requête transmit. Une requet aléatoire de type Get-request est générée à partir du manager (Figure 3.24) :

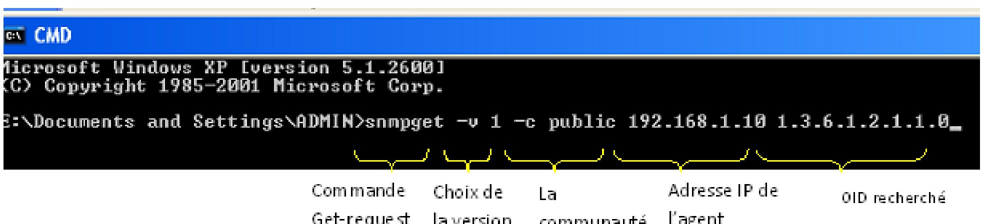

FIG. 3.24 – Commande « get-request » à partir du manager SNMP

La réponce de l'agent SNMP implémenté dans la carte FPGA capturé par l'outil de diagnostique réseaux « ***Wireshark*** » est donnée par la figure suivante :

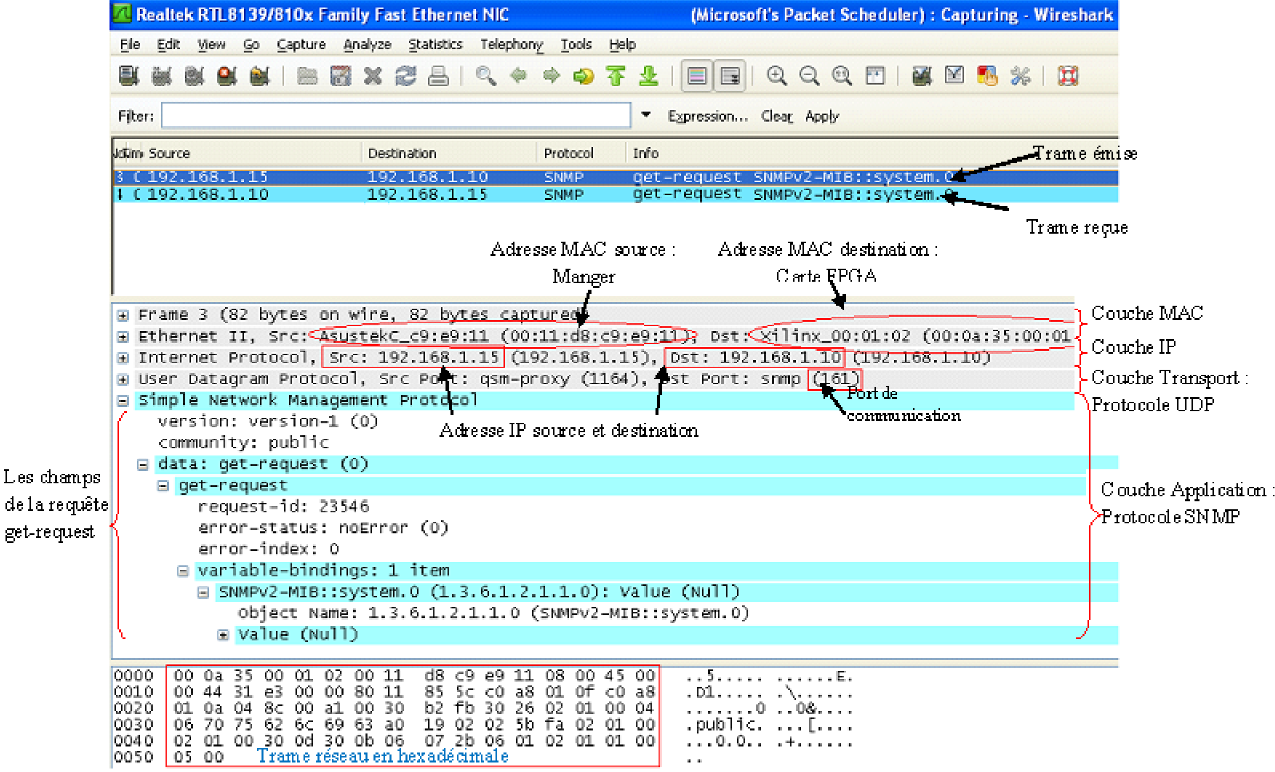

FIG. 3.25 – Capture des trames réseaux par l'outil « Wireshark »

Conclusion

A travers ce chapitre, nous avons présenté la méthodologie de conception des systèmes sur puce utilisée pour générer notre IP core Agent SNMP. Nous avons donné les deux architectures de notre système qui font parties du flot de conception, à savoir l'architecture hardware et l'architecture software.

Le code C généré par l'environnement de développement TauSDL décrit dans le chapitre précédent est adapté, pour être exploité dans la génération de l'application software dans l'environnement SDK d'EDK.

En fin les résultats de mise en œuvre ont validé le modèle proposé.

Conclusion générale

L'évolution de la technologie des semi-conducteurs et des outils de conception des systèmes sur puce permet de placer des dizaines de microprocesseurs avec des circuits annexes sur une seule puce de silicium. Ces circuits s'avèrent donc de plus en plus complexes. La nécessité de leur supervision à l'échelle du composant ou du système s'impose ainsi de fait.

Dans ce mémoire nous nous sommes inspirés des réseaux informatiques pour transposer les outils utilisés dans ce but. Une solution a été proposée concernant le développement d'un IP CORE autour d'un processeur soft MicroBlaze et une pile protocolaire LwIP, dédié à la supervision d'un SoC, basé sur un protocole de gestion de réseaux simple SNMP.

Pour ce faire, la recherche d'une méthodologie adaptée est plus que nécessaire. Nous avons opté au co-design dans lequel la partie software est déduite d'une modélisation préalable conduisant à un niveau d'abstraction élevé.

A travers cette méthodologie, l'objectif de notre développement a été atteint et un IP core agent SNMP a été déduit. Il peut être intégré dans un système sur puce pour réaliser la fonction de supervision avec un protocole relativement simple et des fonctionnalités suffisamment puissantes.

On soulignera le manque de sécurité évident qui subsiste sur la première version de SNMP (SNMPv1). La version Trois (SNMPv3) peut être implémentée dans notre IP Core pour améliorer la sécurité à travers l'ajout d'un module de securité.

Le langage de spécification et de description formel SDL épaulé par l'environnement de développement TauSDL de Telelogic se trouve tout à fait approprié pour la spécification de notre système.

L'association de deux environnements de développement à savoir TauSDL de Telelogic et EDK de Xilinx permet la déduction d'un nouvel environnement à travers lequel nous avons pu modéliser notre agent SNMP et générer un code C pour l'utilisation dans le flot de conception de la partie software du co-design. Cet environnement facilite aussi la création de la plateforme hardware en utilisant d'autres IP core.

Cette méthode peut être appliquée à n'importe quelle application ciblant les systèmes sur puces.

Bibliographie

[1] Éric Piel, « Ordonnancement de systèmes parallèles temps-réel : De la modélisation à la mise en oeuvre par l'ingénierie dirigée par les modèles », Université des Sciences et Technologies de Lille, 2007.

[2] http://www.net-snmp.org/.

[3] Patrice KADIONIK, « l'administration de réseau : mise en œuvre de net-snmp », ENSEIRB.

[4] J. Case, M. Fedor, M. Schoffstall, and J. Davin. RFC 1157 « A Simple Network Management Protocol (SNMP) ». Internet Engineering Task Force (IETF) - Request for Comments, 1157. Obsoletes : RFC 1098. May 1990. http://www.ietf.org/rfc/rfc1157.txt.

[5] Rose, M.T., and K. McCloghrie. « Structure and Identification of Management Information for TCP/IP-based Internets » RFC 1155, May 1990. http://www.ietf.org/rfc/rfc1155.txt.

[6] Case, J.D., K. McCloghrie, M.T. Rose, and S.L. Waldbusser, « Introduction to version 2 of the Network Management Framework » RFC 1441, April 1993. http://www.ietf.org/rfc/rfc1441.txt.

[7] Case, J.D., K. McCloghrie, M.T. Rose, and S.L. Waldbusser, « Coexistence between version 1 and version 2 of the Internet-standard Network Management Framework », RFC 1452, April 1993. http://www.ietf.org/rfc/rfc1452.txt.

[8] D. Harrington, R. Presuhn, B. Wijnen, « An Architecture for Describing Simple Network Management Protocol (SNMP) Management Frameworks » RFC3411, December 2002.http://www.ietf.org/rfc/rfc3411.txt.

[9] R. Presuhn, J. Case, J. Case, M. Rose, S. Waldbusser, « Management Information Base (MIB) for the Simple Network Management Protocol (SNMP) », RFC 3418, December 2002.http://www.ietf.org/rfc/rfc3418.txt.

[10] http://www.frameip.com/.

[11] Mark A. Miller, « Managing Internetworks with SNMP », 2nd Edition, M & T Books.

[12] Douglas Mauro, Kevin Schmidt, « Essential SNMP, 2nd Edition », September 2005.

[13] Arnaud Grasset, « synthèse des interfaces de communication des systèmes monopuces : de la spécification à la génération automatique », Institut national de Grenoble, janvier 2005.

[14] Jean-Marc DAVEAU, « Spécification système et synthèse de la communication pour le co-design Logiciel / Matériel », Institut National Polytechnique de Grenoble, pages de 37 à 59, 1997.

[15] : Emmanuel Gaudin, « Langage de développement SDL et UML : mariage de raison pour la conception des logiciels temps réel », Article de Mars 2004 n°145 – Electronique.

[16] Zoubir Mammeri « SDL, Modélisation de protocoles et systèmes réactifs » Hermès Science Europe, 2000.

[17] ITU-T « Recommendation Z.120, Diagrammes des séquences de messages (MSC) », 1999.

[18] ITU-T « Recommendation Z.100, Specification and Design Language (SDL) », 1996.

[19] ITU-T « Recommendation Z.100, Specification and Design Language (SDL2000) », 1999,

[20] Telelogic, « ObjectGeode, method guidelines », April 2000, `www.telelogic.com`.

[21] Jean-Philippe Babau « Le langage SDL pour les systèmes temps réel », CITI : Centre d'Innovation en Télécommunication et Intégration de services,INSA Lyon.

[22] `www.telelogic.com`.

[23] « ITU-T Z.100 Functionnal Specifcation and Description Language, Recommandation Z.100 - Z.104 », March 1993.

[24] Olivier Dubuisson, « ASN.1 Communication between Heterogeneous Systems », June 5 2000.

[25] Telelogic TauSDL 4.0 Help.

[26] M. Robert, « CAO de systèmes sur puce », LIRMM, UMR CNRS /Université Montpellier.

[27] SDK Help, « FPGA Configuration Overview ».

[28] « EDK OS and libraries Reference Manual », Embedded Development Kit 6.3i. UG 114(V3.0), 20 aout 2004.

[29] « Microblaze processor reference guide », Embedded Development Kit 6.3. UG081(V4.0), 24 aout 2004.

[30] « Processor Local Bus (PLB) v4.6 (v1.00a) », Product Specification, DS531 August 9, 2007.

[31] « XPS Ethernet Lite MediaAccess Controller (v2.01a) », Product Specification, DS580 April 24, 2009.

[32] « XPS Timer/Counter (v1.01a) », Product Specification, DS573 April 24, 2009.

[33] « XPS Interrupt Controller (v2.00a) », Product Specification, DS572 April 24, 2009.

[34] « MicroBlaze Debug Module (MDM) (v1.00e) », Product Specification, DS641 April 24, 2009.

[35] Adam Dunkels, « Design and Implementation of the lwIP TCP/IP Stack », Swedish Institute of Computer Science, February 20, 2001.

[36] « Spartan-3E FPGA Starter Kit Board User Guide », UG230 (v1.1) June 20, 2008.

[37] ITRS, « International Technology Roadmap for Semiconductors ». Executive summary, 2005 edition 26.`http://www.itrs.net/`.

[38] Laurant DOLDI, « Validation of Telecom Systems with SDL », John Wiley & Sons Ltd, England, pp.9-24, 2003.

[39] Z.MAMMERIE, « Introduction au langage de description et de spécification », Université Paule Sabatier, Toulouse, 2001.

Annexes

Annexe A

Symboles graphiques de SDL

La grammaire graphique est décrite dans la norme ITU-T Z.100. Nous présentons, dans les figures A.1 et A.2, les différents symboles graphiques pour permettre la compréhension de système SDL décrit graphiquement.

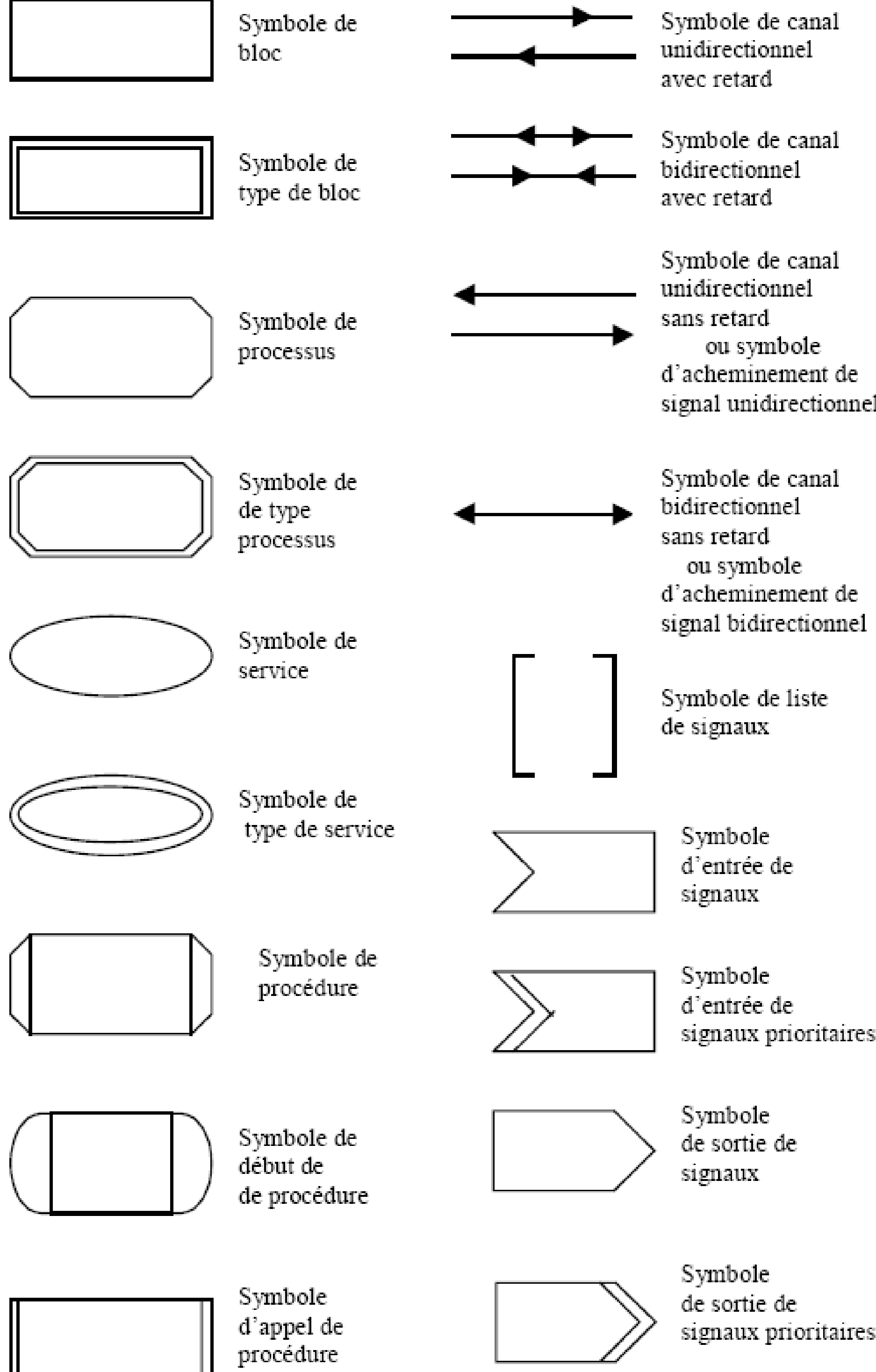

FIG. A.1 – Symboles graphiques de SDL (1)

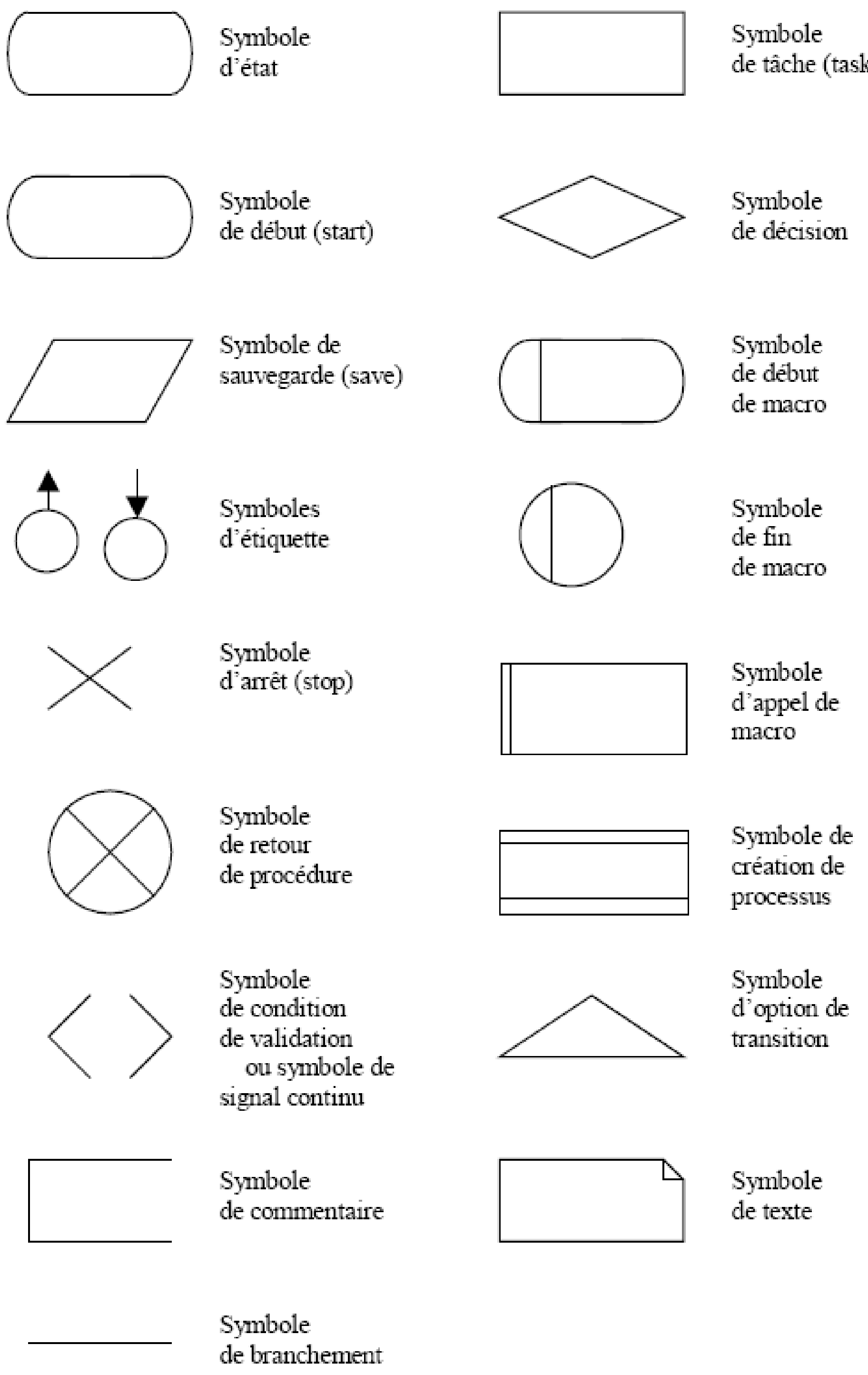

Fig. A.2 – Symboles graphiques de SDL (2)

Annexe B

Description SDL de l'Agent_SNMP

Les figures suivantes représentent la description de tout les processus et les procédures du modèle SDL de l' Agent_SNMP proposé dans le chapitre 3. Nous commencerons par donner les figures concernant le bloc CONFIGURATION, en suite le bloc ENTREE puis le bloc MIB et enfin nous terminerons par la description des processus du bloc INTERFACE et SRTIE.

Nous donnerons aussi les modules ASN.1 de la RFC1155 et la RFC1157.

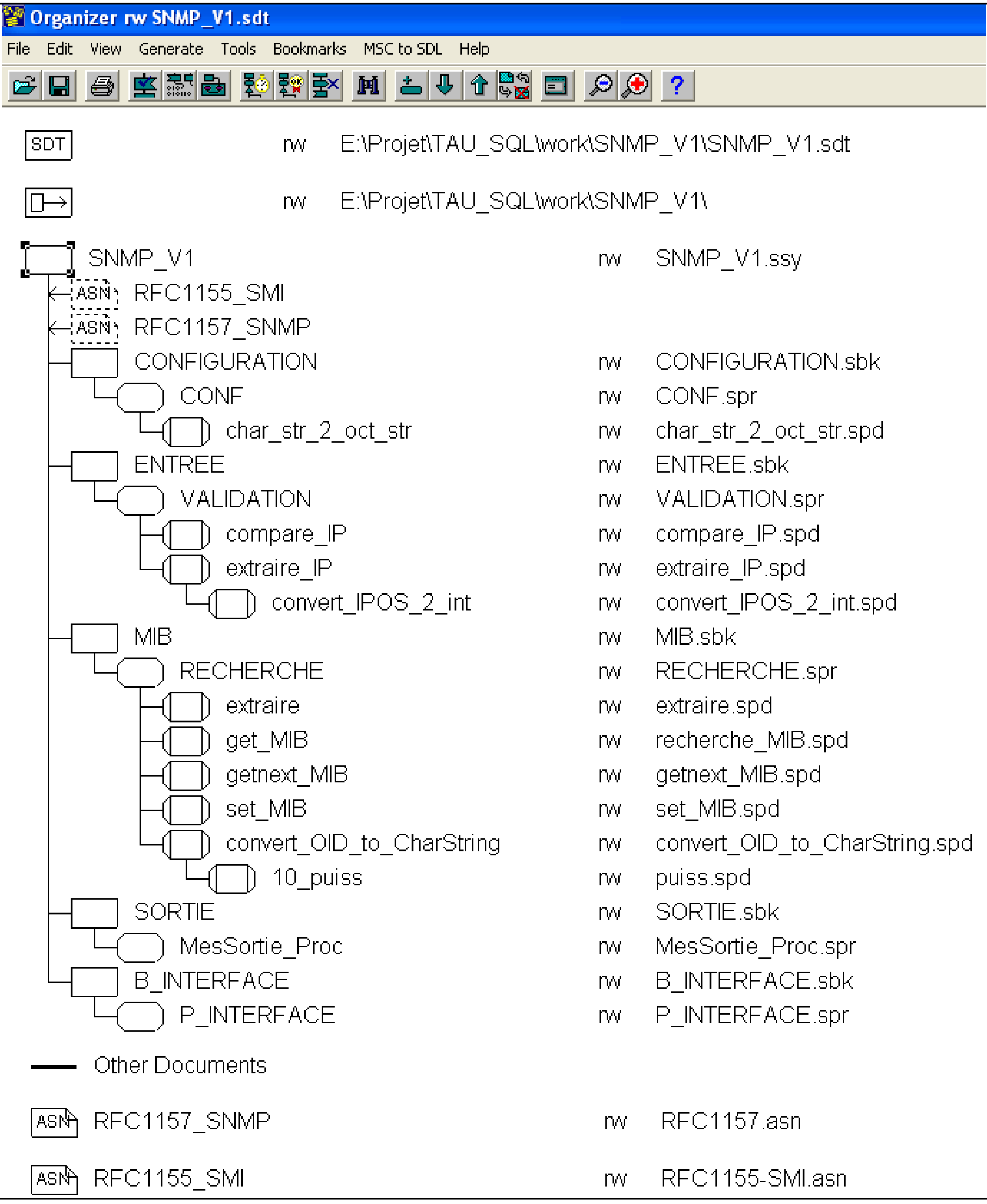

Fig. B.1 – Interface graphique de TuaSDL

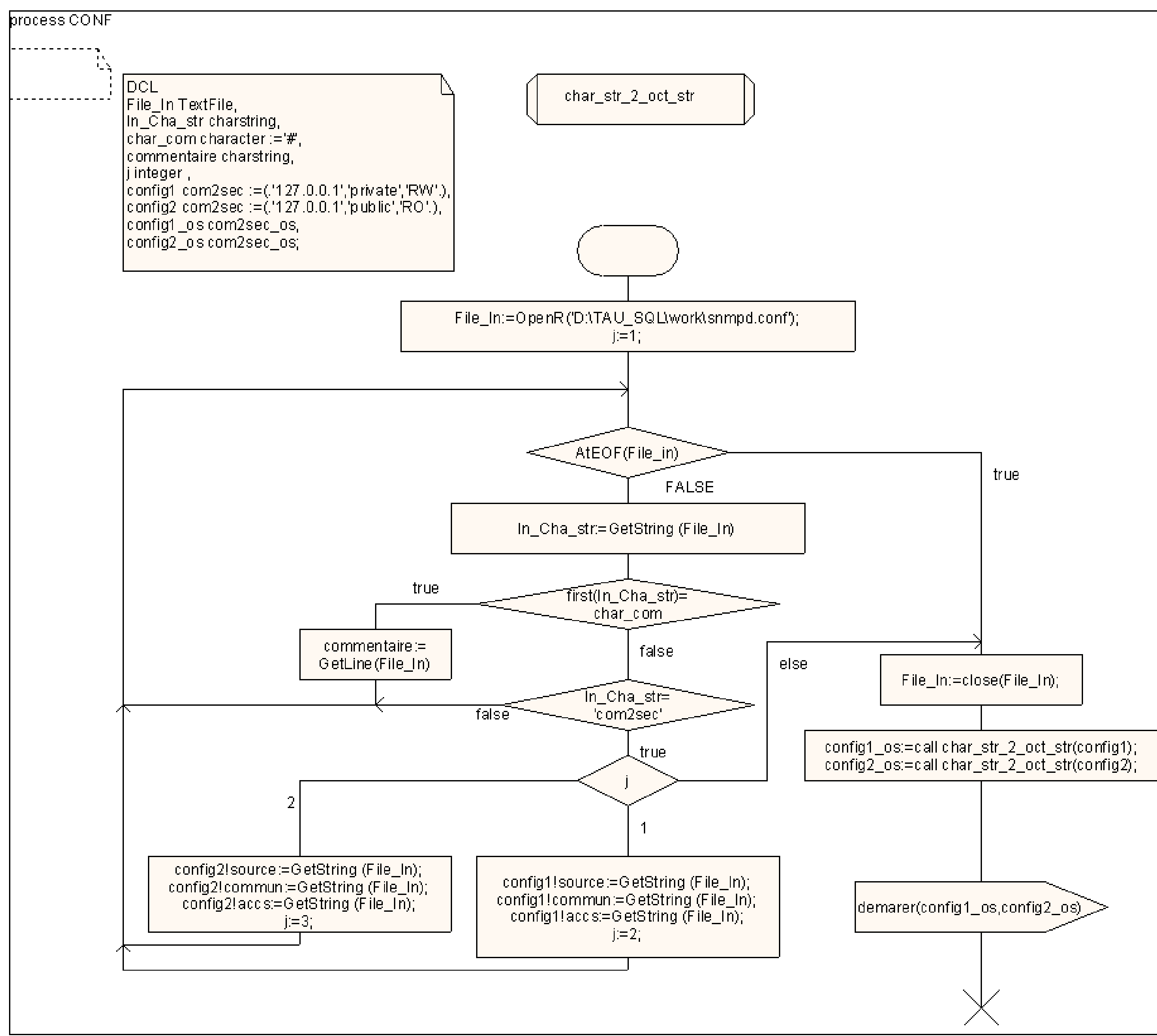

Fig. B.2 – Processus « CONF »

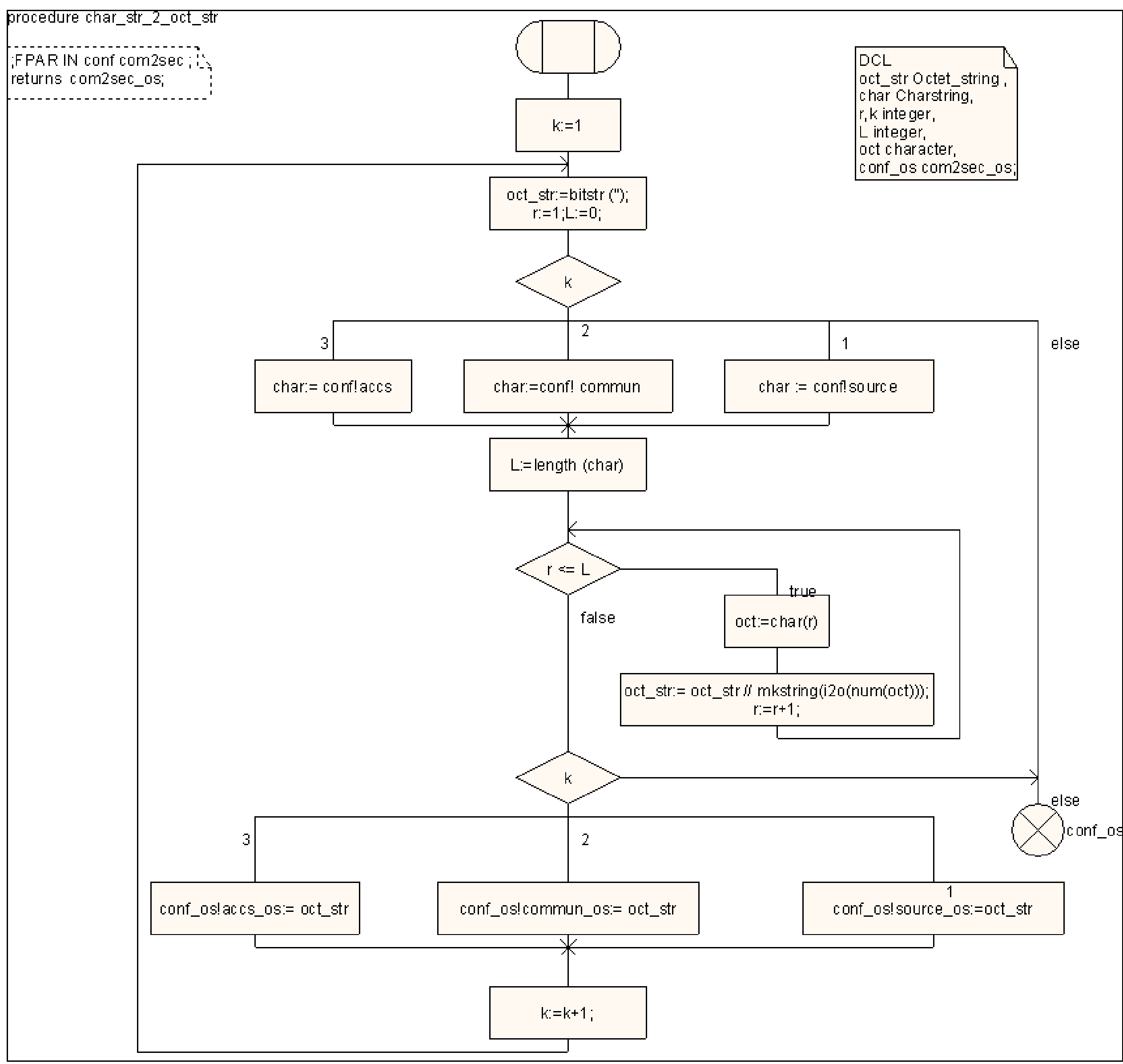

FIG. B.3 – Procédure « char_str_2_oct_str »

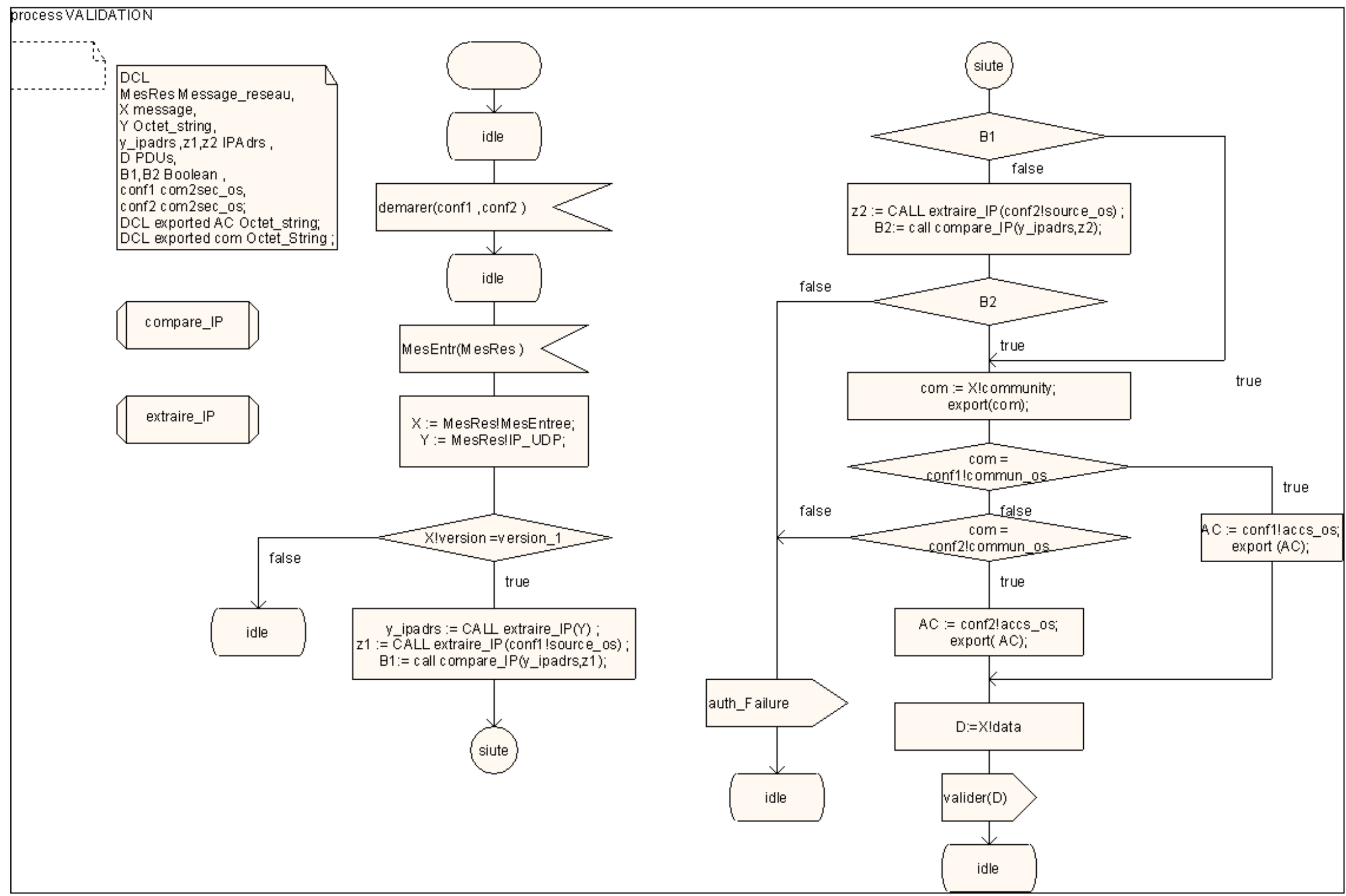

Fig. B.4 – Processus « VALIDATION »

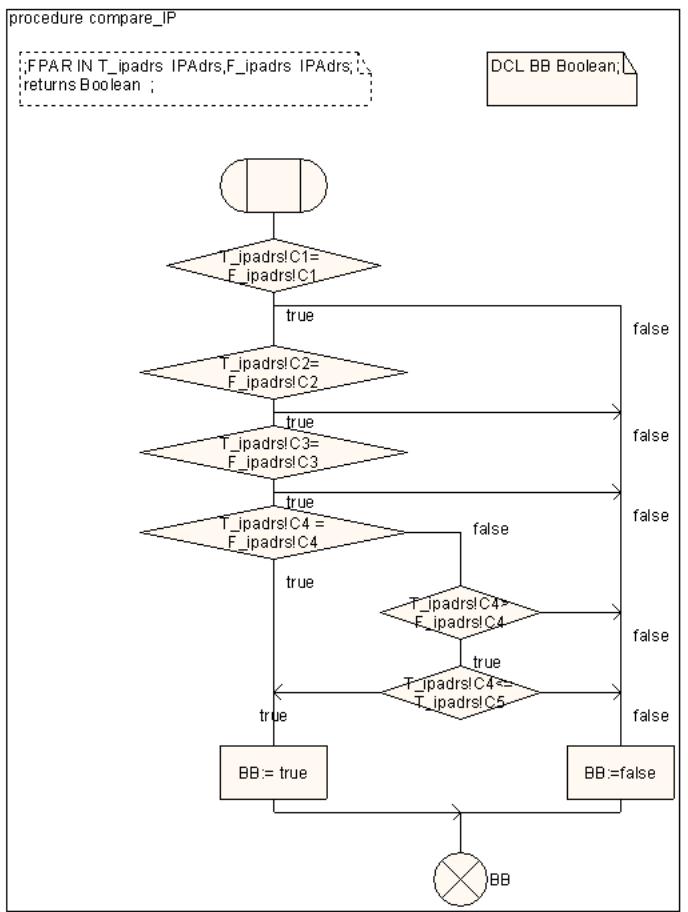

Fig. B.5 – Procédure « compare_IP »

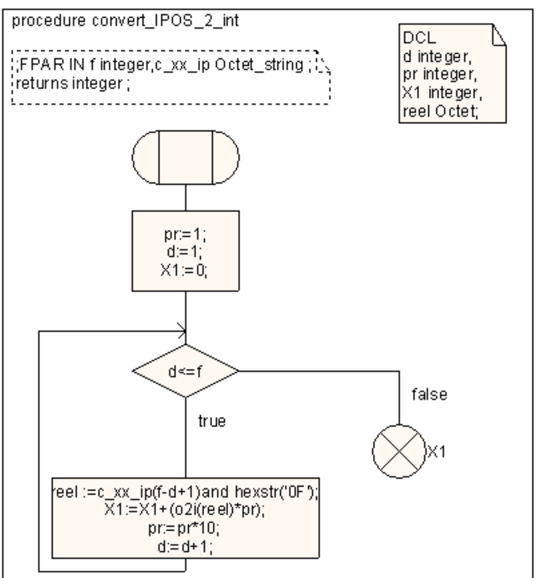

Fig. B.6 – Procédure « convert_IPOS_2_int »

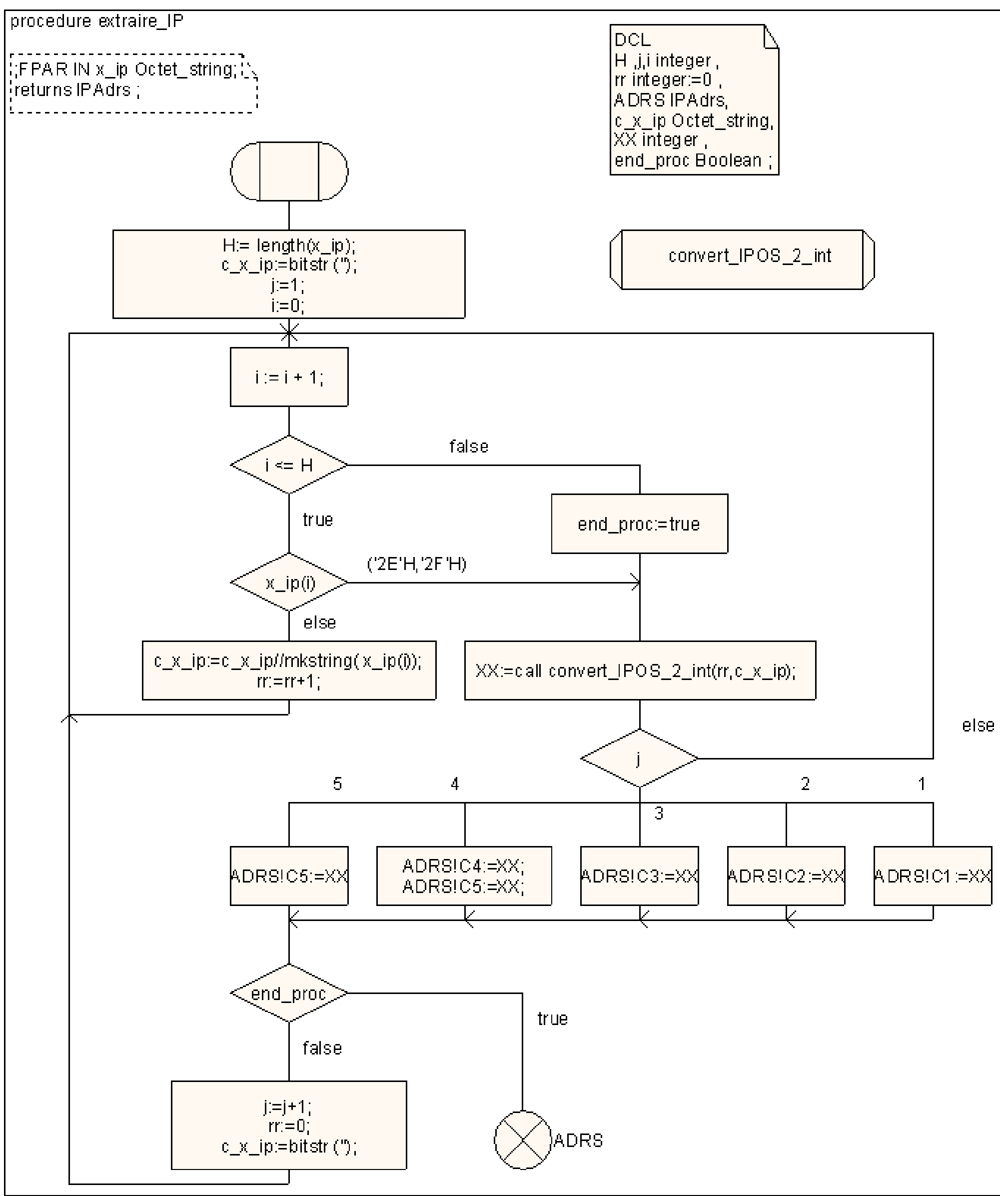

FIG. B.7 – Procédure « extraire_IP »

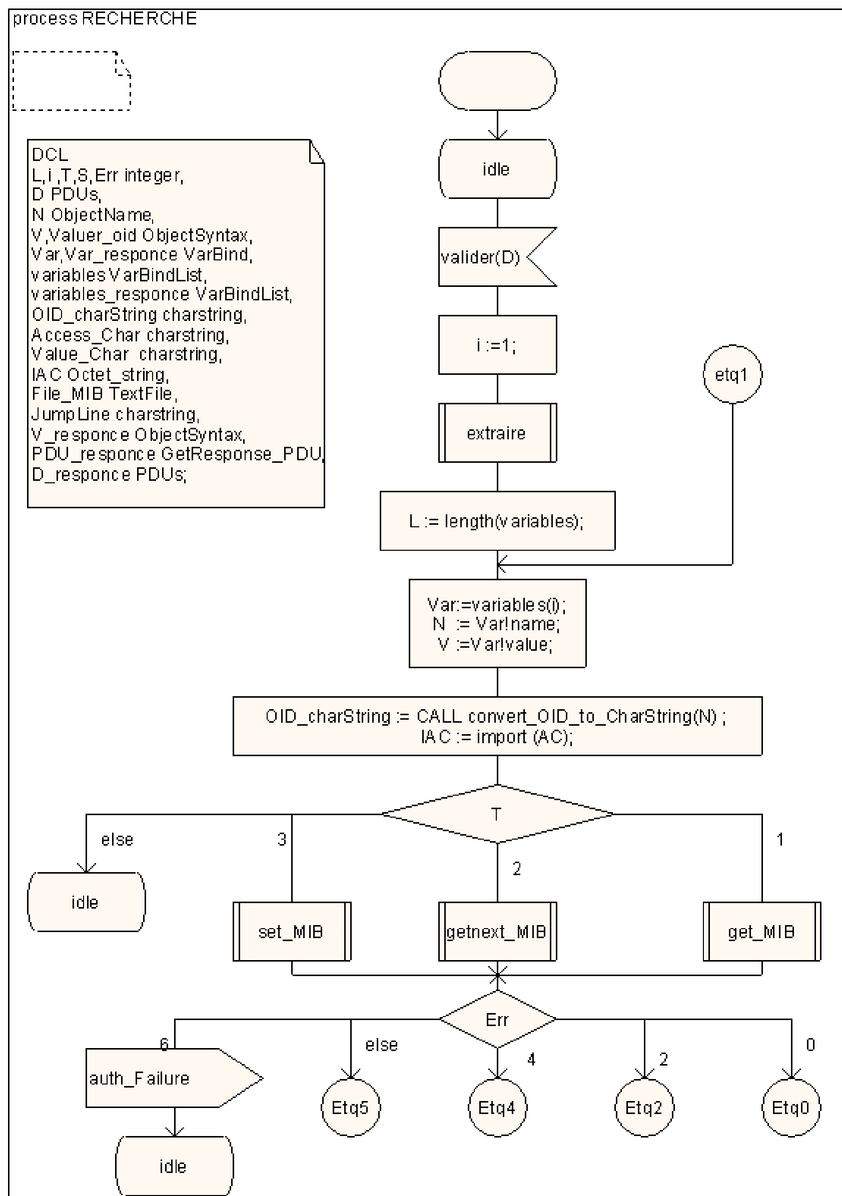

Fig. B.8 – Processus « RECHERCHE » (page1/3)

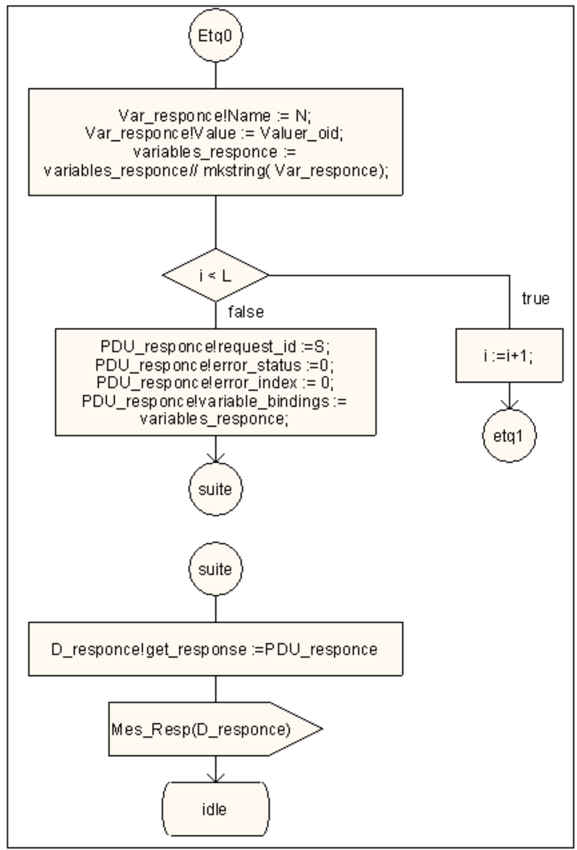

Fig. B.9 – Processus « RECHERCHE » (page2/3)

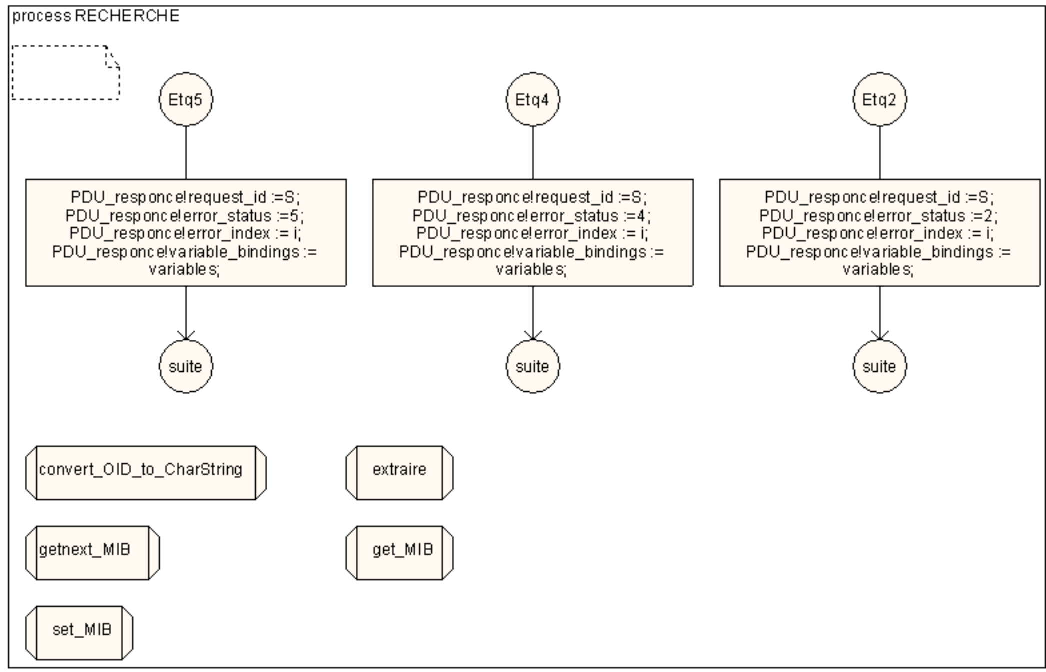

Fig. B.10 – Processus « RECHERCHE » (page3/3)

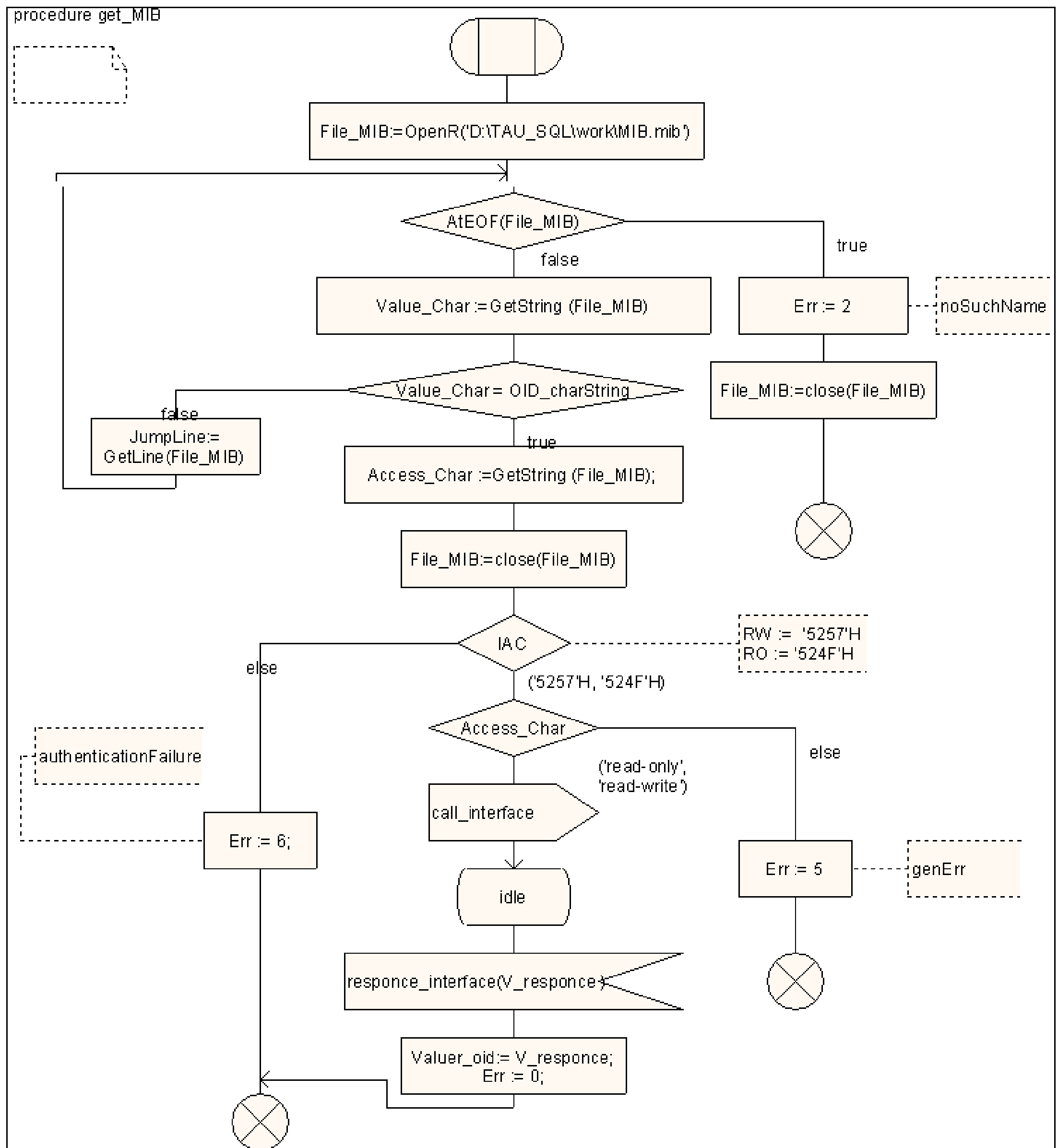

Fig. B.11 – Procédure « get_MIB »

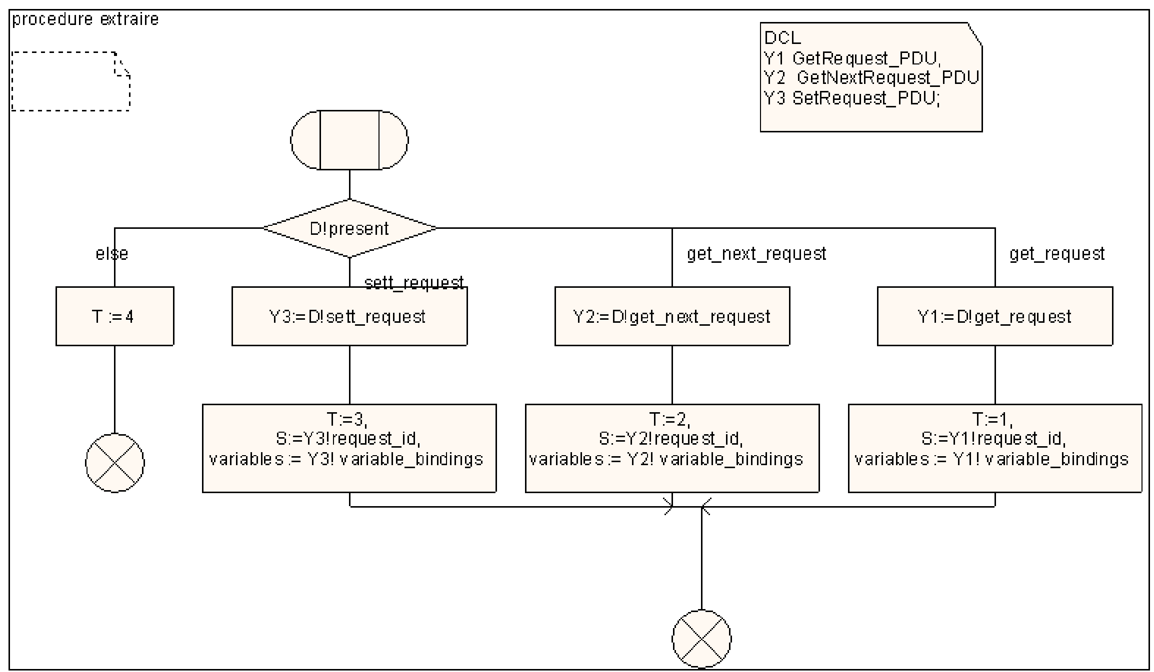

Fig. B.12 – Procédure « extraire »

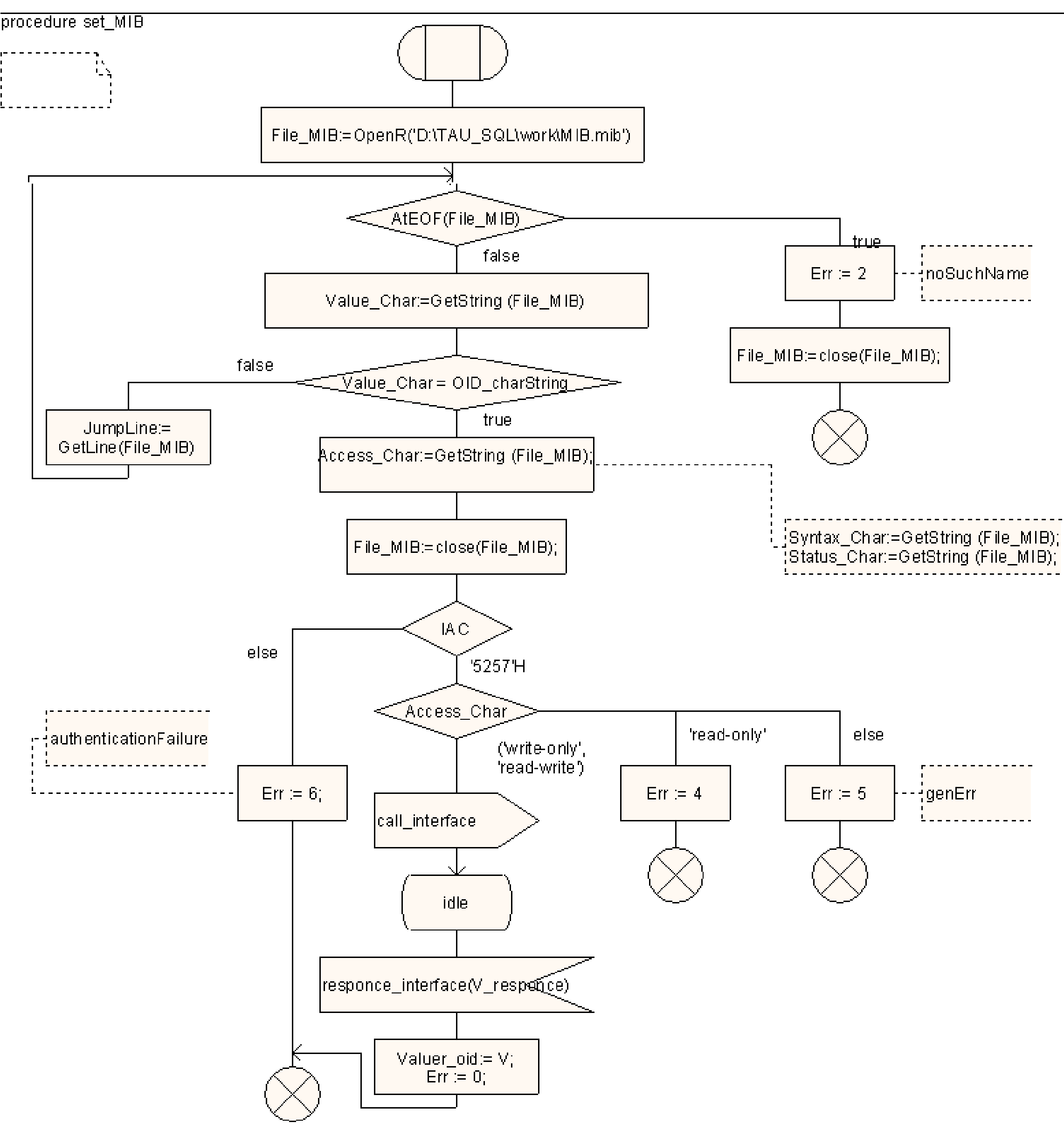

FIG. B.13 – Procédure « set_MIB »

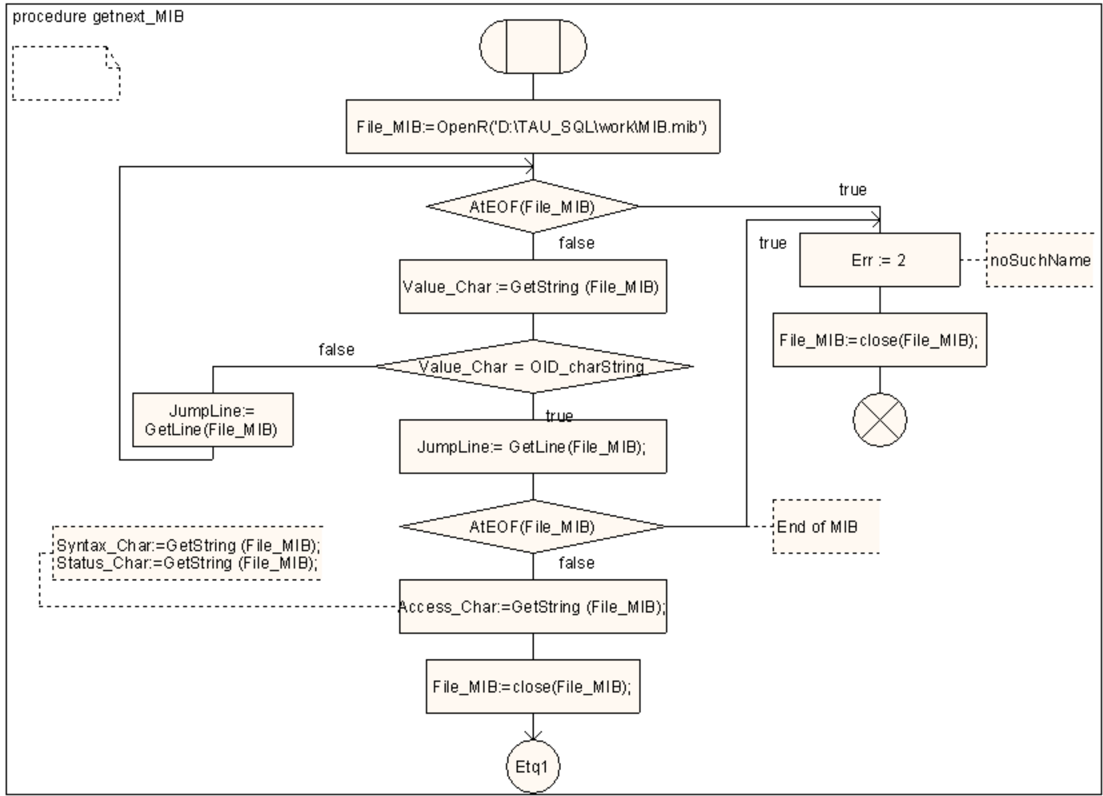

Fig. B.14 – Procédure « getnext_MIB » (page1/2)

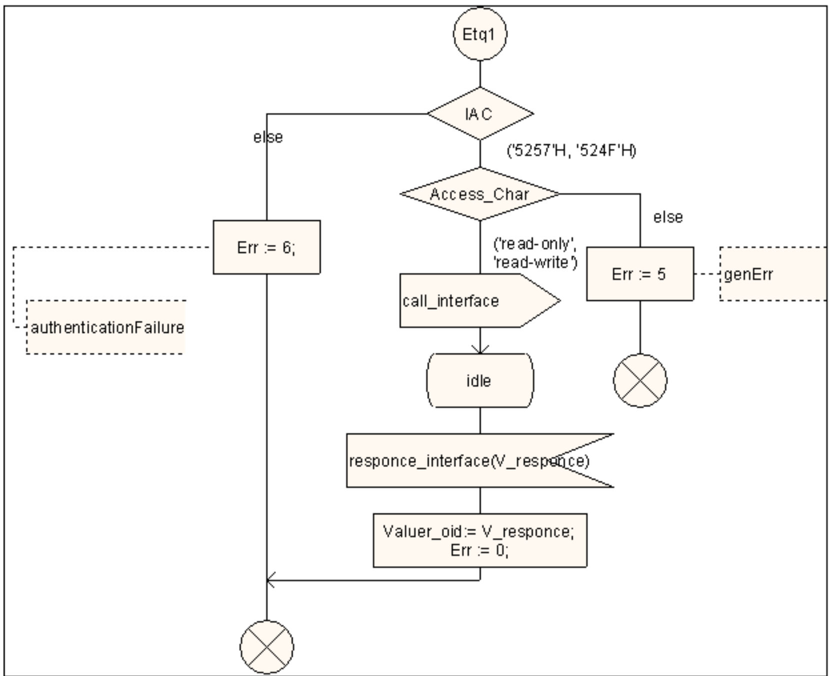

Fig. B.15 – Procédure « getnext_MIB » (page2/2)

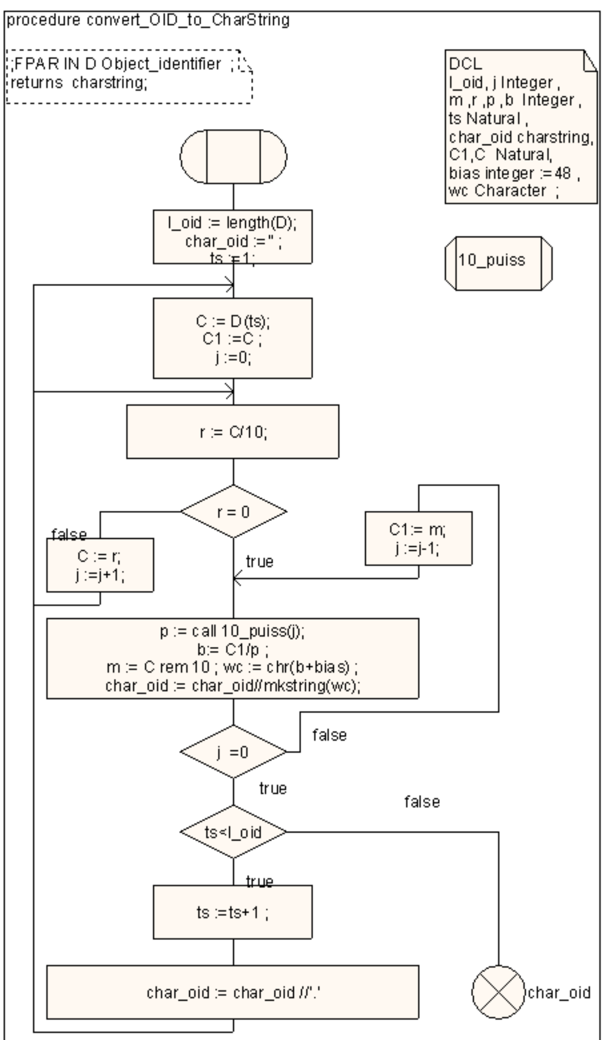

Fig. B.16 – Procédure « convert_OID_to_CharString »

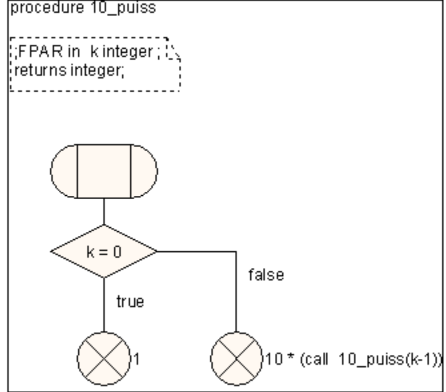

Fig. B.17 – Procédure « 10_puiss »

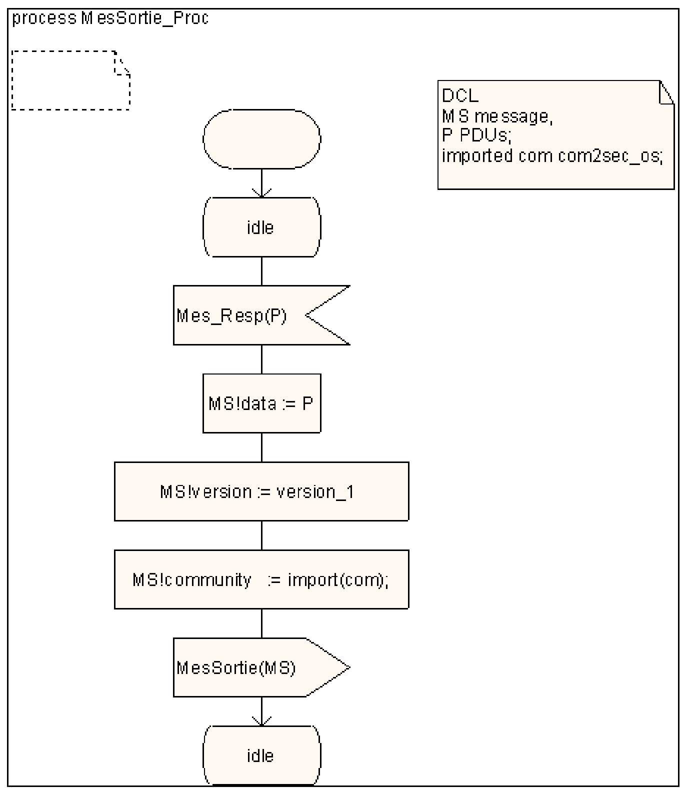

Fig. B.18 – Processus « MesSortie_Proc »

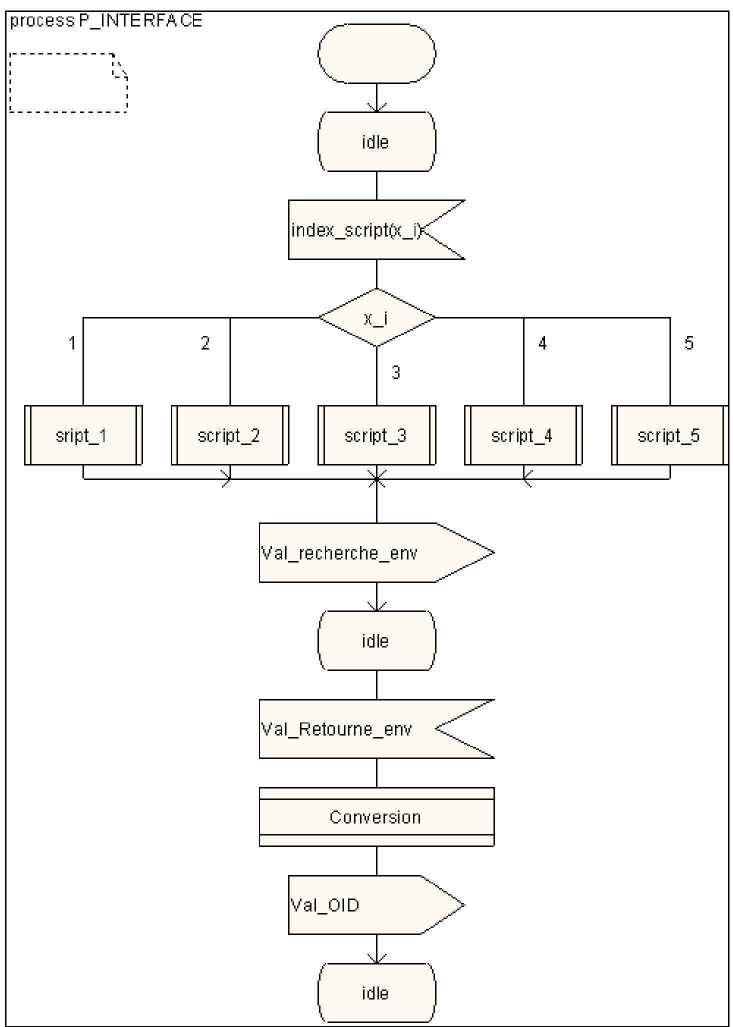

FIG. B.19 – Processus « P_INTERFACE »

Module ASN.1 de la RFC1157

```
qRFC1157-SNMP DEFINITIONS : := BEGIN

IMPORTS

ObjectName, ObjectSyntax, NetworkAddress, IpAddress, TimeTicks

FROM RFC1155-SMI ;

-- top-level message

Message : := SEQUENCE { version INTEGER { version-1(0) }, -- version-1 for this RFC

                       community OCTET STRING,            -- community name

                       data PDUs                          -- e.g., PDUs if trivial

                                                          --ANY authentication is being used

                     }

-- protocol data units

PDUs : := CHOICE { get_request GetRequest_PDU,
```

```
                  get_next_request GetNextRequest_PDU,
                  get_response GetResponse_PDU,
                  sett_request SetRequest_PDU
                  trap Trap_PDU
                  }
GetRequest_PDU : := [0] IMPLICIT SEQUENCE { request_id RequestID,
                                           error_status ErrorStatus, -- always 0
                                           error_index ErrorIndex, -- always 0
                                           variable_bindings VarBindList
                                           }
GetNextRequest_PDU : := [1] IMPLICIT SEQUENCE { request_id RequestID,
                                               error_status ErrorStatus, -- always 0
                                               error_index ErrorIndex, -- always 0
                                               variable_bindings VarBindList
                                               }
GetResponse_PDU : := [2] IMPLICIT SEQUENCE { request_id RequestID,
                                            error_status ErrorStatus, -- always 0
                                            error_index ErrorIndex, -- always 0
                                            variable_bindings VarBindList
                                            }
SetRequest_PDU : := [3] IMPLICIT SEQUENCE { request_id RequestID,
                                            error_status ErrorStatus, -- always 0
                                            error_index ErrorIndex, -- always 0
                                            variable_bindings VarBindList
}
Trap_PDU : := [4] IMPLICIT SEQUENCE { enterprise OBJECT IDENTIFIER,
                                     agent_addr NetworkAddress,
                                     generic_trap INTEGER { coldStart(0),
                                     warmStart(1),
                                     linkDown(2),
                                     linkUp(3),
                                     authenticationFailure(4),
                                     egpNeighborLoss(5),
                                     enterpriseSpecific(6)
                                     },
                                     specific_trap INTEGER,
                                     time_stamp TimeTicks,
                                     variable_bindings VarBindList
                                     }
request/response information
RequestID : := INTEGER
ErrorStatus : := INTEGER { noError(0),
```

```
                        tooBig(1),
                        noSuchName(2),
                        badValue(3),
                        readOnly(4),
                        genErr(5)
                        }
ErrorIndex : := INTEGER
-- variable bindings
VarBind : := SEQUENCE { name ObjectName,
                     value ObjectSyntax
}
VarBindList : := SEQUENCE OF VarBind
END
```

Module ASN.1 de la RFC1155

```
RFC1155-SMI DEFINITIONS : := BEGIN
EXPORTS -- EVERYTHING
             ObjectName, ObjectSyntax, SimpleSyntax,
             ApplicationSyntax, NetworkAddress, IpAddress,
             Counter, Gauge, TimeTicks, Opaque ;
-- the path to the root
-- definition of object types
-- names of objects in the MIB
ObjectName : := OBJECT IDENTIFIER
-- syntax of objects in the MIB
ObjectSyntax : := CHOICE {
                            Simple SimpleSyntax,
                                    -- mentioned here to keep things simple (i.e.,
                                    -- prevent mis-use). However, application-wide
                                    -- types which are IMPLICITly encoded simple
                                    -- SEQUENCEs may appear in the following CHOICE
                            application-wide ApplicationSyntax
                        }
SimpleSyntax : := CHOICE {
                            Number INTEGER,
                            String OCTET STRING,
                            Object OBJECT IDENTIFIER,
                            Empty NULL
                        }
ApplicationSyntax : := CHOICE {
                            Address NetworkAddress,
```

```
                              Counter Counter,
                              Gauge Gauge,
                              Ticks TimeTicks,
                              Arbitrary Opaque
                              -- other application-wide types, as they are
                              -- defined, will be added here
                             }
-- application-wide types
NetworkAddress : := CHOICE {
                              Internet IpAddress
                           }
IpAddress : := [APPLICATION 0]            -- in network-byte order
                                         IMPLICIT OCTET STRING (SIZE (4))
Counter : := [APPLICATION 1]              IMPLICIT INTEGER (0..4294967295)
Gauge : := [APPLICATION 2]                IMPLICIT INTEGER (0..4294967295)
TimeTicks : := [APPLICATION 3]            IMPLICIT INTEGER (0..4294967295)
Opaque : := [APPLICATION 4]               -- arbitrary ASN.1 value,
                                         IMPLICIT OCTET STRING -- "double-wrapped"

END
```