THÈSE

présentée par

Tarek BEN ISMAIL

pour obtenir le titre de **DOCTEUR**

de l'INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE
(arrêté ministériel du 30 Mars 1992)

Spécialité : **Informatique**

SYNTHÈSE AU NIVEAU SYSTÈME ET CONCEPTION DE SYSTÈMES MIXTES LOGICIELS/MATÉRIELS

Date de soutenance : 9 Janvier 1996

Composition du Jury :

Messieurs	Guy MAZARÉ	Président
	Ivo BOLSENS	Rapporteur
	Patrice QUINTON	Rapporteur
	Jean-Louis LARDY	Examinateur
	Ahmed Amine JERRAYA	Examinateur

Thèse préparée au sein du Laboratoire TIMA-INPG
46, Avenue Félix Viallet, 38031 Grenoble

À ma mère,
À mon père,
À Hatem, et
À Samia,

Résumé

L'objet de ces travaux de thèse est d'étudier la spécification et la synthèse de systèmes de contrôle, qui peuvent être composés à la fois de logiciel et de matériel, sur des architectures multiprocesseurs (ASIC, FPGA, et logiciel). Ce sujet de recherche fait partie à la fois de la synthèse de systèmes VLSI et de la conception mixte logicielle/matérielle. Afin d'atteindre ces objectifs, une méthodologie qui permet de concevoir conjointement le logiciel et le matériel a été développée. L'originalité de ce travail vient du fait que les spécifications à traiter sont décrites à un très haut niveau d'abstraction, appelé "niveau système", avec le langage SDL. Ceci permet de concevoir des applications de plus en plus complexes. Ces travaux traitent principalement le problème du découpage de systèmes de contrôle en sous-systèmes de granularité plus fine et donc plus facilement synthétisables. L'approche de découpage qui a été développée se base sur une boîte à outils qui offre au concepteur le moyen de transformer, raffiner, découper un système puis d'affecter chaque sous-système à une technologie particulière en logiciel (C) ou en matériel (VHDL). La méthode de découpage suivie est interactive et utilise une forme intermédiaire basée sur un modèle de machines à états finis étendues communicantes via des canaux abstraits. Une autre tâche tout aussi importante dans cette méthodologie de raffinement est de synthétiser la communication entre les différentes partitions résultat d'un découpage. Cela se traduit par une étape d'allocation de protocoles de communication et une étape de synthèse d'interfaces entre les sous-systèmes communicants. La première étape consiste à sélectionner dans une bibliothèque les modèles de communication nécessaires entre les sous-systèmes. La deuxième étape consiste à adapter ou générer les interfaces des différents sous-systèmes.

Mots clés: synthèse au niveau système, conception conjointe logiciel/matériel, machine d'états finis étendue, SDL, C/VHDL, découpage de descriptions, synthèse de la communication, sélection de protocoles de communication, génération d'interfaces.

Abstract

The objective of this thesis is to develop a system-level specification and synthesis approach that allows an interactive hardware/software codesign of applications onto multiprocessor architectures composed of ASICs, FPGAs, or software processors. This thesis presents a hardware/software codesign methodology that starts with a specification given in the system-level description language, called SDL, and generates, through an intermediate representation called Solar, hardware and software descriptions in VHDL and C languages respectively. Two main steps are required in order to transform this specification into mixed hardware/software descriptions used for synthesising the hardware and compiling the software parts. Firstly, a system-level partitioning step is needed in order to transform, and split the model into a set of communicating subsystems. Secondly, a communication synthesis step, including protocol selection and interface generation tasks, is needed in order to refine the model into a set of interconnected subsystems. Each of these subsystems is described either in C code or in VHDL. Software parts may be compiled for a standard microprocessor and hardware parts may feed existing high-level synthesis tools in order to programme FPGAs or design ASICs.

Keywords: System-level synthesis, hardware/software codesign, extended finite state machine, SDL, C/VHDL, partitioning, communication synthesis, communication protocol selection, interface generation.

Table des Matières

Liste des Figures vii
Liste des Tableaux x

Chapitre 1: Introduction 1
1.1. Motivations 2
1.2. La conception conjointe de logiciel/matériel 3
1.3. Objectifs 5
1.4. Contribution 6
1.5. Plan de la thèse 7

Chapitre 2: Modèles, langages de spécification au niveau système, et systèmes de conception logiciel/matériel 9
2.1. Introduction 10
2.2. Systèmes de conception conjointe logiciel/matériel : État de l'art 11
2.2.1. Taxonomie des outils de conception logiciel/matériel 13
2.3. Modélisation: État de l'art 15
2.4. Langages du niveau système : État de l'art 19
2.4.1. Taxonomie des langages de spécification 20
2.4.1.1. Puissance d'expression 20
2.4.1.2. Puissance d'analyse 22
2.4.1.3. Arguments commerciaux 23
2.4.2. Comparaison des langages de spécification 23
2.5. Conclusion 26

Chapitre 3: Modélisation pour la synthèse de systèmes mixtes logiciels/matériels 27
3.1. Introduction 28
3.2. Le format SOLAR : Les concepts de base 28
3.3. La table d'états 33
3.4. L'unité de conception 34
3.5. Le canal de communication 35
3.6 Conclusion 39

Liste des Figures

1.1. La statégie de conception d'un système....4
2.1. Approche typique de synthèse....12
2.2. Étapes du cycle de développement d'un système....12
2.3. Modèle de machine à états finis pour un four à micro-onde....17
3.1. Environnement de Solar....29
3.2. Système de contrôle d'une pompe minière
(a) Représentation schématique
(b) Représentation au niveau système par des MEFs communicantes....30
3.3. (a) Représentation de ABC par un StateCharts
(b) Représentation hiérarchique de ABC....32
3.4. Attributs d'une table d'états....33
3.5. Machine Request : (a) Représentation par un StateCharts,
(b) Représentation hiérarchique et parallèle de la table d'état : Request....34
3.6. Unités de conception....35
3.7. Structure d'un canal Solar qui offre m services....36
3.8. Niveaux d'abstraction d'un canal connectant deux processeurs :
(a) vue conceptuelle, (b) extrait de la spécification en Solar du système,
(c) description en Solar d'un canal,
(d) réalisation d'une unité canal physique....38
3.9. Organisation de la structure d'une description en Solar....39
4.1. Approche de synthèse dans COSMOS....42
4.2. Saisies des spécifications....44
4.3. Conversion du modèle SDL en un modèle Solar....45
4.4. Étape de découpage dans COSMOS....49
4.5. Étape d'affectation des canaux....51
4.6. Étape de synthèse des interfaces....52
4.7. Étape de génération d'un prototype virtuel....53
4.8. Procédé de traduction d'une table d'états :
(a) Flot de traitement, (b) Exemple de traduction d'une table d'états....55
4.9. Procédé de traduction de l'unité de conception structurelle....56
4.10. Style de description d'un programme C généré :
(a) Modèle de la machine d'états décrite en Solar,

Liste des Tableaux

2.1. Tableau comparatif des outils de conception logiciel/matériel 14
2.2. Comparaison des langages de spécification 24
4.1. Correspondance entre Solar et VHDL comportemental 54
5.1. Résultat de fusion de MEFs 69
5.2. Estimation du coût de chaque processus de la figure 5.2 69
5.3. Résultats de découpage du système de la figure 5.2 70
5.4. Performances des partitions générées à la suite du découpage 90

"Nous avançons dans l'obscurité, lentement, nous n'avons guère de force. Mais nous avançons et notre marche au milieu des ténèbres infinies me paraît magnifique. Le doute n'est point pour nous «un oreiller commode» où l'on pose la tête pour rêver. L'état de doute représente une étape dans la recherche scientifique où vont naître les hypothèses que l'on devra vérifier."

Robert DEBRÉ dans le livre "Ce que je crois", 1976.

Chapitre 1

Introduction

Dans ce chapitre d'introduction, les motivations et les objectifs de cette thèse seront définis. Les différents problèmes rencontrés dans la synthèse au niveau système et lors de la conception conjointe de logiciel/matériel seront passés en revue. Une approche de ces problèmes et les idées conduisant à leurs résolutions seront présentées. La contribution apportée au cours de cette thèse sera brièvement exposée. Finalement, un plan de la thèse sera fourni.

Chapitre 2

Modèles, langages de spécification au niveau système, et systèmes de conception logiciel/matériel

---

Le but de ce chapitre est de présenter l'état de l'art des systèmes de conception logiciel/matériel ainsi que des modèles et langages de spécification au niveau système. Les systèmes existants de conception mixte logicielle/matérielle seront passés en revue. Les caractéristiques de chaque système seront détaillées afin de montrer les domaines d'application respectifs. Ensuite, une comparaison des langages de spécification au niveau système sera proposée à travers les concepts inhérents à chaque langage.

Chapitre 3

Modélisation pour la synthèse de systèmes mixtes logiciels/matériels

---

Dans ce chapitre le format intermédiaire, Solar, sera présenté. Ce format sert de modèle de représentation de spécifications provenant de langages de description de logiciel ou de matériel. Ainsi les parties logicielles et matérielles d'un même système peuvent être unifiées dans le même format Solar. Le modèle de description adopté dans Solar est une extension au modèle des machines à états finis. Les extensions concernent le parallélisme, la hiérarchie, la communication entre machines à états finis, et le traitement des exceptions.

Chapitre 4

Méthodologie de conception dans COSMOS

Le but de ce chapitre est de donner une vue globale de COSMOS, une nouvelle méthode de conception conjointe logicielle/matérielle. Cette approche se compose d'un ensemble d'étapes qui servent à transformer une spécification de haut niveau en une réalisation comportant à la fois du logiciel et du matériel. Dans ce chapitre, chaque étape sera décrite à travers les modèles qu'elle utilise et les transformations qu'elle est amenée à réaliser.

Chapitre 5

Le découpage de systèmes au niveau système

Le but de ce chapitre est de présenter une nouvelle méthode de découpage de systèmes pour la conception conjointe de systèmes mixtes pouvant contenir du logiciel et du matériel. Il s'agit de transformer une description système en un ensemble de sous-systèmes appelés partitions. Chaque partition comporte une ou plusieurs fonctions du système. Ce découpage sert à réduire la complexité des éléments traités lors de la conception mais aussi à choisir une réalisation (matérielle ou logicielle) pour chaque sous-système. La méthode de découpage proposée se situe au niveau système et se base sur le principe diviser pour régner. Cette méthode est interactive et permet l'application d'un ensemble de primitives de découpage et de transformation.

Chapitre 6

La synthèse de communication

Le but de ce chapitre est de présenter une approche de synthèse de la communication au niveau-système. Cette étape est vue comme un problème d'allocation d'unités de communication e.g. canaux avec leurs protocoles. Cette synthèse de communication comporte à la fois la synthèse de protocoles et la synthèse d'interfaces. La première consiste à choisir des protocoles (standards ou spécifiques) pour établir des communications entre les différents composants d'un système. La deuxième sert à adapter l'interface des sous-systèmes afin de communiquer à travers les protocoles préalablement sélectionnés.

Chapitre 7

Conclusions et perspectives

Bibliographie

Bibliographie

[Abri88] J-R. Abrial, “Une Approche Formelle du Développement des Logiciels,” Génie logiciel & Systèmes experts, N° 11, Mars 1988.

[AdSc93] J.K. Adams, H. Schmitt, and D.E. Thomas, “A Model and Methodology for Hardware-Software Codesign,” Handouts of Int'l Wshp on Hardware-Software Co-design, Cambridge, Massachusetts, IEEE CS Press, October 1993.

[AnBa94] S. Antoniazzi, A. Balboni, W. Fornaciari, and D. Sciuto, “A Methodology for Control-Dominated Systems Cedesign,” Proc. Third Int'l Wshp on Hardware/Software Codesign (CODES/CASHE), Grenoble, France, IEEE CS Press, pp. 2-9, September 1994.

[AnCa93] P. Angosto, B. Caillaud, M. Delaure, R. Gueschel, B. Jouga, D. Le Foll, J. Maret, P. Rocher, and G. Vaucher, “L'ingénierie des protocoles,” InterEditions Publishers, Paris, 1993, 222 pages.

[Andr91] G.R. Andrews, “Concurrent Programming, Principles and Practice,” Benjamin/Cummings (eds), Redwood City, Calif., pp. 484-494, 1991.

[AnMa90] S. Antoniazzi, and M. Mastretti, “An Interactive Environment for Hardware-Software System Design at the Specification Level,” Microprocessing & Microprogramming, North-Holland, Vol. 30, pp. 545-554, 1990.

[Arno90] A. Arnold, “Systèmes de transitions finis et sémantique des processus communicants,” Technique et Science Informatiques (TSI), Vol. 9, N° 3, Bordas (eds), pp. 193-216, 1990.

[Atam94] Y. Atamna, “Réseaux de Petri temporisés stochastiques classiques et bien formés : définition, analyse et application aux systèmes distribués temps réel,” Rapport LAAS N° 94418, Thèse de Doctorat, Université Paul Sabatier, Toulouse, Octobre 1994.

[AuBe94] M. Auguin, M. Belhadj, J. Benzakki, C. Carrière, G. Durrieu, Th. Gautier, M. Israël, P. Le Guernic, M. Lemaître, E. Martin, P. Quinton, L. Rideau, F. Rousseau, and O. Sentieys, “Towards a Multi-Formalism Framework for Architectural Synthesis: the ASAR Project,” Proc. Third Int'l Wshp on Hardware/Software Codesign (CODES/CASHE), Grenoble, France, IEEE CS Press, pp. 25-32, September 1994.

[BaEc93] M. Bauer, and W. Ecker, “Communication Mechanisms for VHDL Specification and Design Starting at System Level,” Proc. VHDL Forum for CAD in Europe, Spring '93 Meeting, Innsbruck, Austria, pp. 95-106, March 1993.

[BaRo92] E. Barros, and W. Rosentiel, “A Method for hardware Software Partitioning,” IEEE Comp. Euro, 1992.

[BaRo94] E. Barros, W. Rosentiel, and X. Xiong, “A Method for Partitioning UNITY Language in Hardware and Software,” Proc. European Design Automation Conference (EuroDAC), IEEE CS Press, Grenoble, France, September 1994.

Publications personnelles

Publications personnelles

Publications dans des ouvrages et des revues

[BIJe95] T. Ben Ismail, and A.A. Jerraya, “Synthesis Steps and Design Models for CoDesign,” IEEE Computer, Special issue on Rapid-Prototyping of Microelectronic Systems, Vol. 28, N° 2, February 1995, pp. 44-52.

[BIOB95] T. Ben Ismail, K. O'Brien, and A.A. Jerraya, “PARTIF : Interactive System-level Partitioning,” to appear in VLSI Design, Special issue on Decomposition Systems, Gordon & Breach Science Publishers, 1995.

[BIDa95] T. Ben Ismail, J-M. Daveau, K. O'Brien, and A.A. Jerraya, “A System-Level Communication Synthesis Approach for Hardware/Software Systems,” to appear in Int'l Journal Microprocessors and Microsystems, special issue on Hardware/Software Codesign, Butterworth-Heinemann Publishers, 1995.

[JeOB93] A.A. Jerraya, K. O'Brien, and T. Ben Ismail, “Linking System Design Tools and Hardware Design Tools,” Proc. Conference on Hardware Description Languages (CHDL), Ed. D. Agnew, L. Claesen, and R. Composano, Publ. Elsevier, Ottawa, Canada, April 1993, pp. 331.

[DaBI96] J-M. Daveau, T. Ben Ismail, G. Marchioro, and A.A. Jerraya, “Protocol Selection and Interface Generation for HW-SW Codesign,” submitted to IEEE Transactions on VLSI Systems, Special issue on Design Automation of complex integrated systems, September 1996.

[BIMa96] T. Ben Ismail, G. Marchioro, and A.A. Jerraya, “Découpage de Systèmes VLSI à partir d'une Spécification de haut niveau,” accepté à Technique et Science Informatiques (TSI), Hermes (eds), 1996.

Publications dans des conférences internationales

[BIOJ94] T. Ben Ismail, K. O'Brien, and A.A. Jerraya, “Interactive System-level Partitioning with PARTIF,” Proc. European Design & Test Conference (ED&TC), Paris, France, IEEE CS Press, March 1994, pp. 464-468.

[BIA94a] T. Ben Ismail, M. Abid, K. O'Brien, and A.A. Jerraya, “An Approach for Hardware-Software Codesign,” Proc. Int'l Wshp on Rapid System Prototyping (RSP), Grenoble, France, IEEE CS Press, June 1994, pp. 73-80.

[BIA94b] T. Ben Ismail, M. Abid, and A.A. Jerraya, “COSMOS : A CoDesign Approach for Communicating Systems,” Proc. Int'l Wshp on Hardware/Software Codesign (CODES/CASHE), Grenoble, France, IEEE CS Press, September 1994, pp. 17-24.

[BIJe94] T. Ben Ismail, and A.A. Jerraya, “Tutorial : Hardware/Software Codesign,” Invited paper, Eighth Brasilian Symposium on Integrated Circuits Design, Gramado, Brasil, November 1994, pp. 17.

Glossaire

Glossaire

ADA	Langage de programmation (standard ISO)
ADDL	*Algorithm Design Description Language* Langage de spécification basé sur un sous-ensemble de Pascal et une extension afin de décrire le parallèlisme et une sémantique matérielle
ASIC	*Application Specific Integrated Circuit* Circuit intégré dédié à une application
ATM	*Asynchronous Transfer Mode* Protocole de communication asynchrone à haut débit
B	Langage formel basé sur les mathématiques
C^{x}	Langage de sépécification qui est une extension au langage C pour décrire le parallèlisme entre processus et les contraintes de temps
CAO	Conception Assistée par Ordinateur
CCITT	*International Consultative Committee for Telegraphy and Telephony* Organisme de normalisation des protocoles de communication
CMOS	Technologie de fabrication des circuits intégrés
CODES	Outil de conception logicielle/matérielle à Siemens
Co-simulation	Simulation mixte logicielle/matérielle e.g. en C et en VHDL
Co-spécification	Spécification hétérogène d'un système e.g. en C et en VHDL
COSMOS	Projet de conception mixte logiciel/matériel en cours de développement dans l'équipe SLS du laboratoire TIMA
COSYMA	Outil de conception logicielle/matérielle à l'Université de Braunschweig
CPU	*Central Processing Unit* Unité centrale de calcul
CRCW	*Concurrent Read Concurrent Write* Lectures parallèles et écritures parallèles

Annexe B

Étude des principales approches de conception de logiciel/matériel

---

L'objet de cet annexe est de fournir une étude des principales approches de conception de logiciel/matériel. Ces approches sont développées soit dans des universités soit dans des centres de recherches rattachés à des compagnies industrielles.

Annexe C

Algorithmes des primitives

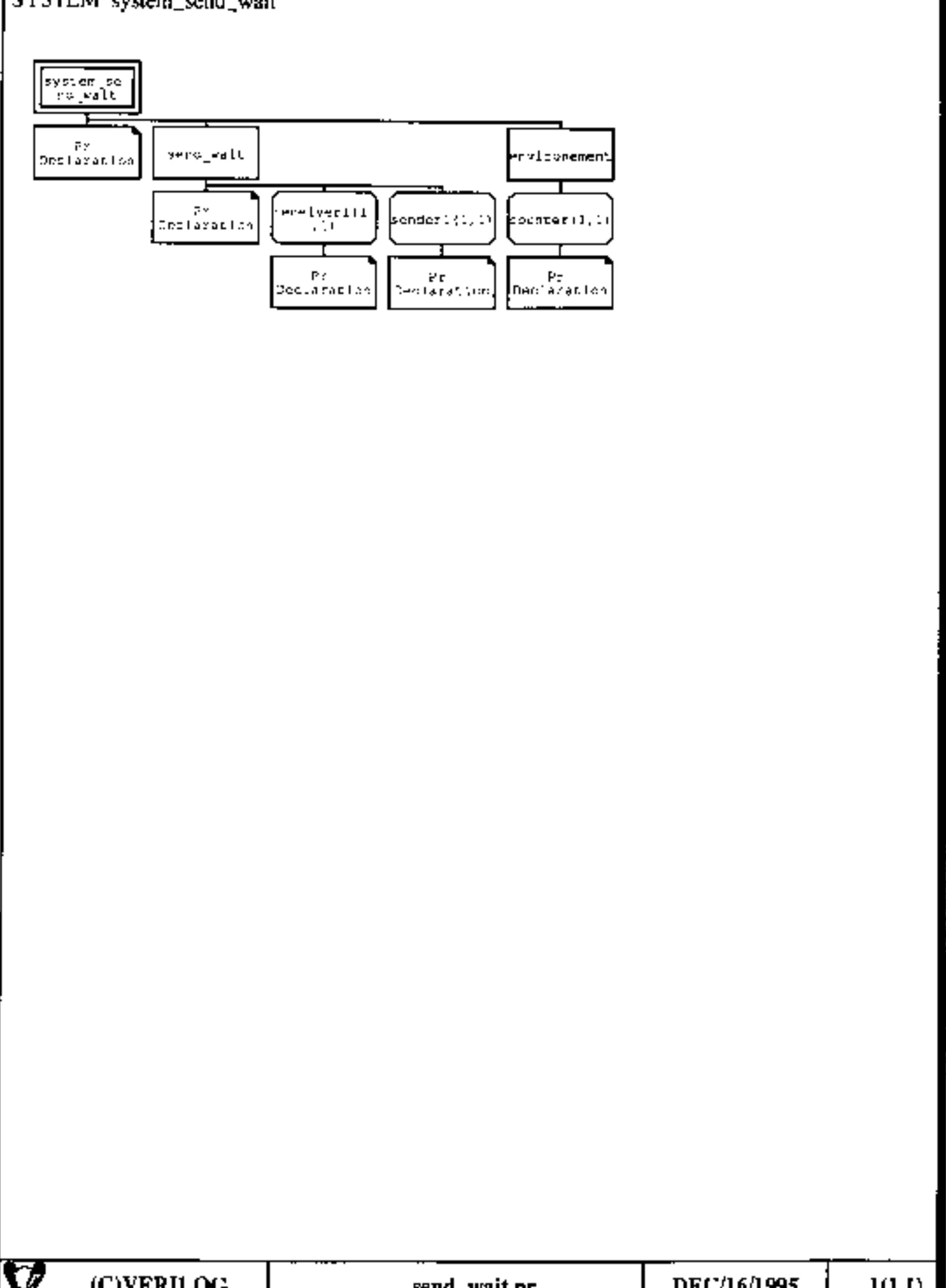
Dans cet annexe les algorithmes des primitives de découpage et de synthèse de communication seront présentés. Ces algorithmes ont été programmés en langage C++ dans un environnement de stations de travail sous le système d'exploitation UNIX.

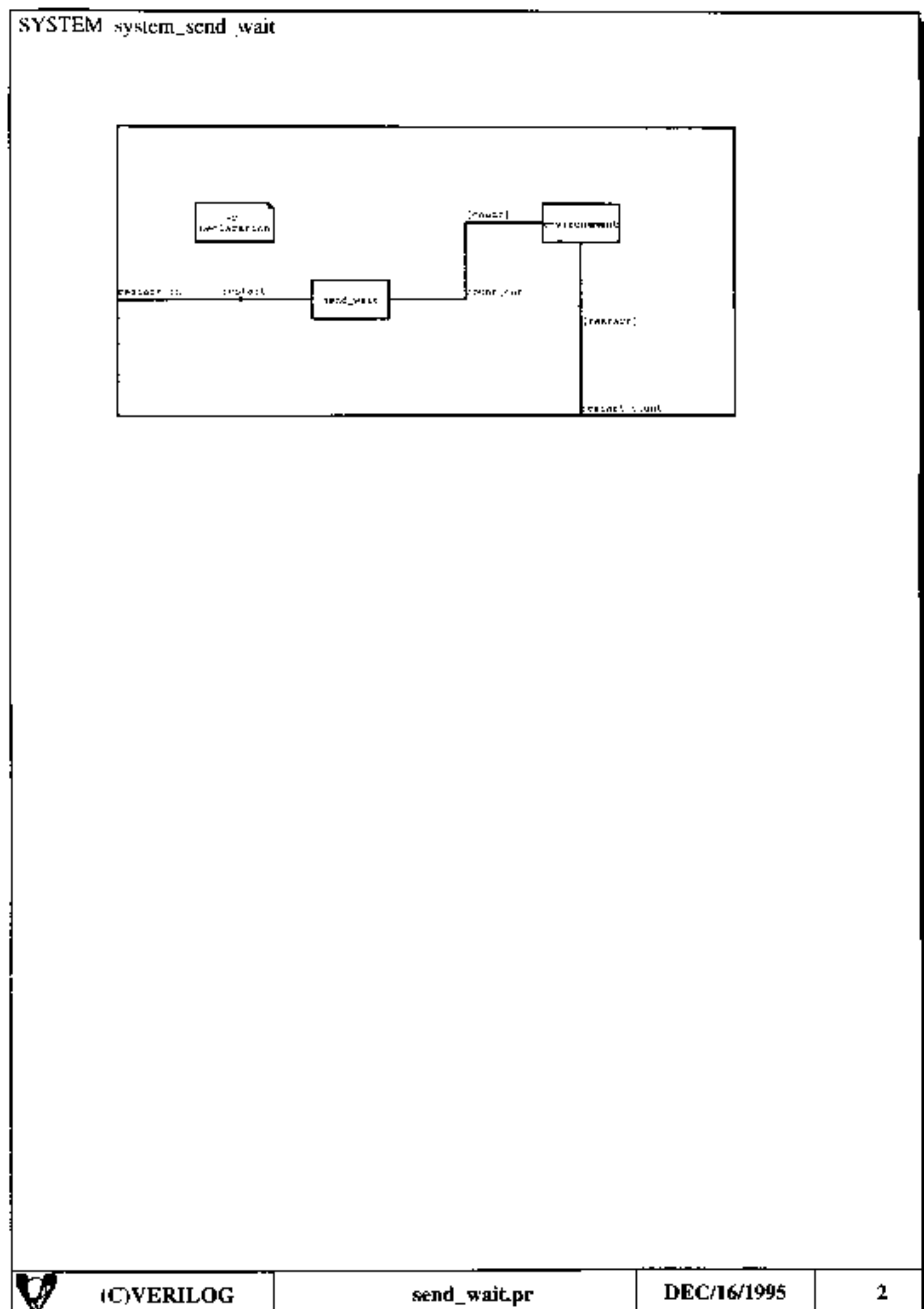
Annexe D

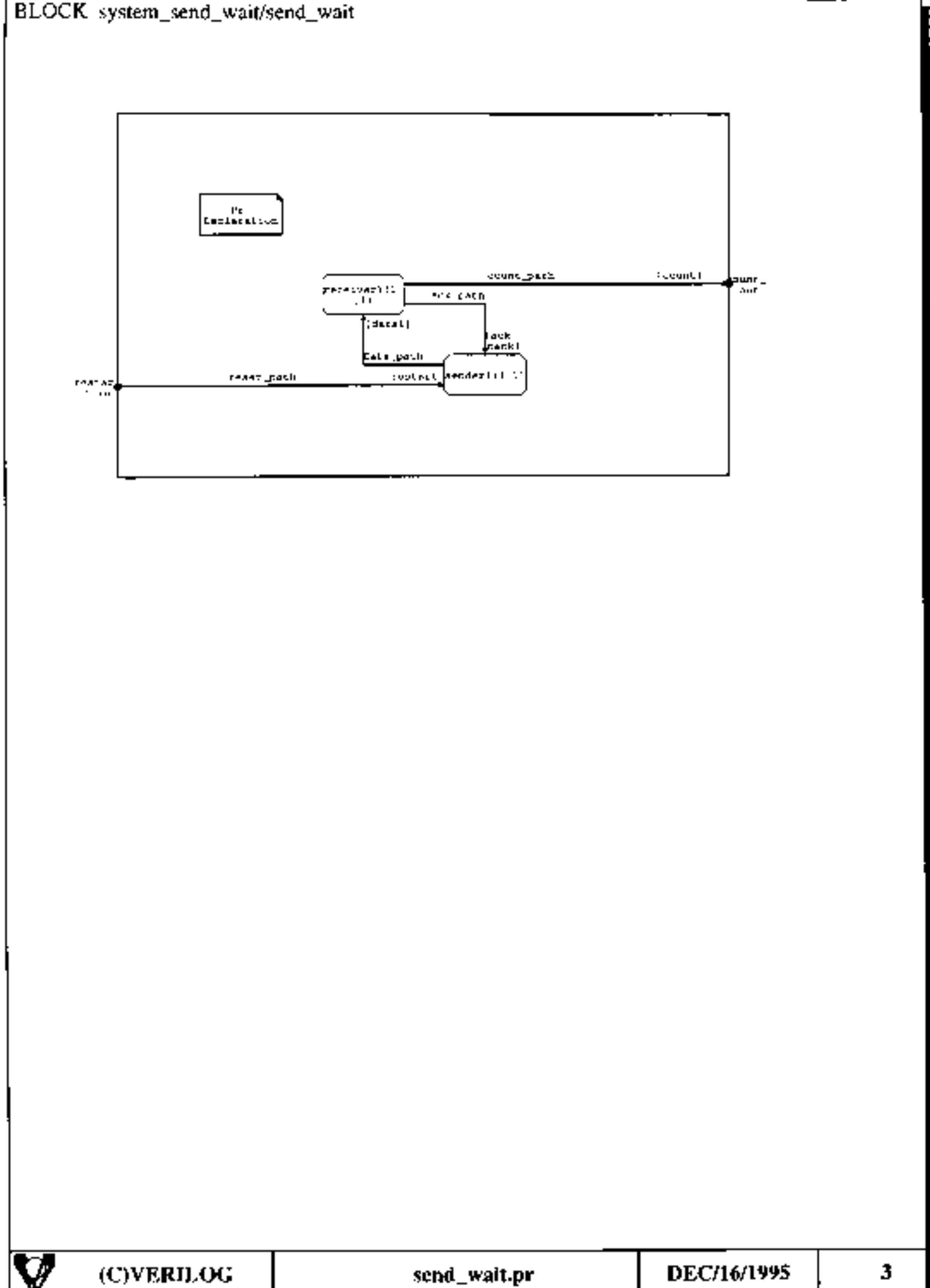
Exemples en SDL graphique

Dans cet annexe deux exemples en SDL graphique seront présentés. Le premier est un système émetteur/récepteur et le deuxième est un bloc simplifié de télécommande d'un satellite. Pour le système émetteur/récepteur, une description en SDL sera donnée, ensuite des copies d'écran des représentations graphiques en Solar seront présentées, et finalement, les descriptions C et VHDL générées automatiquement seront proposées.

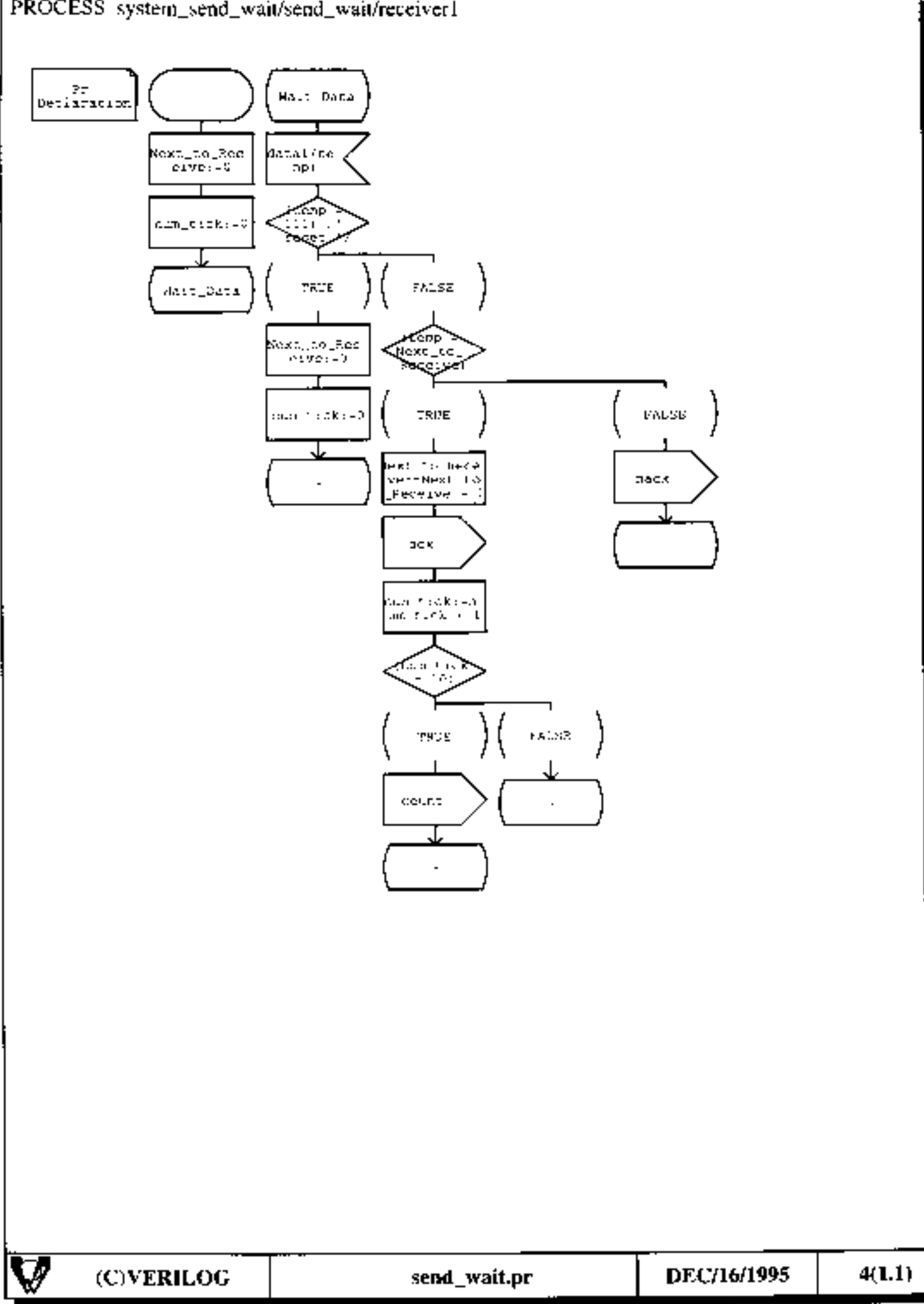
Description en SDL graphique du système Émetteur/Récepteur


SYSTEM system_send_wait
(C)VERILOG
send_wait.pr
DEC/16/1995
1(1.1)

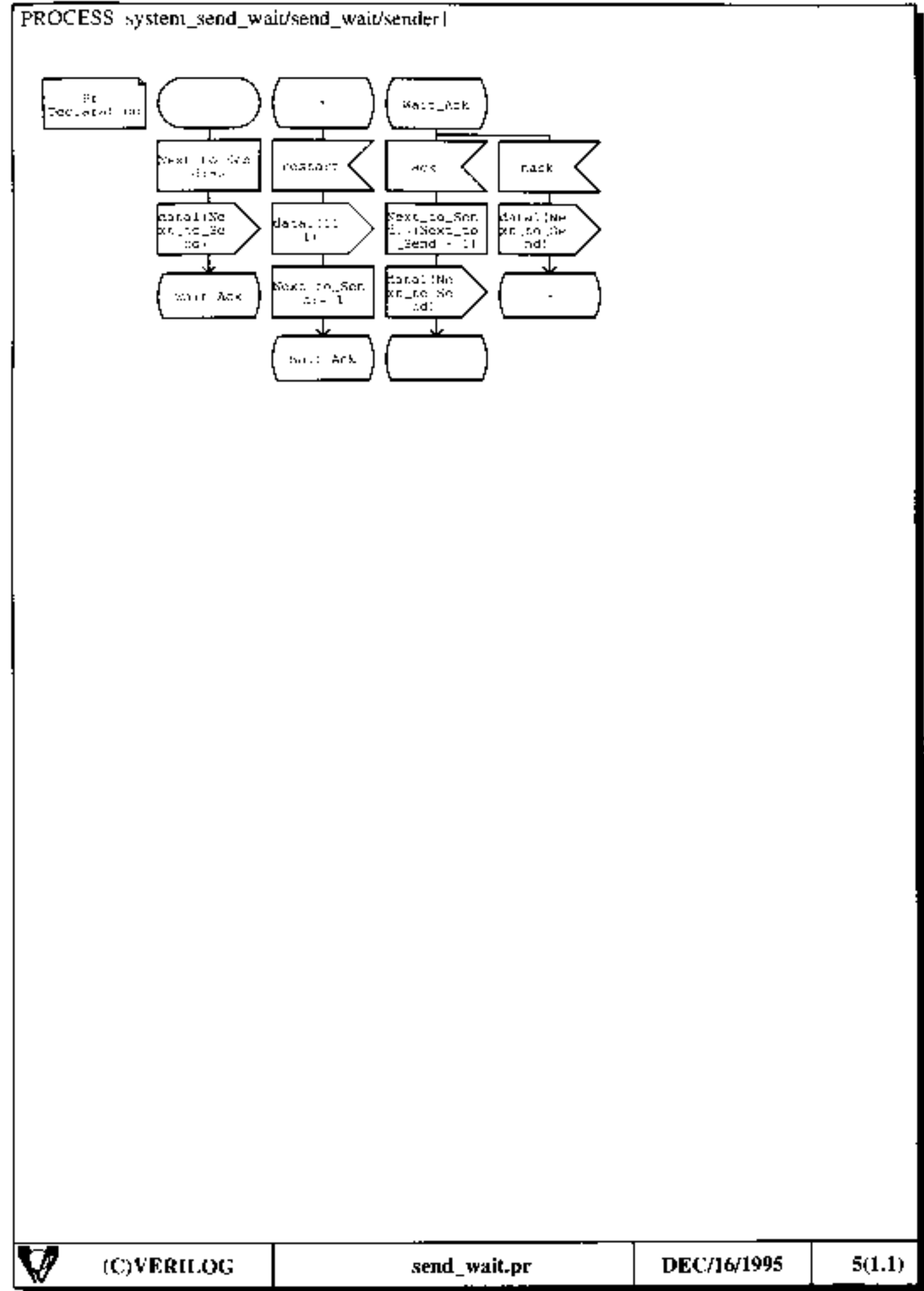

SYSTEM system_send_wait
(C)VERILOG
send_wait.pr
DEC/16/1995
2

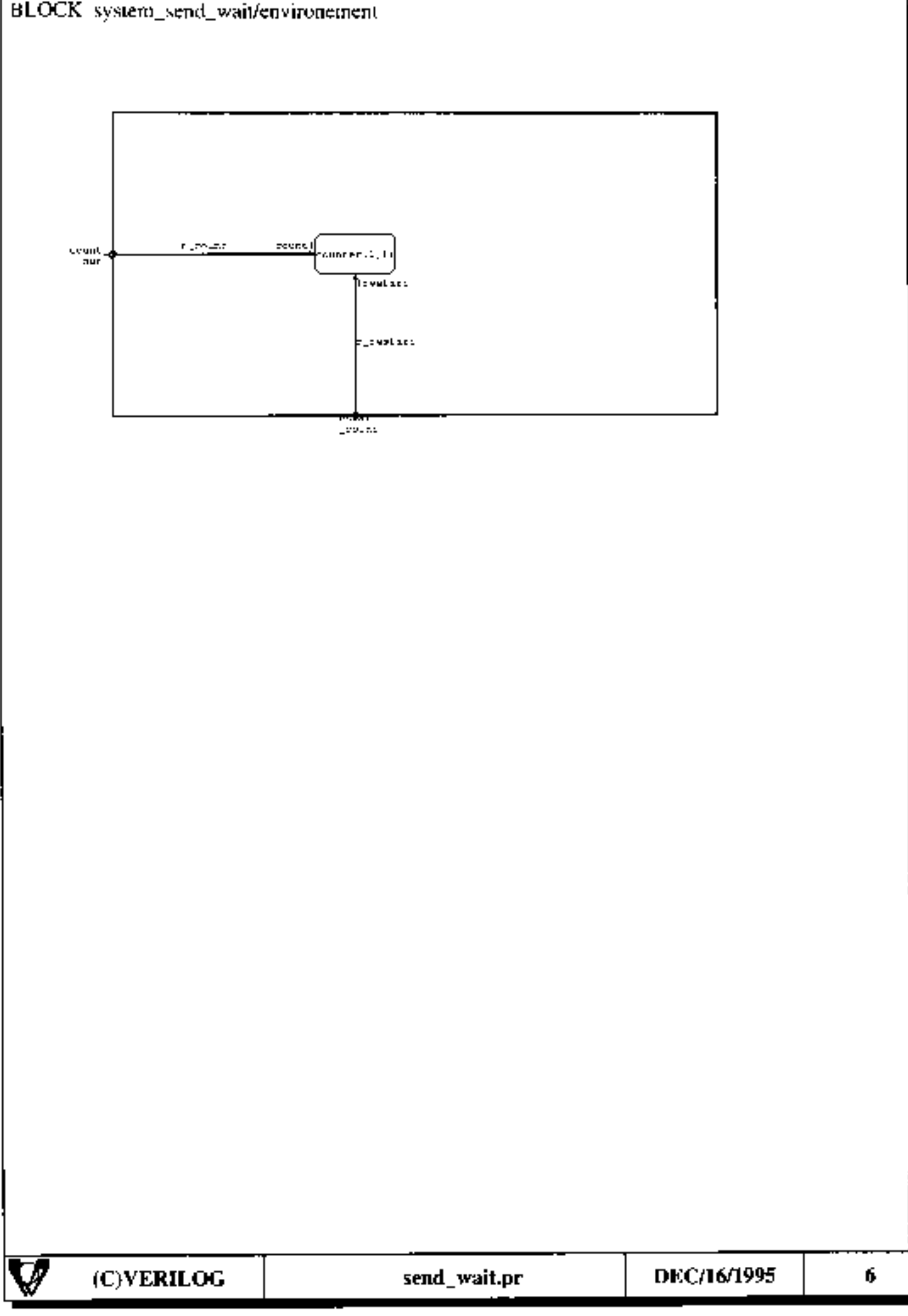

BLOCK system_send_wait/send_wait
(C)VERILOG
send_wait.pr
DEC/16/1995
3

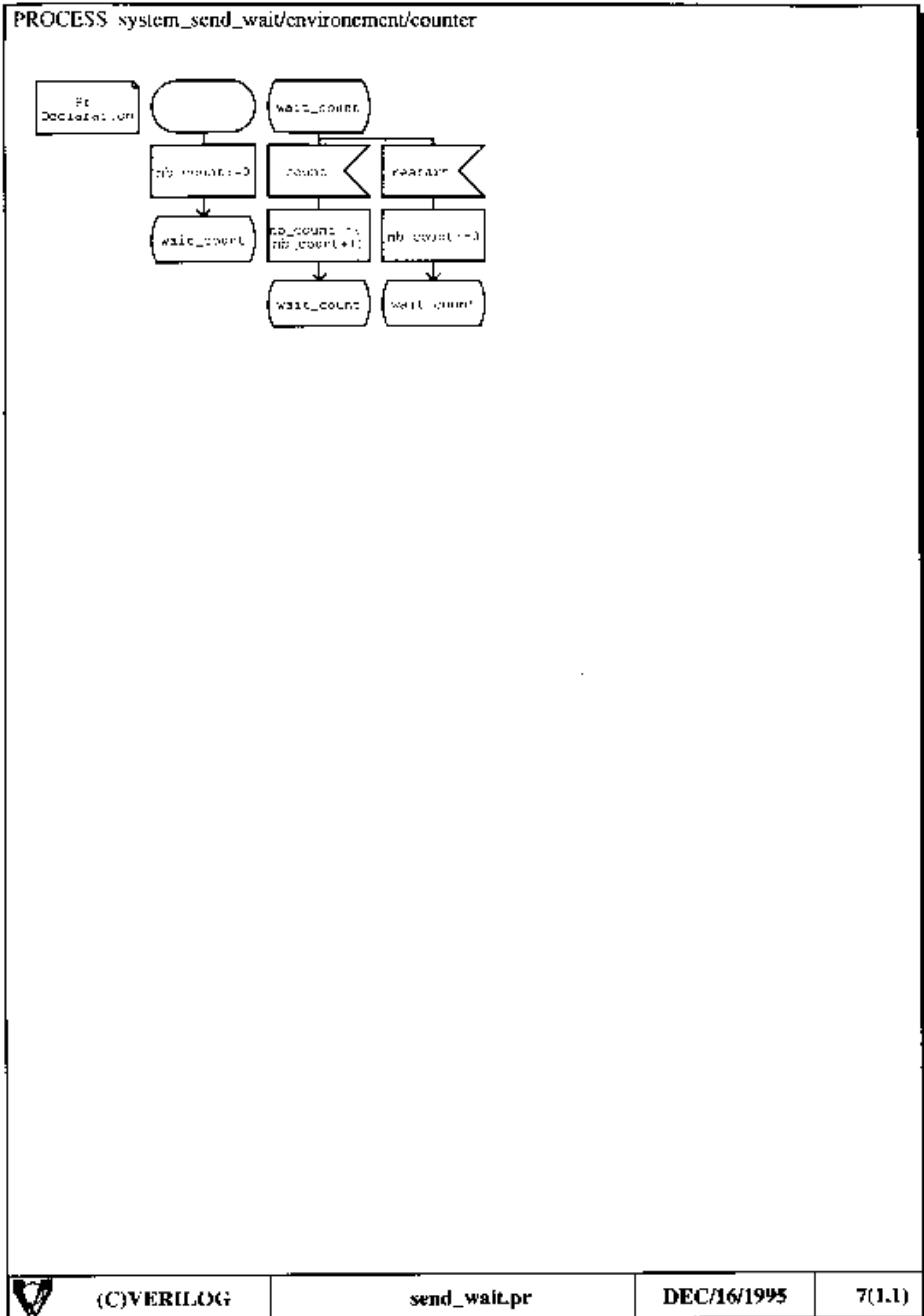
PROCESS system_send_wait/send_wait/receiver1



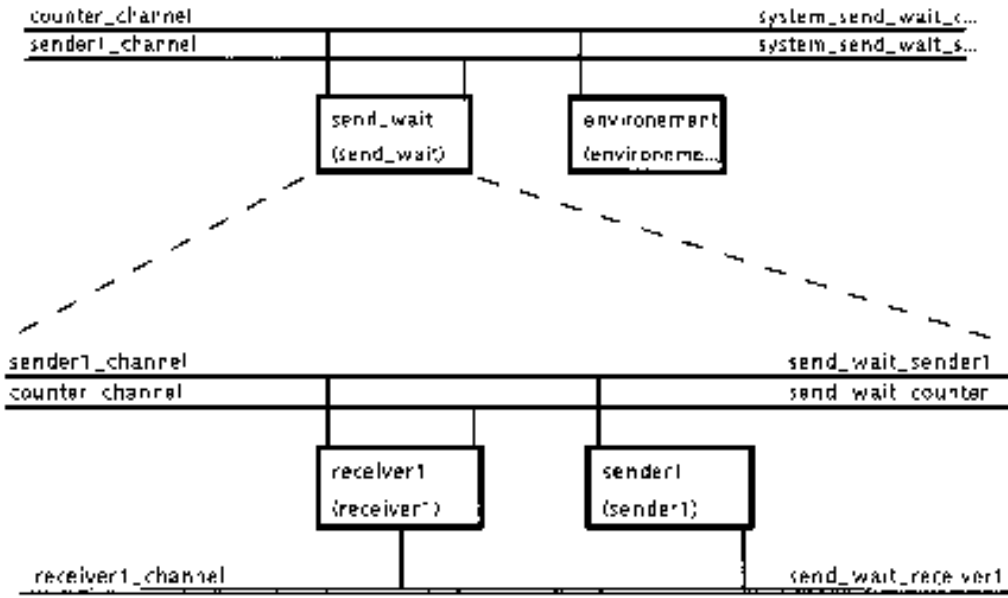
(C)VERILOG	send_wait.pr	DEC/16/1995	4(1.1)


PROCESS system_send_wait/send_wait/sender1
Wait_Ack
nack
(C)VERILOG
send_wait.pr
DEC/16/1995
5(1.1)


BLOCK system_send_wait/environement
(C)VERILOG
send_wait.pr
DEC/16/1995
6

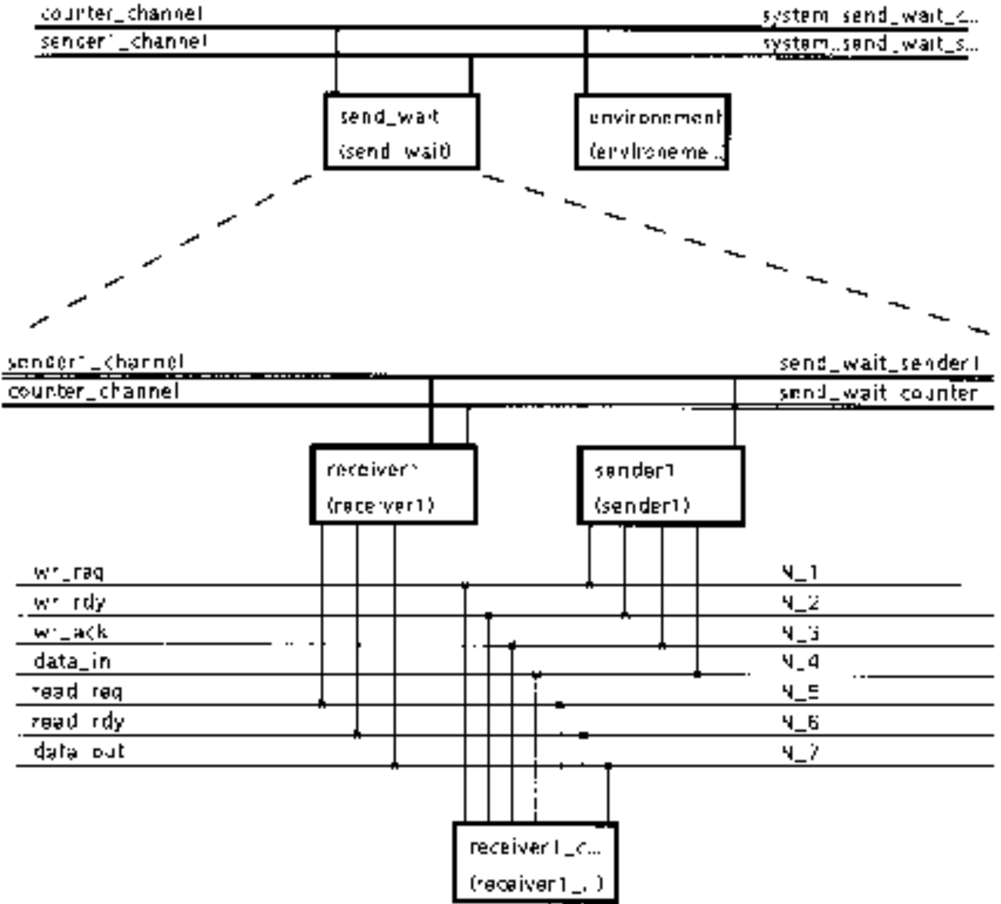

PROCESS system_send_wait/environement/counter
wait_count
wait_count
wait_count
(C)VERILOG
send_wait.pr
DEC/16/1995
7(1.1)

Représentations du système graphiquement en Solar

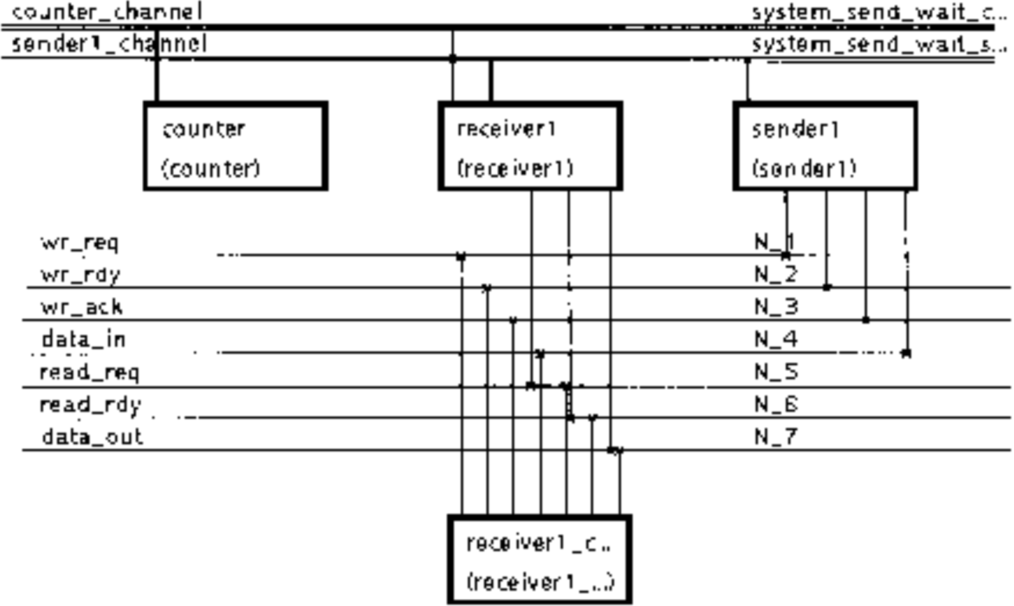


Représentation graphique de la description en Solar du système émetteur/récepteur.

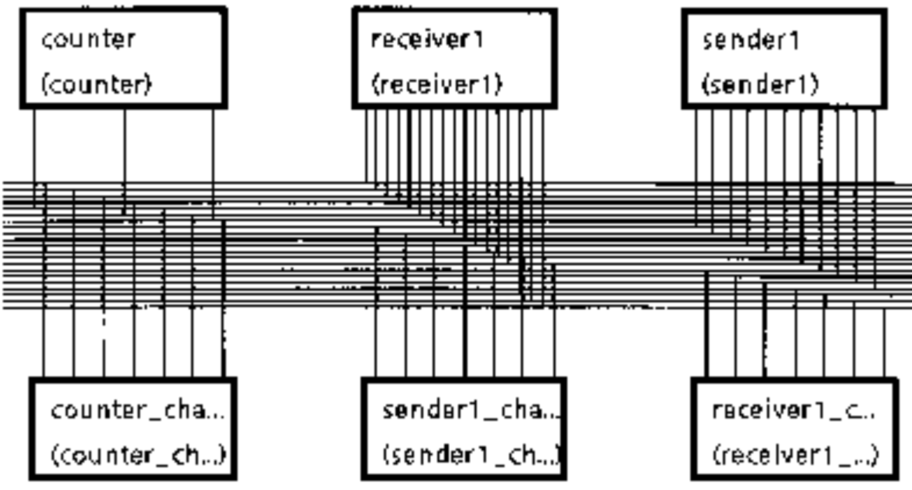
L'unité "send_wait" est hiérarchique et contient les deux unités (processus) d'émission et de réception. Les différentes unités sont connectées à travers des canaux de Solar. Dans cette représentation et celles qui suivent, la convention suivante a été utilisée : les canaux et signaux externes sont représentés en haut des unités alors que les canaux et signaux internes sont représentés en bas.



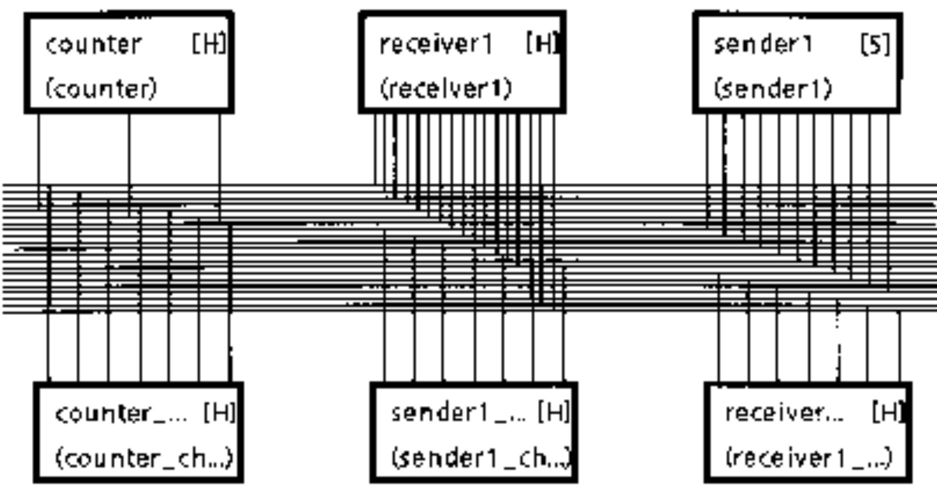
Représentation graphique de l'unité "send_wait" après application de l'opération "MAP" sur le canal appelé "receiver1_channel" qui est montré dans la figure précédente.



Représentation graphique de tout le système après deux applications successives de l'opération "FLAT" sur l'unité "send_wait" et l'unité "environement". Cette opération "FLAT" est appliquée à une unité à la fois et met à plat un seul niveau de sa hiérarchie.



Représentation graphique de tout le système après deux applications successives de l'opération "MAP" sur le canal "sender1_channel" et le canal "counter_channel" de la figure précédente. Le résultat est un système composé d'unités interconnectées uniquement à travers des signaux et sans canaux.



Représentation graphique de tout le système après affectation de chaque unité à une réalisation logicielle ou matérielle.

La notation "[S]" veut dire que l'unité correspondante est affectée à une réalisation en logiciel. De la même manière la notation "[H]" veut dire que l'unité correspondante est affectée à une réalisation en matériel.

Description de l'Émetteur générée automatiquement en C "DU sender1.c"

```
1     /* C-File DU_sender1.c */
2     /* DUObject_CC ViewObject GetViewtype "behaviour" */
3     /*DUObject_CC DU_sender1*/
4     *******************************************************/
5     *includesDUObject_CC*/
6     #include "DU_sender1.h"
7
8     *******************************************************
9
10    int PCALL = 1 ;
11
12    *******************************************************/
13    /*processDUObject_CC*/
14    int DU_sender1()
15    {
16            /*interfaceProcessDUObject_CC DU_sender1itf.vol*/
17        /* IfaceObject_CC */
18        /* ViewObject_CC */
19        /*defineParamCallProcedureObject_CC*/
20        INTEGER dataput_signal;
21        INTEGER process_idput_signal;
22        /*defineParamCallProcedureObject_CC*/
23        INTEGER dataget_signal;
24        /*******************************************************/
25        /* ContentsObject_CC*/
26        /*sender1_behaviourDU_sender1*/
27        INTEGER count  =  1 ;  /*ConstObject_CC*/
28    #define _count  1
29        INTEGER restart  =  2 ;  /*ConstObject_CC*/
30    #define _restart  2
31        INTEGER nack  =  4 ;  /*ConstObject_CC*/
32    #define _nack  4
33        INTEGER ack  =  5 ;  /*ConstObject_CC*/
34    #define _ack  5
35        INTEGER next_to_send;  /*VarObject_CC*/
36        /* STObject_CC_DECLAREOpe */
37        typedef enum
38        {
39          statetable_sender1IDLEState,
40          initialState,wait_ackState
41        } statetable_sender1StateType;
42        statetable_sender1StateType statetable_sender1NextState = initialState;
43        statetable_sender1StateType INITIALState = statetable_sender1NextState;
44        int AGAIN = 1;
45        int ToSend = 0;
46        int ToReceive = 1;
47        while (AGAIN == 1)
48        {
49          if (AGAIN == 1)
50          {
51            if (ToSend == 1) /*TestItToSend*/
52            {
53              DU_sender1itfSendOUT(IPCKEY);
54              ToSend = 0;
55            }
56            else
57            {
58              DU_sender1itfReceiveIN(IPCKEY);
59            }
60          } /*end if (AGAIN == 1) */
61          switch(statetable_sender1NextState) /* STObject_CC */
62          {
63            case initialState : /*STRINGOpe_StateObject_CC*/
64            {
65              next_to_send = ( 0  ); /* AssignObject_CC */
66              process_idput_signal =  2 ;/*PARASSIGFlagSTRINGOpeParamassObject_CC*/
67              put_signalProc(&(next_to_send),&process_idput_signal ); /*PCallObject_CC*/
68              statetable_sender1NextState = wait_ackState; /* NxtstObject_CC */
69            } break;/*BREAKCompCond*/
70            case wait_ackState : /*STRINGOpe_StateObject_CC*/
71            {
72              get_signalProc(&(signal)); /*PCallObject_CC*/
73              /*CaseObject_CC*/
74               if ((signal)==(restart) ) /* AltObject_CC */
75              {
76                paramput_signal =  111 ;/*PARASSIGFlagSTRINGOpeParamassObject_CC*/
77                process_idput_signal =  2 ;/*PARASSIGFlagSTRINGOpeParamassObject_CC*/
78                put_signalProc(&paramput_signal ,&process_idput_signal ); /*PCallObject_CC*/
79                next_to_send = (  1 ); /* AssignObject_CC */
80                statetable_sender1NextState = wait_ackState; /* NxtstObject_CC */
81              }
82              else
```

```
83              if ((signal1==lack) ) /* AltObject_CC */
84              {
85                next_to_send = ((next_to_send)+( 1 ) ); /* AssignObject_CC */
86                process_idput_signal =  2 ;/*PARASSIGFlagSTRINGOpeParamassObject_CC*/
87                put_signalProc(&(next_to_send),&process_idput_signal ); /*PCallObject_CC*/
88                statetable_sender1NextState = wait_ackState; /* NxtstObject_CC */
89              }
90              else
91              if ((signal1==lnack) ) /* AltObject_CC */
92              {
93                process_idput_signal =  1 ;/*PARASSIGFlagSTRINGOpeParamassObject_CC*/
94                put_signalProc(&(next_to_send),&process_idput_signal ); /*PCallObject_CC*/
95                statetable_sender1NextState = wait_ackState; /* NxtstObject_CC */
96              }
97              else
98              ;
99             } break;/*BREAKCompCond*/
100        }
101      } /* end while (AGAIN == 1) */
102      return AGAIN;
103      /*************************************************************/
104      /* ContentsObject_CC*/
105      /*sender1_behaviourDC_sender1*/
106       /*endProcessDCObject_CC*/
107   }
108   /***********************************************************/
109   /* end DC_sender1.c */
```

Description du reste du système générée automatiquement en VHDL "send_receive.vhd"

```
1        ----- ------------------        --------        ----------
2          Generated by AMICAL transVHDL (TAT) V 3.0 of 01-JUL-1994
3          Design Unit  DU_counter
4            ------------------------    -
5
6       library mvl_7;
7       use mvl 7.TYPES.all;
8       use mvl 7.arithmetic.all;
9                  ---------------------------------
10       library SYNOPSYS;
11       use SYNOPSYS.attributes.all;
12       library WORK;
13       use WORK.send_waitpkg.all;
14       --------------          ------------------
15       -----------------            ---------------
16       entity   DU_counter is
17      generic( LOCKBY : INTEGER:=1);
18         Port (
19              wr_req:  IN BIT_VECTOR(1 to  2 );
20              wr_rdy:  OUT BIT_VECTOR(1 to  2 );
21              wr_ack:  OUT BIT_VECTOR(1 to  2 );
22              data_in:  IN INTEGER_VECTOR(1 to  2 );
23                     read_req :  OUT BIT;
24                     read_rdy :  IN BIT;
25                     data_out :  IN INTEGER       );
26       end DU_counter;
27      -------------      ----------------------
28       architecture   counter_behaviour of DU_counter is
29       begin
30         process
31         ---------------       --------------------
32           variable PCALL : INTEGER := 1 ;
33           type get_signalSTATE_TYPE is ( PIDLE,  request, wait_rdy  get );
34           variable get_signalNEXT_STATE  :  get_signalSTATE_TYPE  := request;
35           --defineParamCallProcedureObject VV
36           variable dataget_signal :  INTEGER;
37           procedure get_signal( variable data : inout INTEGER );
38           procedure get_signal( variable data : inout INTEGER ) is
39           begin
40             ----------------------                    ---
41             PCALL := 1;
42             while (PCALL = 1) loop
43                 ------------------------------
44               case get_signalNEXT_STATE is
45                     --------------------------------
46                 when( request ) =>
47                     read_req <=  '1';
48                   get_signalNEXT_STATE  :=  wait_rdy ;
49                 when( wait_rdy ) =>
50                                 if NOT((  read_rdy=  '1') ) then
51                     wait  until (  read_rdy=  '1');
52                   end if;
53                   get_signalNEXT_STATE  :=  get ;
54                 when( get ) =>
55                    data :=  data_out;
56                    read_req <=  '0';
57                   get_signalNEXT_STATE  := PIDLE ;
58                 when others =>
59                   PCALL := 0 ;
60                   get_signalNEXT_STATE  :=  request ;
61               ------       ----------------------------
62               end case ;
63             end loop;
64             -----       ----------------------------
65           end  get_signal;
66           -------       ---------------------------
67           constant  count:  INTEGER := (  1  ) ;
68           constant  restart:  INTEGER := (  2  ) ;
69           constant  nack:  INTEGER := (  4  ) ;
70           constant  ack:  INTEGER := (  5  ) ;
71           variable  nb_count :  INTEGER ;
72           type statetable_counterSTATE_TYPE is ( initial, wait_count );
73           variable statetable_counterNEXT_STATE  :  statetable_counterSTATE_TYPE  := initial
        ;
74           --------------    ----------------------
75         begin
76           -------------    -----------------------
77
78           case statetable_counterNEXT_STATE is
79           -------   ----------------------------
80             when( initial ) =>
81                nb_count :=  0 ;
```

```
82              statetable_counter:NEXT_STATE  := wait_count ;
83           when( wait_count ) =>
84              get_signal( signal );
85        if ( signal= count)
86        then
87              nb_count := ( nb_count + 1 );
88              statetable_counter:NEXT_STATE  := wait_count ;
89    else
90        if ( signal= restart)
91        then
92              nb_count  := 0 ;
93              statetable_counter:NEXT_STATE  := wait_count ;
94    end if;
95    end if;
96
97          ---------------------------------------
98          end case ;
99          ---------------------------------------
100       end process ;
101     end counter_behaviour;
102         -----------------------------------
103     library SYNOPSYS;
104     use SYNOPSYS.attributes.all;
105     library WORK;
106     use WORK.send_waitpkg.all;
107     ------------           -----------------
108     ----------             -----------------
109     entity   DU_receiver1 is
110    generic( IPCKEY : INTEGER:=-1);
111       Port (
112            wr_req:  IN BIT_VECTOR(1 to  2 );
113            wr_rdy:  OUT BIT_VECTOR(1 to  2 );
114            wr_ack:  OUT BIT_VECTOR(1 to  2 );
115            data_in:  IN INTEGER_VECTOR(1 to  2 );
116                  read_req :  OUT BIT;
117                  read_rdy :  IN BIT;
118                  data_out :  IN INTEGER       );
119     end DU_receiver1;
120          ---------------------------------
121     architecture   receiver1_behaviour of DU_receiver1 is
122     begin
123       process
124                  ------------------------------
125         variable PCALL : INTEGER := 1 ;
126         type get_signalSTATE_TYPE is ( PIDLE,  request, wait_rdy, get );
127         variable get_signalNEXT_STATE  :  get_signalSTATE_TYPE  := request;
128           defineParamCallProcedureObject_VV
129         variable dataget_signal :  INTEGER;
130         procedure get_signal( variable data : inout INTEGER );
131         procedure get_signal( variable data : inout INTEGER ) is
132         begin
133                 --------------------------------
134           PCALL := 1;
135           while (PCALL = 1) loop
136                      ------------------------------
137             case get_signalNEXT_STATE is
138                       --------------------------
139               when( request ) =>
140                  read_req <=  '1';
141                 get_signalNEXT_STATE  :=  wait_rdy ;
142               when( wait_rdy ) =>
143                            if NOT((  read_rdy=  '1') ) then
144                   wait  until ( read_rdy=  '1');
145                 end if;
146                 get_signalNEXT_STATE  :=  get ;
147               when( get ) =>
148                  data :=  data_out;
149                  read_req <=  '0';
150                 get_signalNEXT_STATE  := PIDLE ;
151               when others =>
152                 PCALL := 0 ;
153                 get_signalNEXT_STATE  :=  request ;
154             ----------        ---------------------
155             end case ;
156           end loop;
157           ------          ---------------------------
158         end  get_signal;
159         -----        ----------------------------
160         type put_signalSTATE_TYPE is ( PIDLE,  request, wait_wrdy, submit );
161         variable put_signalNEXT_STATE  :  put_signalSTATE_TYPE  := request;
162           defineParamCallProcedureObject_VV
163         variable dataput_signal :  INTEGER;
```

```
164         variable process_idput_signal :  INTEGER;
165         procedure put_signal( variable data : inout INTEGER; variable process_id : inout I
    NTEGER );
166         procedure put_signal( variable data : inout INTEGER; variable process_id : inout I
    NTEGER ) is
167         begin
168           ----                    -  ---------
169           PCALL := 1;
170           while (PCALL = 1) loop
171           ----------------------------------------
172             case put_signalNEXT_STATE is
173             ----------------------------------------
174               when( request ) =>
175                 wr_req( process_id ) :=  '1';
176                 put_signalNEXT_STATE  :=  wait_wrdy ;
177               when( wait_wrdy ) =>
178                               if NOT(( wr_rdy( process_id )=  '1') ) then
179                   wait  until ( wr_rdy( process_id )=  '1');
180                 end if;
181                 data_in( process_id ) :=  data;
182                 put_signalNEXT_STATE  :=  submit ;
183               when( submit ) =>
184                               if NOT(( wr_ack( process_id )=  '1') ) then
185                   wait  until ( wr_ack( process_id )=  '1');
186                 end if;
187                 wr_req( process_id ) :=  '0';
188                 put_signalNEXT_STATE  := PIDLE ;
189               when others =>
190                 PCALL := 0 ;
191                 put_signalNEXT_STATE  :=  request ;
192             ----------------------------------------
193             end case ;
194           end loop;
195             ----------------------------------------
196         end  put_signal;
197              ----------------------------------
198         constant  count:  INTEGER := (  1  ) ;
199         constant  restart:  INTEGER := (  2  ) ;
200         constant  nack:  INTEGER := (  4  ) ;
201         constant  ack:  INTEGER := (  5  ) ;
202         variable  next_to_receive :  INTEGER ;
203         variable  num_tick :  INTEGER ;
204         variable  temp :  INTEGER ;
205         type statetable_receiverSTATE_TYPE is ( initial, wait_data );
206         variable statetable_receiverNEXT_STATE  :  statetable_receiverSTATE_TYPE  := ini
    tial;
207         ------------------------------
208       begin
209         -----------------------------------
210 
211         case statetable_receiverNEXT_STATE is
212         ----------------------------
213           when( initial ) =>
214               next_to_receive :=  0 ;
215               num_tick :=  0 ;
216             statetable_receiverNEXT_STATE  :=  wait_data ;
217           when( wait_data ) =>
218             get_signal(  temp );
219             if (  temp=  111 ) then
220                 next_to_receive :=  0 ;
221                 num_tick :=  0 ;
222               statetable_receiverNEXT_STATE  :=  wait_data ;
223 
224             else
225               if (  temp=  next_to_receive) then
226                  next_to_receive := ( next_to_receive +  1 );
227                 process_idput_signal :=  1 ;  PCallObject genBVHDL
228                 put_signal(  ack, process_idput_signal );
229                  num_tick := ( num_tick +  1 );
230                 if (  num_tick=  10 ) then
231                   process_idput_signal :=  1 ;  PCallObject genBVHDL
232                   put_signal(  count, process_idput_signal );
233                   statetable_receiverNEXT_STATE  :=  wait_data ;
234 
235                 else
236                   statetable_receiverNEXT_STATE  :=  wait_data ;
237 
238                 end if;
239 
240               else
241                 process_idput_signal :=  1 ;  PCallObject genBVHDL
242                 put_signal(  nack, process_idput_signal );
```

```
243                   statetable_receiver1NEXT_STATE  :=  wait_data ;
244
245                 end if;
246
247               end if;
248
249           ---------------
250           end case ;
251             --------------------------------
252         end process ;
253      end receiver1_behaviour;
254             --------------------------------
255      library SYNOPSYS;
256      use SYNOPSYS.attributes.all;
257      library WORK;
258      use WORK.send_waitpkg.all;
259           ---------------------------------
260             -------------------------------
261      entity   DU_sender1 is
262     generic( IPCKEY : INTEGER:=1);
263        Port (
264             wr_req:  IN BIT_VECTOR(1 to  3 );
265             wr_rdy:  OUT BIT_VECTOR(1 to  3 );
266             wr_ack:  OUT BIT_VECTOR(1 to  3 );
267             data_in:  IN INTEGER_VECTOR(1 to  3 );
268                   read_req :  OUT BIT;
269                   read_rdy :  IN BIT;
270                   data_out :  IN INTEGER       );
271      end DU_sender1;
272     ----------------------------------------
273      architecture   sender1_behaviour of DU_sender1 is
274      begin
275        process
276         -----------------------------------------
277           variable PCALL : INTEGER := 1 ;
278           type put_signalSTATE_TYPE is ( FIDLE,  request, wait_wrdy, submit );
279           variable put_signalNEXT_STATE  : put_signalSTATE_TYPE  := request;
280             defineParamCallProcedureObject VV
281           variable dataput_signal :  INTEGER;
282           variable process_idput_signal :  INTEGER;
283           procedure put_signal( variable data : inout INTEGER; variable process_id : inout I
    NTEGER );
284           procedure put_signal( variable data : inout INTEGER; variable process_id : inout I
    NTEGER ) is
285           begin
286             ----- --            .          ----------
287             PCALL := 1;
288             while (PCALL = 1) loop
289             -----          -- -- .    . ----------
290               case put_signalNEXT_STATE is
291               ----------                         ..
292                 when( request ) =>
293                   wr_req( process_id ) <=  '1';
294                   put_signalNEXT_STATE  :=  wait_wrdy ;
295                 when( wait_wrdy ) =>
296                             if NOT(( wr_rdy( process_id )=  '1') ) then
297                     wait  until ( wr_rdy( process_id )=  '1');
298                   end if;
299                   data_in( process_id ) <=  data;
300                   put_signalNEXT_STATE  :=  submit ;
301                 when( submit ) =>
302                             if NOT(( wr_ack( process_id )=  '1') ) then
303                     wait  until ( wr_ack( process_id )=  '1');
304                   end if;
305                   wr_req( process_id ) <=  '0';
306                   put_signalNEXT_STATE  :=  FIDLE ;
307                 when others =>
308                   PCALL := 0 ;
309                   put_signalNEXT_STATE  :=  request ;
310               -------------------     ----        ----
311               end case ;
312             end loop;
313             --------------------              ----
314           end  put_signal;
315           --------------------        . . -----
316           type get_signalSTATE_TYPE is ( FIDLE,  request, wait_rrdy, get );
317           variable get_signalNEXT_STATE  :  get_signalSTATE_TYPE  := request;
318           --defineParamCallProcedureObject VV
319           variable dataget_signal :  INTEGER;
320           procedure get_signal( variable data : inout INTEGER );
321           procedure get_signal( variable data : inout INTEGER ) is
322           begin
```

```
323          ----------------------------------------
324          PCALL := 1;
325          while (PCALL = 1) loop
326          ----------------------------------------
327            case get_signalNEXT_STATE is
328            -------------
329              when( request ) =>
330                 read_req <= '1' ;
331                get_signalNEXT_STATE  :=  wait_rdy ;
332              when( wait_rdy ) =>
333                                 if NOT((  read_rdy=  '1') ) then
334                  wait  until (  read_rdy=  '1');
335                end if;
336                get_signalNEXT_STATE  :=  get ;
337              when( get ) =>
338                 data :=  data_out;
339                 read_req <=  '0';
340                get_signalNEXT_STATE  := PIDLE ;
341              when others =>
342                PCALL := 0 ;
343                get_signalNEXT_STATE  :=  request ;
344                 .                            -------------
345            end case ;
346          end loop;
347                                   --------------------
348        end  get_signal;
349                                    -----------------
350        constant  count:  INTEGER := (  1  ) ;
351        constant  restart:  INTEGER := (  2  ) ;
352        constant  nack:  INTEGER := (  4  ) ;
353        constant  ack:  INTEGER := (  5  ) ;
354        variable  next_to_send :  INTEGER ;
355        type statetable_sender1STATE_TYPE is ( initial, wait_ack );
356        variable statetable_sender1NEXT_STATE  :  statetable_sender1STATE_TYPE  := initial
    ;
357        ---------------------------------------
358      begin
359        -------------------------------------- .
360
361        case statetable_sender1NEXT_STATE is
362         -----------------------
363          when( initial ) =>
364             next_to_send :=  3 ;
365            process_idput_signal :=   3 ;  PCallObject_genBVHDL
366            put_signal(  next_to_send, process_idput_signal );
367            statetable_sender1NEXT_STATE  :=  wait_ack ;
368          when( wait_ack ) =>
369            get_signal(  signal );
370      if (  signal=  restart)
371      then
372            paramput_signal :=  11 ;--PCallObject genBVHDL
373            process_idput_signal :=  2 ;--PCallObject genBVHDL
374            put_signal( paramput_signal, process_idput_signal );
375             next_to_send := ( - 1 );
376            statetable_sender1NEXT_STATE  :=  wait_ack ;
377   else
378      if (  signal=  ack)
379      then
380             next_to_send := ( next_to_send -  1 );
381            process_idput_signal :=  1 ;--PCallObject_genBVHDL
382            put_signal(  next_to_send, process_idput_signal );
383            statetable_sender1NEXT_STATE  :=  wait_ack ;
384   else
385      if (  signal=  nack)
386      then
387            process_idput_signal :=  1 ;--PCallObject genBVHDL
388            put_signal(  next_to_send, process_idput_signal );
389            statetable_sender1NEXT_STATE  :=  wait_ack ;
390   end if;
391   end if;
392   end if;
393
394         ---------------------------------------
395        end case ;
396      --------------------------------------
397       end process ;
398    end sender1_behaviour;
399         ------------------------------------
400    library SYNOPSYS;
401    use SYNOPSYS.attributes.all;
402    library WORK;
403    use WORK.send_waitpkg.all;
```

```
404    -----------      -------------------------
405    ---------       -------------------------
406    entity   receiver1_channel is
407   generic( IPCKEY : INTEGER:=1);
408      Port (
409           wr_req:  IN BIT_VECTOR(1 to  2 );
410           wr_rdy:  OUT BIT_VECTOR(1 to  2 );
411           wr_ack:  OUT BIT_VECTOR(1 to  2 );
412           data_in:  IN INTEGER_VECTOR(1 to  2 );
413                  read_req :  IN BIT;
414                  read_rdy :  OUT BIT;
415                  data_out :  OUT INTEGER      );
416    end receiver1_channel;
417    ------------------
418    architecture   receiver1_channel_view of receiver1_channel is
419    begin
420      process
421      ----------------------------------  .     ...
422        variable PCALL : INTEGER := 1 ;
423        constant  nb_sender:  INTEGER := (  2  ) ;
424        constant  queue_size:  INTEGER := (  1024  ) ;
425        --IntObject_gen3TypDef--WARNING. maybe type is re-defined here:
426        type INTEGER_VECTOR is array ( NATURAL range <> ) of INTEGER;
427        variable queue :  INTEGER_VECTOR(1 to  queue_size) ;
428        variable  queue_in_ptr :  INTEGER := (  1  ) ;
429        variable  queue_out_ptr :  INTEGER := (  1  ) ;
430        variable  indice :  INTEGER := (  1  ) ;
431        type controllerSTATE_TYPE is ( Init, send_receive );
432        variable controllerNEXT_STATE  :  controllerSTATE_TYPE  := Init;
433        ------------- .             --------------
434      begin
435        --------------- .            ------------
436
437        case controllerNEXT_STATE is
438        ------------------------
439          when( Init ) =>
440              queue_in_ptr := 1 ;
441              queue_out_ptr :=  1 ;
442              indice :=  1 ;
443            controllerNEXT_STATE  :=  send_receive ;
444          when( send_receive ) =>
445     Not_Defined ParamObject_genDVHDL
446
447                  -------------------------------
448        end case ;
449      -----       ---------------------------
450      end process ;
451    end receiver1_channel_view;
452    -----------      ---------------------
453    library SYNOPSYS;
454    use SYNOPSYS.attributes.all;
455    library WORK;
456    use WORK.send_waitpkg.all;
457    --------------------             ------
458    ---------------------          . ---
459    entity   sender1_channel is
460   generic( IPCKEY : INTEGER:=1);
461      Port (
462           wr_req:  IN BIT_VECTOR(1 to  2 );
463           wr_rdy:  OUT BIT_VECTOR(1 to  2 );
464           wr_ack:  OUT BIT_VECTOR(1 to  2 );
465           data_in:  IN INTEGER_VECTOR(1 to  2 );
466                  read_req :  IN BIT;
467                  read_rdy :  OUT BIT;
468                  data_out :  OUT INTEGER      );
469    end sender1_channel;
470   -------------------------           ---
471    architecture   sender1_channel_view of sender1_channel is
472    begin
473      process
474      ----------------------        . ---------
475        variable PCALL : INTEGER := 1 ;
476        constant  nb_sender:  INTEGER := (  2  ) ;
477        constant  queue_size:  INTEGER := (  1024  ) ;
478        --IntObject_gen3TypDef--WARNING. maybe type is re-defined here:
479        type INTEGER_VECTOR is array ( NATURAL range <> ) of INTEGER;
480        variable queue :  INTEGER_VECTOR(1 to  queue_size) ;
481        variable  queue_in_ptr :  INTEGER := (  1  ) ;
482        variable  queue_out_ptr :  INTEGER := (  1  ) ;
483        variable  indice :  INTEGER := (  1  ) ;
484        type controllerSTATE_TYPE is ( Init, send_receive );
485        variable controllerNEXT_STATE  :  controllerSTATE_TYPE  := Init;
```

```
486                                                    ---------
487      begin
488        ----------------------- --     ----------
489
490        case controllerNEXT_STATE is
491                                ------------------
492          when( Init ) =>
493              queue_in_ptr := 1 ;
494              queue_out_ptr := 1 ;
495              indice := 1 ;
496            controllerNEXT_STATE  :=  send_receive ;
497          when( send_receive ) =>
498    -Not Defined_ParactObject_genBVHDL
499
500        ------------------------------------------
501        end case ;
502      ------------------------------------------
503      end process ;
504    end sender1 channel view;
505    ---         --- -----------------------
506    library SYNOPSYS;
507    use SYNOPSYS.attributes.all;
508    library WORK;
509    use WORK.send_waitpkg.all;
510    ------------------------
511    --------------------
512    entity   counter_channel is
513   generic( IPCKEY : INTEGER:=-1);
514      Port (
515          wr_req:  IN BIT_VECTOR(1 to  2 );
516          wr_rdy:  OUT BIT_VECTOR(1 to  2 );
517          wr_ack:  OUT BIT_VECTOR(1 to  2 );
518          data_in:  IN INTEGER_VECTOR(1 to  2 );
519                  read_req :  IN BIT;
520                  read_rdy :  OUT BIT;
521                  data_out :  OUT INTEGER      );
522    end counter channel;
523          ------------------------------
524    architecture   counter_channel_view of counter_channel is
525    begin
526      process
527          ----------------------------------
528        variable FCALL : INTEGER := 1 ;
529        constant  nb sender:  INTEGER := (  2  ) ;
530        constant  queue size:  INTEGER := (  1024  ) ;
531          TorObject genBTypDef--WARNING. maybe type is re-defined here:
532        type INTEGER VECTOR is array ( NATURAL range <> ) of INTEGER;
533        variable queue :  INTEGER VECTOR(1 to  queue_size) ;
534        variable  queue in ptr :  INTEGER := (  1  ) ;
535        variable  queue_out_ptr :  INTEGER := (  1  ) ;
536        variable  indice :  INTEGER := (  1  ) ;
537        type controllerSTATE_TYPE is ( Init, send receive );
538        variable controllerNEXT_STATE  :  controllerSTATE TYPE  := Init;
539        ----------------                          .
540      begin
541        ----------------                           .
542
543        case controllerNEXT_STATE is
544        ------------------------- -
545          when( Init ) =>
546              queue_in_ptr := 1 ;
547              queue_out_ptr := 1 ;
548              indice := 1 ;
549            controllerNEXT_STATE  :=  send_receive ;
550          when( send_receive ) =>
551   --Not Defined_ParactObject_genBVHDL
552
553        -----------------------
554        end case ;
555      --------------------------------
556      end process ;
557    end counter_channel_view;
558   library mvl_7;
559   use mvl_7.TYPES.all;
560   use mvl_7.arithmetic.all;
561   entity DU_system_send_wait is
562      generic( IPCKEY : INTEGER:=-1);
563      Port (
564          );
565   end DU_system_send_wait;
566   architecture  Structure of DU_system_send_wait is
567
```

```
568      signal  K_1 : BIT_VECTOR(1 to  2 ) ;
569      signal  K_2 : BIT_VECTOR(1 to  2 ) ;
570      signal  K_3 : BIT_VECTOR(1 to  2 ) ;
571      signal  K_4 : INTEGER_VECTOR(1 to  2 ) ;
572      signal  K_5 : BIT ;
573      signal  K_6 : BIT ;
574      signal  K_7 : INTEGER ;
575      signal  M_1 : BIT_VECTOR(1 to  2 ) ;
576      signal  M_2 : BIT_VECTOR(1 to  2 ) ;
577      signal  M_3 : BIT_VECTOR(1 to  2 ) ;
578      signal  M_4 : INTEGER_VECTOR(1 to  2 ) ;
579      signal  M_5 : BIT ;
580      signal  M_6 : BIT ;
581      signal  M_7 : INTEGER ;
582      signal  N_1 : BIT_VECTOR(1 to  2 ) ;
583      signal  N_2 : BIT_VECTOR(1 to  2 ) ;
584      signal  N_3 : BIT_VECTOR(1 to  2 ) ;
585      signal  N_4 : INTEGER_VECTOR(1 to  2 ) ;
586      signal  N_5 : BIT ;
587      signal  N_6 : BIT ;
588      signal  N_7 : INTEGER ;
589
590      component  DU_counter
591     generic( IPCKEY : INTEGER:=1);
592      Port (
593            wr_req:  IN BIT_VECTOR(1 to  2 );
594            wr_rdy:  OUT BIT_VECTOR(1 to  2 );
595            wr_ack:  OUT BIT_VECTOR(1 to  2 );
596            data_in:  IN INTEGER_VECTOR(1 to  2 );
597            read_req :  OUT BIT;
598            read_rdy :  IN BIT;
599            data_out :  IN INTEGER        );
600       end component;
601      component  DU_receiver1
602     generic( IPCKEY : INTEGER:=1);
603      Port (
604            wr_req:  IN BIT_VECTOR(1 to  2 );
605            wr_rdy:  OUT BIT_VECTOR(1 to  2 );
606            wr_ack:  OUT BIT_VECTOR(1 to  2 );
607            data_in:  IN INTEGER_VECTOR(1 to  2 );
608            read_req :  OUT BIT;
609            read_rdy :  IN BIT;
610            data_out :  IN INTEGER        );
611       end component;
612      component  counter_channel
613     generic( IPCKEY : INTEGER:=1);
614      Port (
615            wr_req:  IN BIT_VECTOR(1 to  2 );
616            wr_rdy:  OUT BIT_VECTOR(1 to  2 );
617            wr_ack:  OUT BIT_VECTOR(1 to  2 );
618            data_in:  IN INTEGER_VECTOR(1 to  2 );
619            read_req :  IN BIT;
620            read_rdy :  OUT BIT;
621            data_out :  OUT INTEGER        );
622       end component;
623      component  DU_sender1
624     generic( IPCKEY : INTEGER:=1);
625      Port (
626            wr_req:  IN BIT_VECTOR(1 to  2 );
627            wr_rdy:  OUT BIT_VECTOR(1 to  2 );
628            wr_ack:  OUT BIT_VECTOR(1 to  2 );
629            data_in:  IN INTEGER_VECTOR(1 to  2 );
630            read_req :  OUT BIT;
631            read_rdy :  IN BIT;
632            data_out :  IN INTEGER        );
633       end component;
634      component  sender1_channel
635     generic( IPCKEY : INTEGER:=1);
636      Port (
637            wr_req:  IN BIT_VECTOR(1 to  2 );
638            wr_rdy:  OUT BIT_VECTOR(1 to  2 );
639            wr_ack:  OUT BIT_VECTOR(1 to  2 );
640            data_in:  IN INTEGER_VECTOR(1 to  2 );
641            read_req :  IN BIT;
642            read_rdy :  OUT BIT;
643            data_out :  OUT INTEGER        );
644       end component;
645      component  receiver1_channel
646     generic( IPCKEY : INTEGER:=1);
647      Port (
648            wr_req:  IN BIT_VECTOR(1 to  2 );
649            wr_rdy:  OUT BIT_VECTOR(1 to  2 );
```

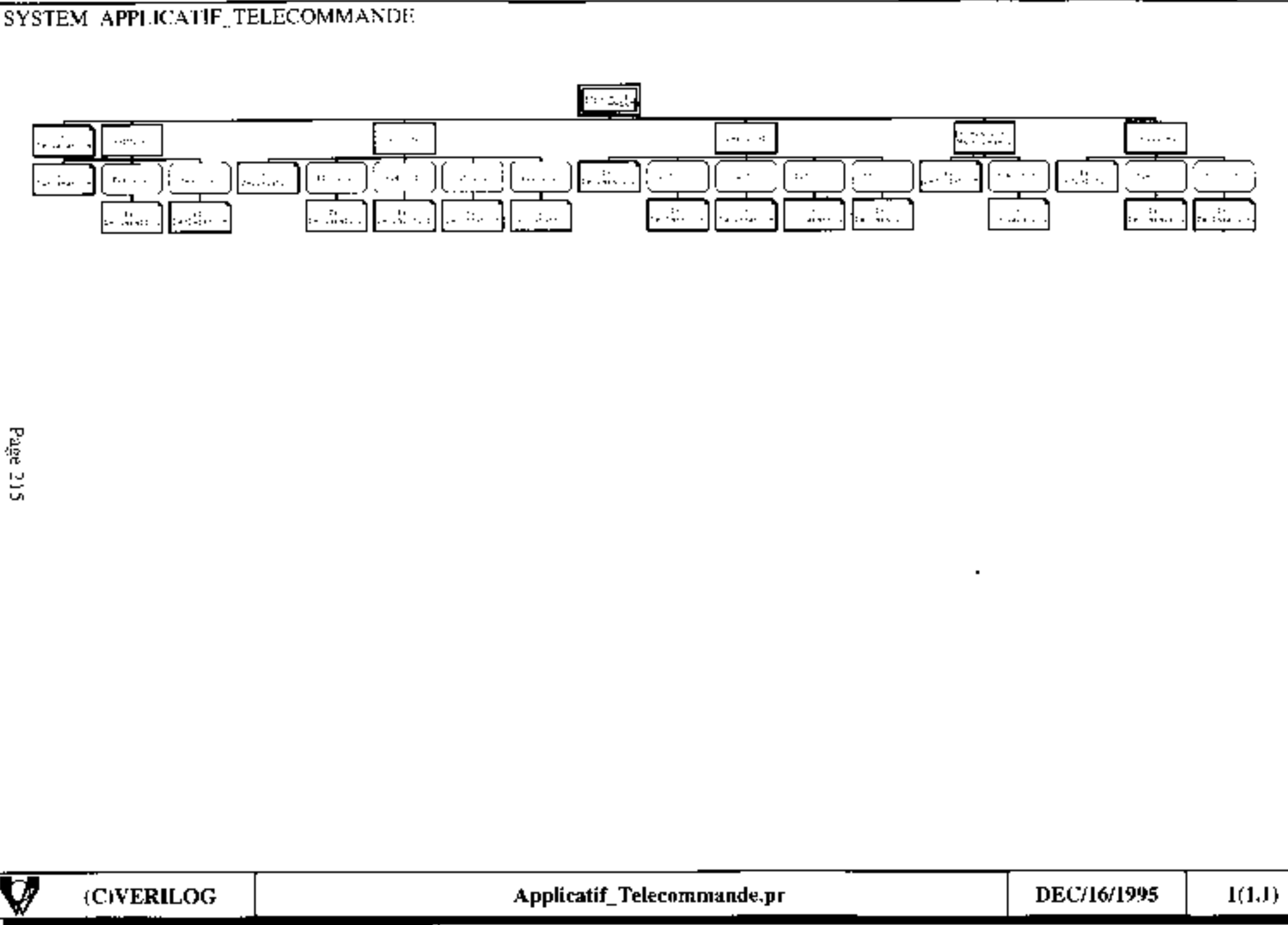
```
650           wr_ack:   OUT BIT_VECTOR(1 to  3 );
651           data_in:   IN INTEGER_VECTOR(1 to  3 );
652           read_req :   IN BIT;
653           read_rdy :   OUT BIT;
654           data_out :   OUT INTEGER        );
655      end component;
656
657     begin
658     counter :   DU_counter
659      Generic Map ( IPCKEY => 1)
660
661      Port Map(
662                   read_req=>N_5,
663                   read_rdy=>N_6,
664                   data_out=>N_7           );
665     receiver1 :   DU_receiver1
666      Generic Map ( IPCKEY => 2)
667
668      Port Map(
669                   wr_req=>N_1,
670                   wr_rdy=>N_2,
671                   wr_ack=>N_3,
672                   data_in=>N_4,
673                   read_req=>N_5,
674                   read_rdy=>N_6,
675                   data_out=>N_7           );
676     counter_channel :   counter_channel
677      Generic Map ( IPCKEY => 3)
678
679      Port Map(
680                   wr_req=>N_1,
681                   wr_rdy=>N_2,
682                   wr_ack=>N_3,
683                   data_in=>N_4,
684                   read_req=>N_5,
685                   read_rdy=>N_6,
686                   data_out=>N_7           );
687     sender1 :   DU_sender1
688      Generic Map ( IPCKEY => 4)
689
690      Port Map(
691                   wr_req=>N_1,
692                   wr_rdy=>N_2,
693                   wr_ack=>N_3,
694                   data_in=>N_4,
695                   read_req=>N_5,
696                   read_rdy=>N_6,
697                   data_out=>N_7           );
698     sender1_channel :   sender1_channel
699      Generic Map ( IPCKEY => 5)
700
701      Port Map(
702                   wr_req=>N_1,
703                   wr_rdy=>N_2,
704                   wr_ack=>N_3,
705                   data_in=>N_4,
706                   read_req=>N_5,
707                   read_rdy=>N_6,
708                   data_out=>N_7           );
709     receiver1_channel :   receiver1_channel
710      Generic Map ( IPCKEY => 6)
711
712      Port Map(
713                   wr_req=>N_1,
714                   wr_rdy=>N_2,
715                   wr_ack=>N_3,
716                   data_in=>N_4,
717                   read_req=>N_5,
718                   read_rdy=>N_6,
719                   data_out=>N_7           );
720     end Structure;
721    configuration DU_system_send_wait_Structure of DU_system_send_wait is
722       for Structure
723         for counter : DU_counter
724         use entity work.DU_counter(counter_behaviour);
725         end for;
726         for receiver1 : DU_receiver1
727         use entity work.DU_receiver1(receiver1_behaviour);
728         end for;
729         for counter_channel : counter_channel
730         use entity work.counter_channel(behaviour);
731         end for;
```

```
732         for sender1 : DD_sender1
733         use entity work.DD_sender1itf(CLI);
734         end for;
735         for sender1_channel : sender1_channel
736         use entity work.sender1_channel(behaviour);
737         end for;
738         for receiver1_channel : receiver1_channel
739         use entity work.receiver1_channel(behaviour);
740         end for;
741       end for;
742     end DD_system_send_wait_Structure;
```

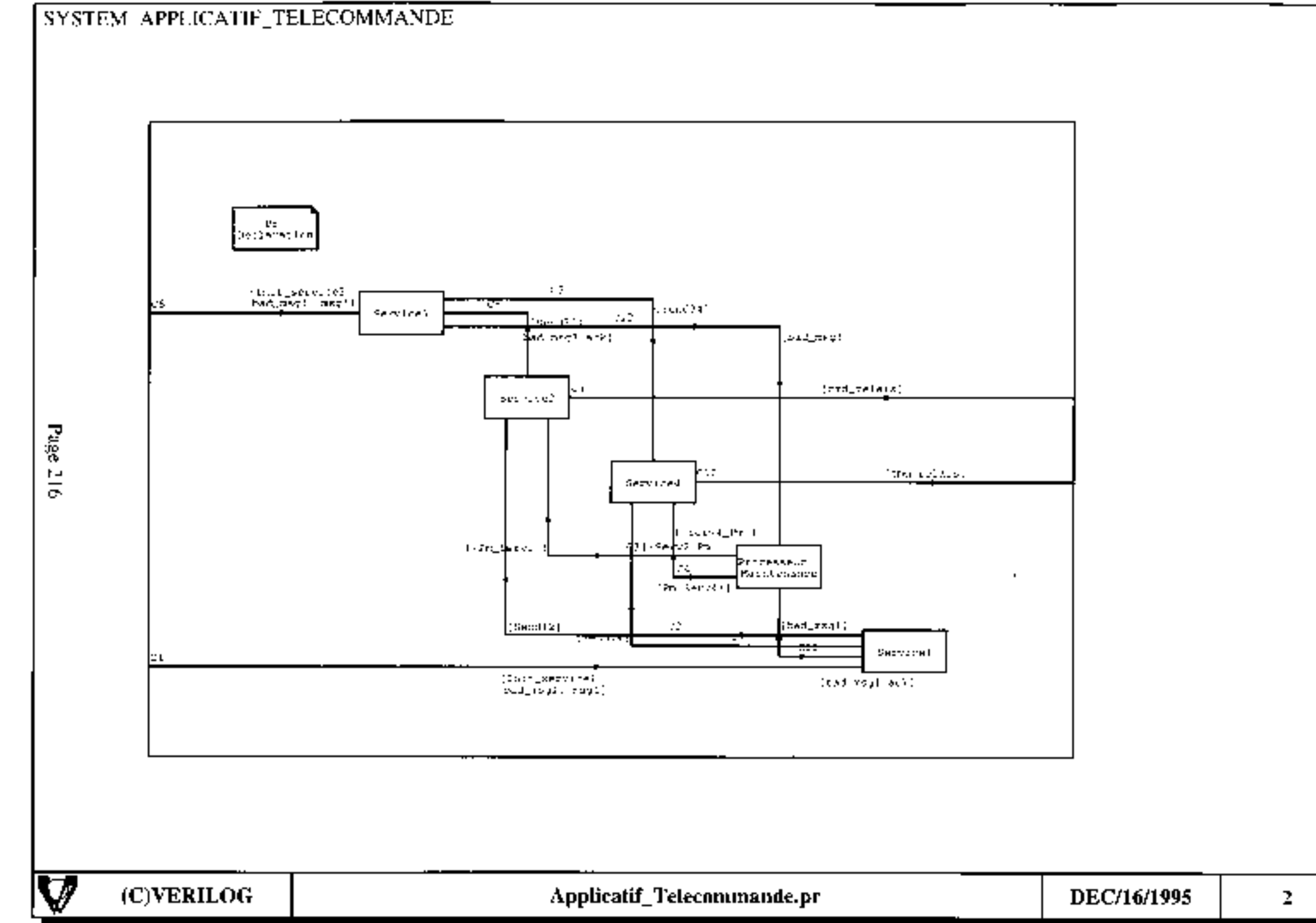
D.2. Bloc télécommande d'un satellite

Dans ce qui suit, l'exemple de l'applicatif télécommande d'un satellite, qui a été décrit dans le chapitre 4 (figure 4.13), sera présenté à travers sa description en SDL. Il s'agit d'un système très simplifié dont l'objet est de prendre en considération les aspects de maintenance dès la phase de spécification. Dans cet exemple les blocs SDL appelés "service1" et "service3" sont identiques. De même les blocs SDL appelés "service2" et "service4" sont identiques. Ils intègrent chacun des processus équivalents qui permettent de réagir en cas de défaillances. Le bloc appelé "Processeur_Maintenance" comporte un processus de maintenance global de tout le système

Description en SDL graphique du bloc, simplifié, de télécommande d'un satellite qui est présenté à la figure 4.13

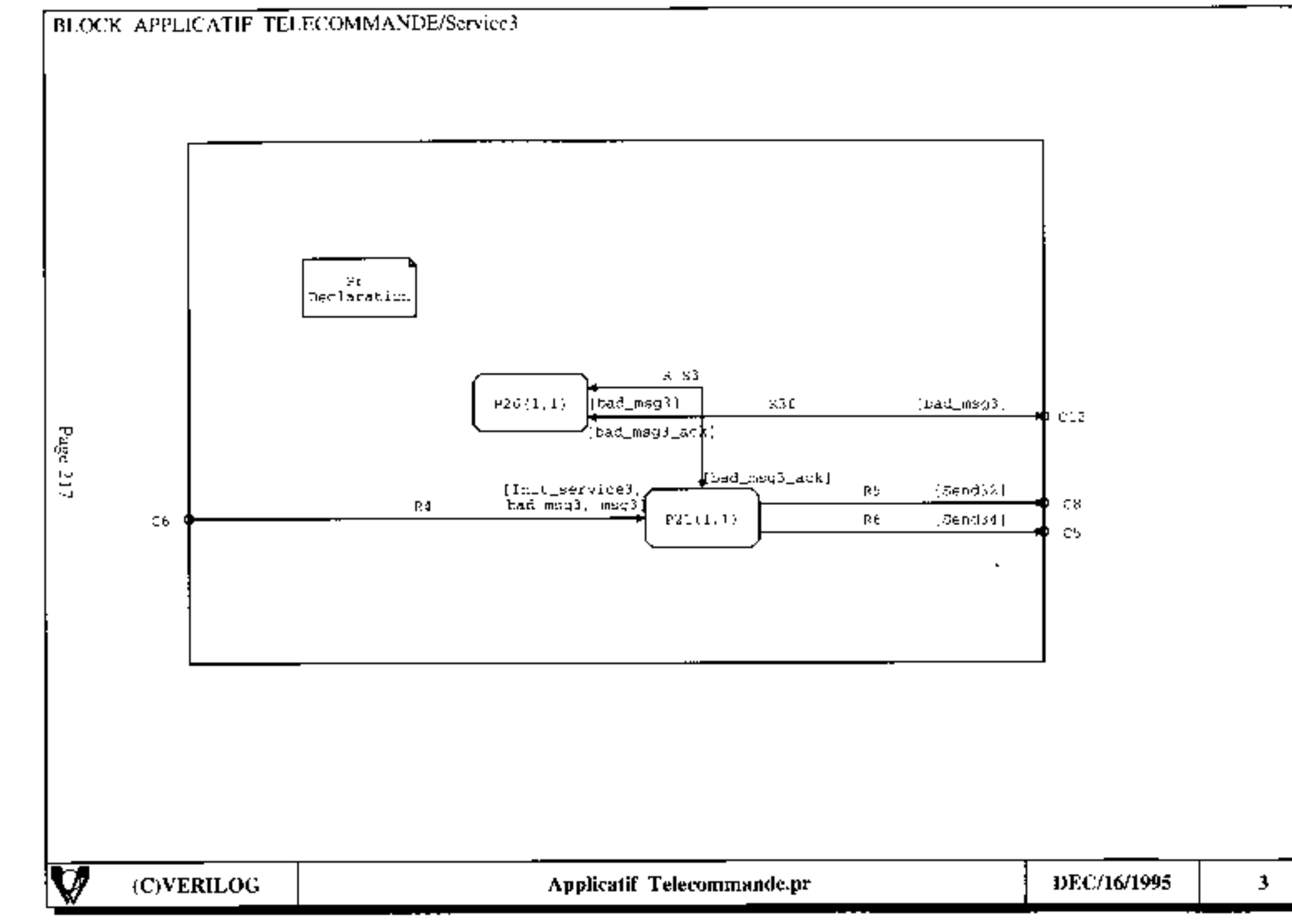

SYSTEM APPLICATIF_TELECOMMANDE
(C)VERILOG
Applicatif_Telecommande.pr
DEC/16/1995
1(1.1)

SYSTEM APPLICATIF_TELECOMMANDE

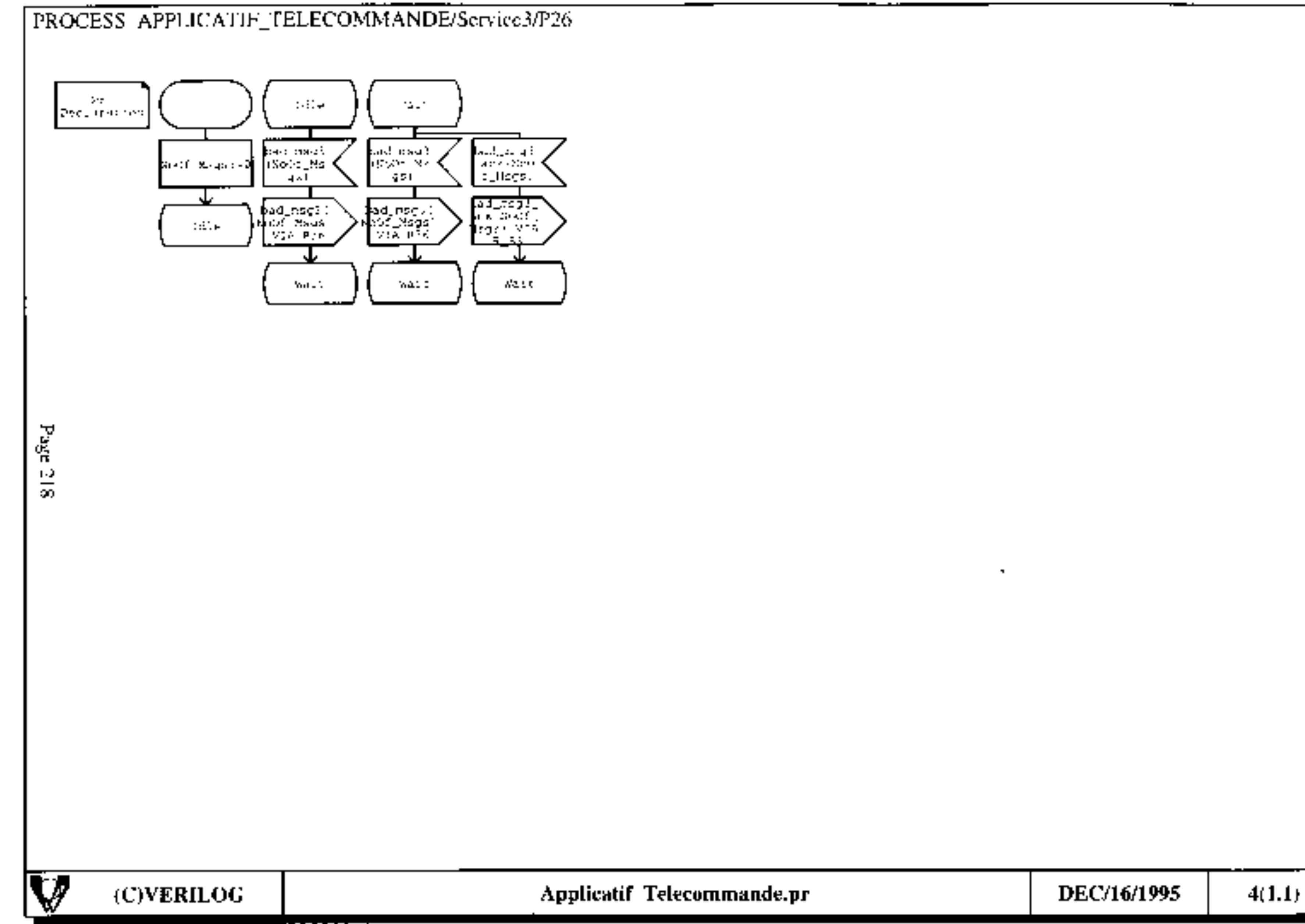


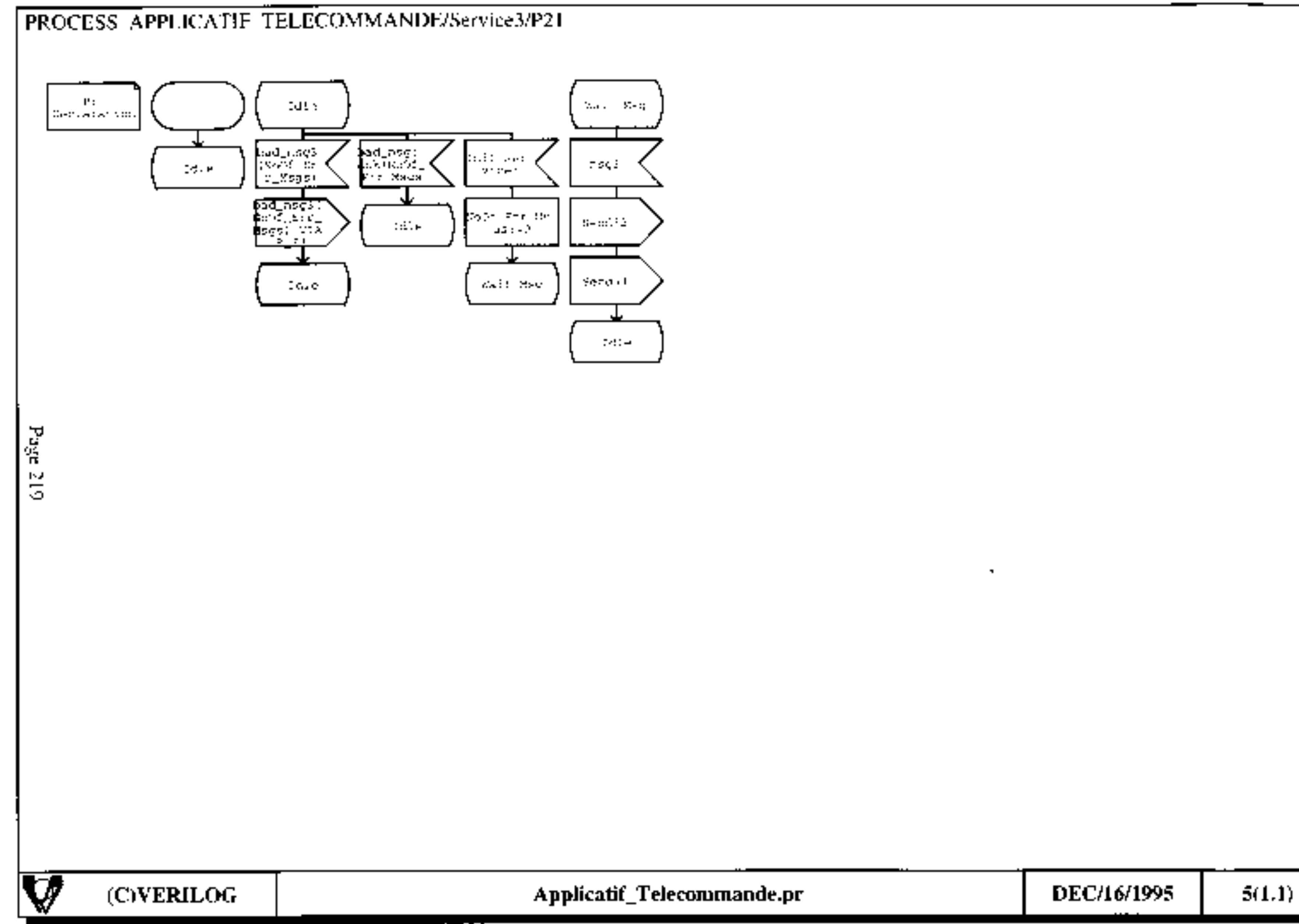
(C)VERILOG	Applicatif_Telecommande.pr	DEC/16/1995	2

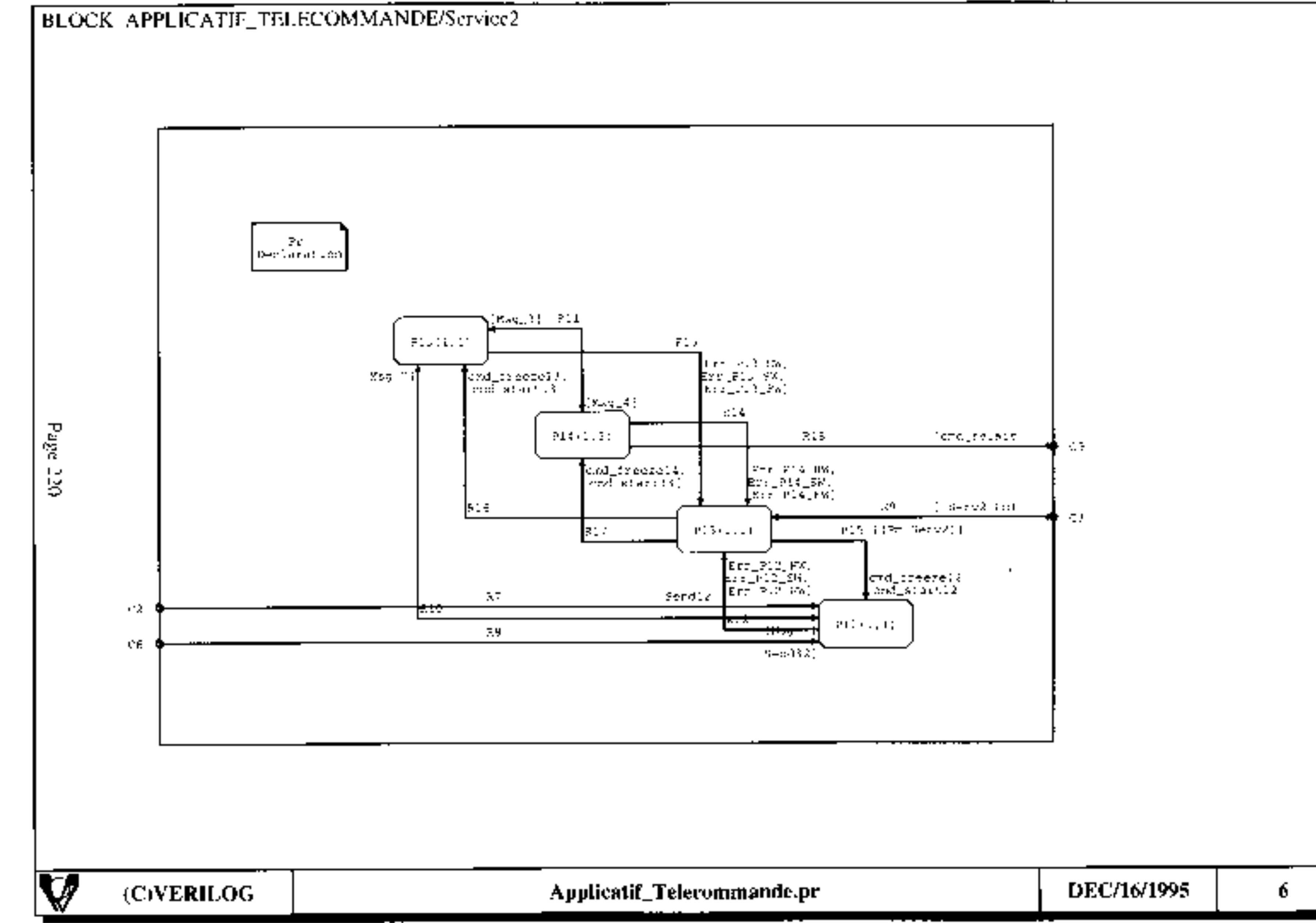
BLOCK APPLICATIF TELECOMMANDE/Service3



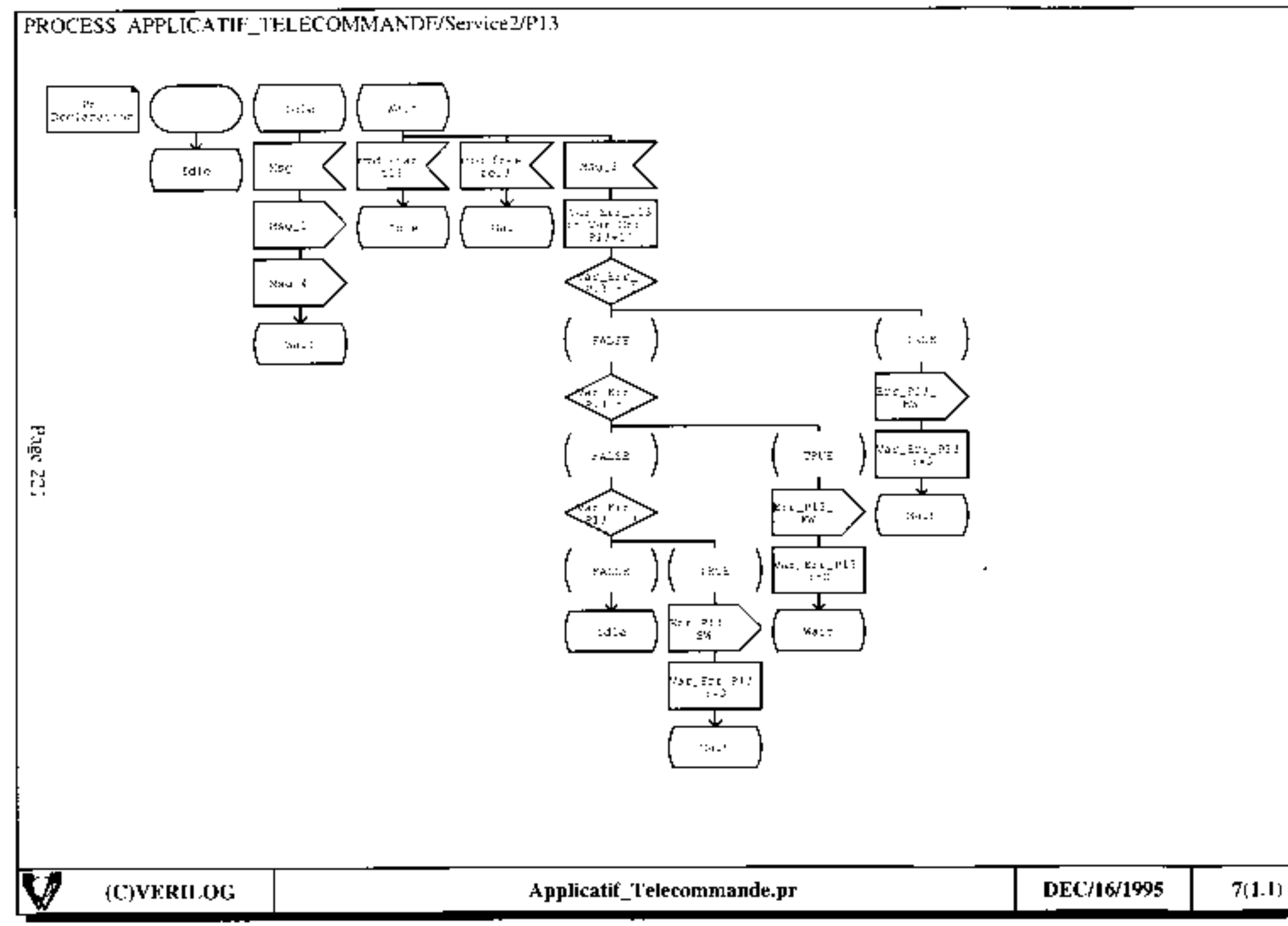
(C)VERILOG | Applicatif Telecommande.pr | DEC/16/1995 | 3


PROCESS APPLICATIF_TELECOMMANDE/Service3/P26
(C)VERILOG
Applicatif Telecommande.pr
DEC/16/1995
4(1.1)

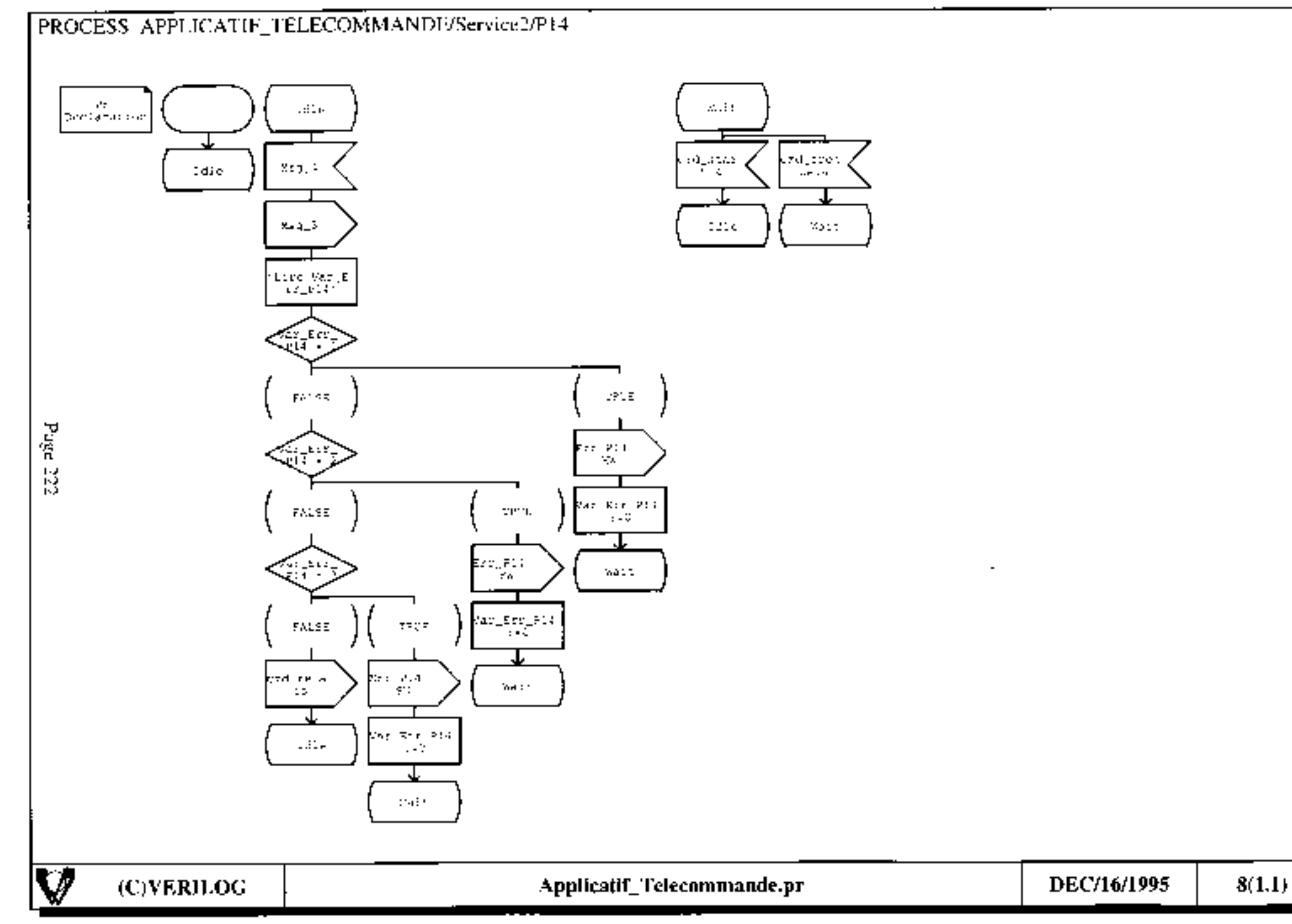

PROCESS APPLICATIF TELECOMMANDE/Service3/P21
(C)VERILOG
Applicatif_Telecommande.pr
DEC/16/1995
5(1.1)


R15
R16
S7
S9
Send12

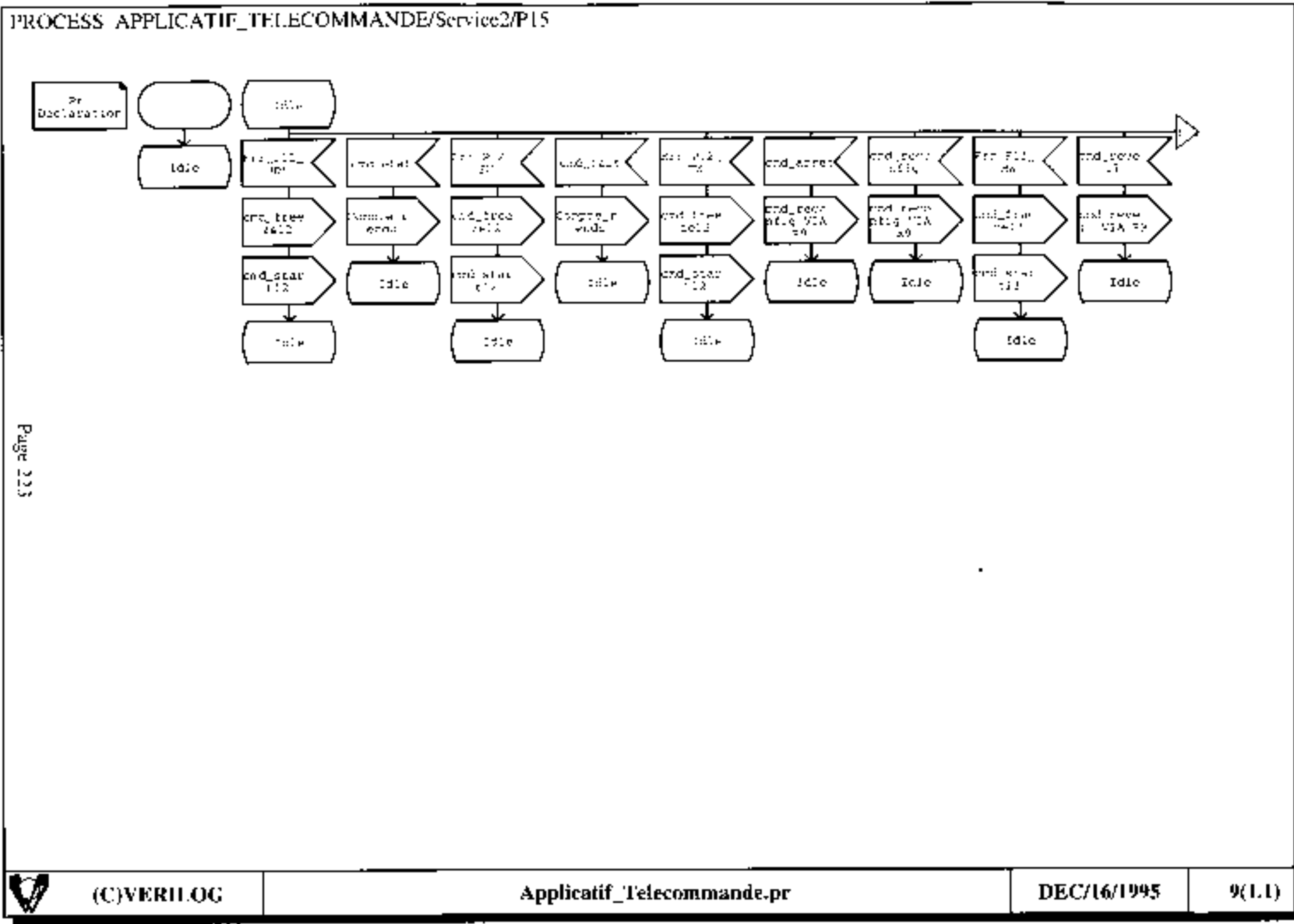
PROCESS APPLICATIF_TELECOMMANDE/Service2/P13



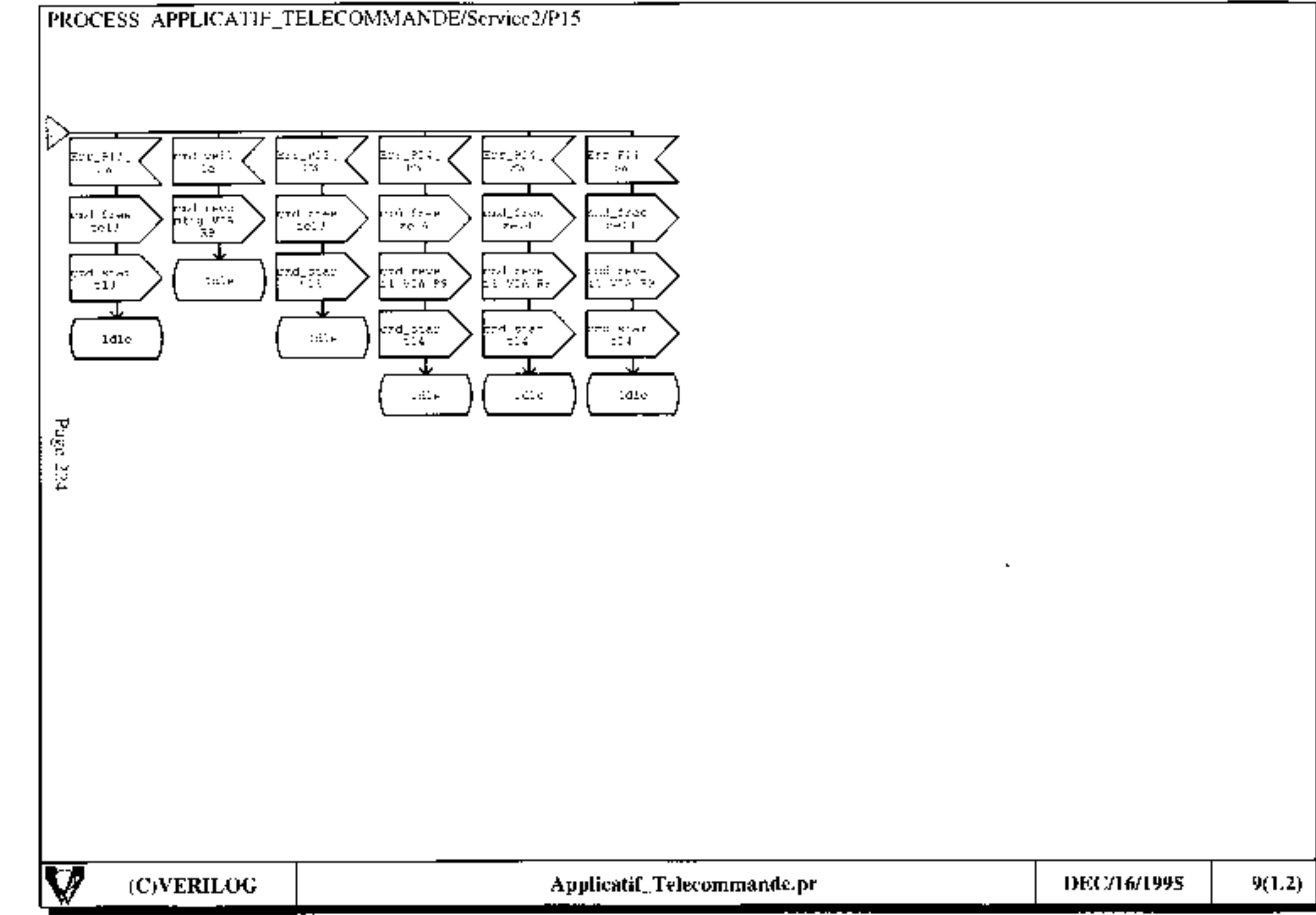
PROCESS APPLICATIF_TELECOMMANDE/Service2/P14



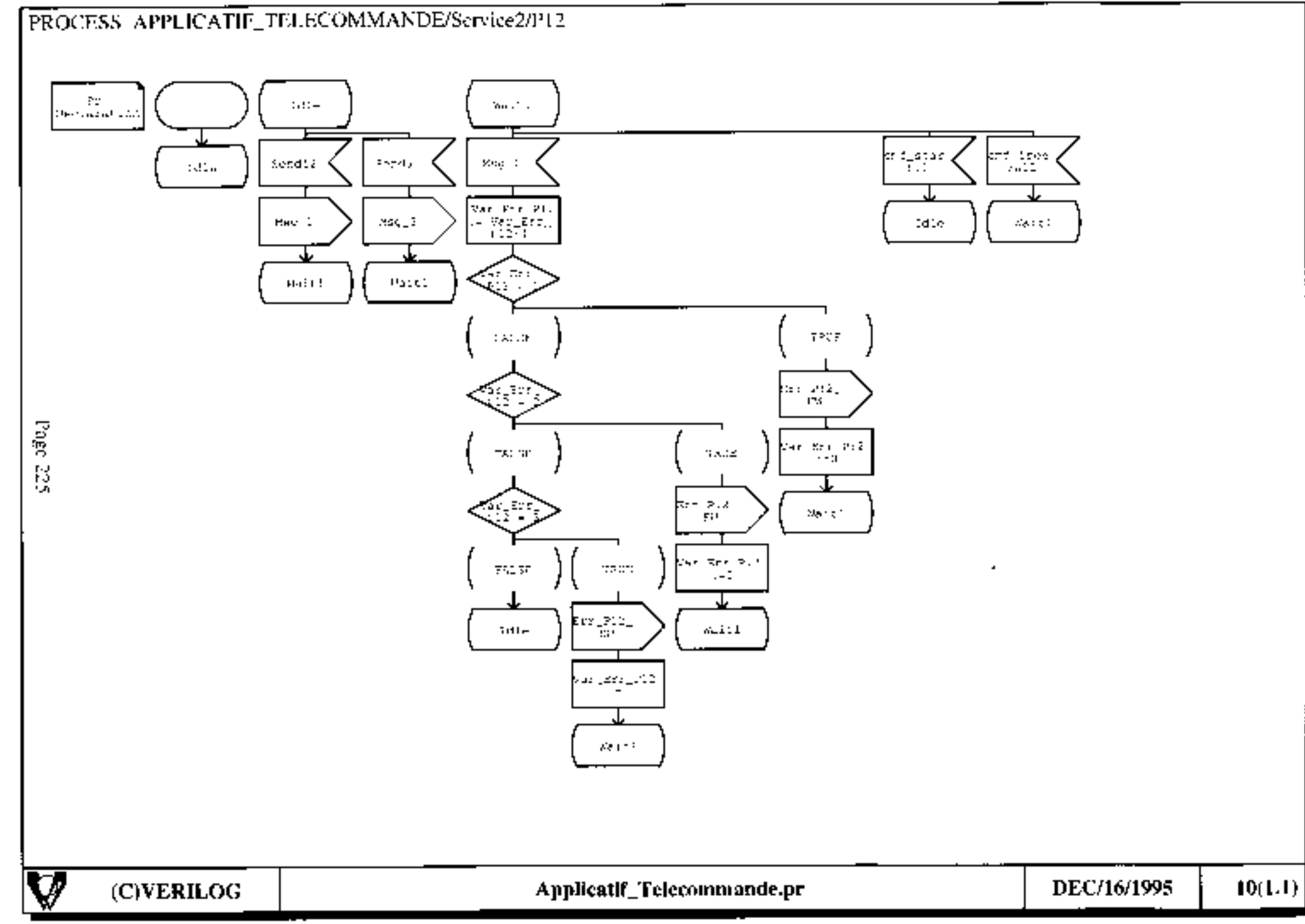
PROCESS APPLICATIF_TELECOMMANDE/Service2/P15



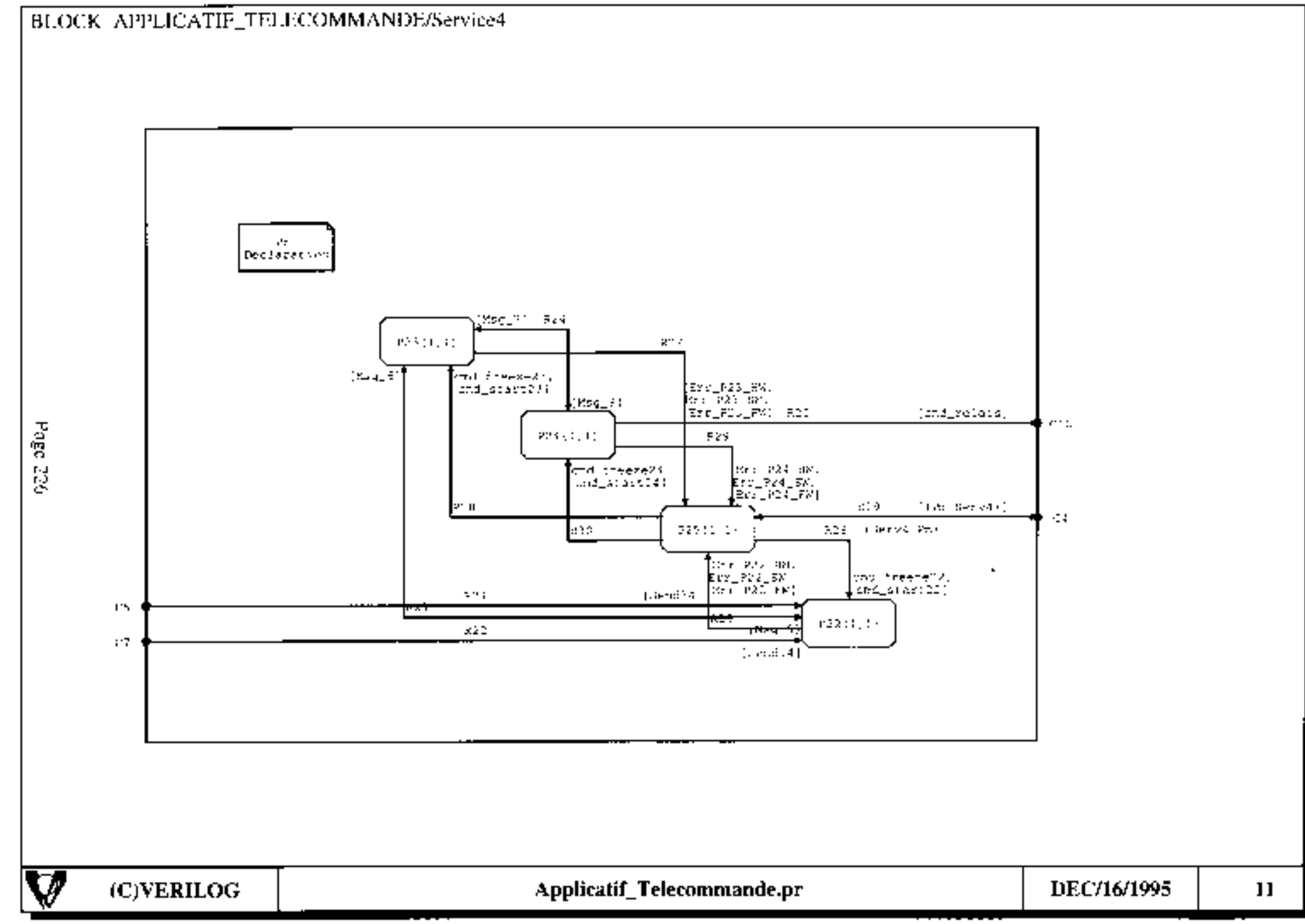
PROCESS APPLICATIF_TELECOMMANDE/Service2/P15



PROCESS APPLICATIF_TELECOMMANDE/Service2/P12

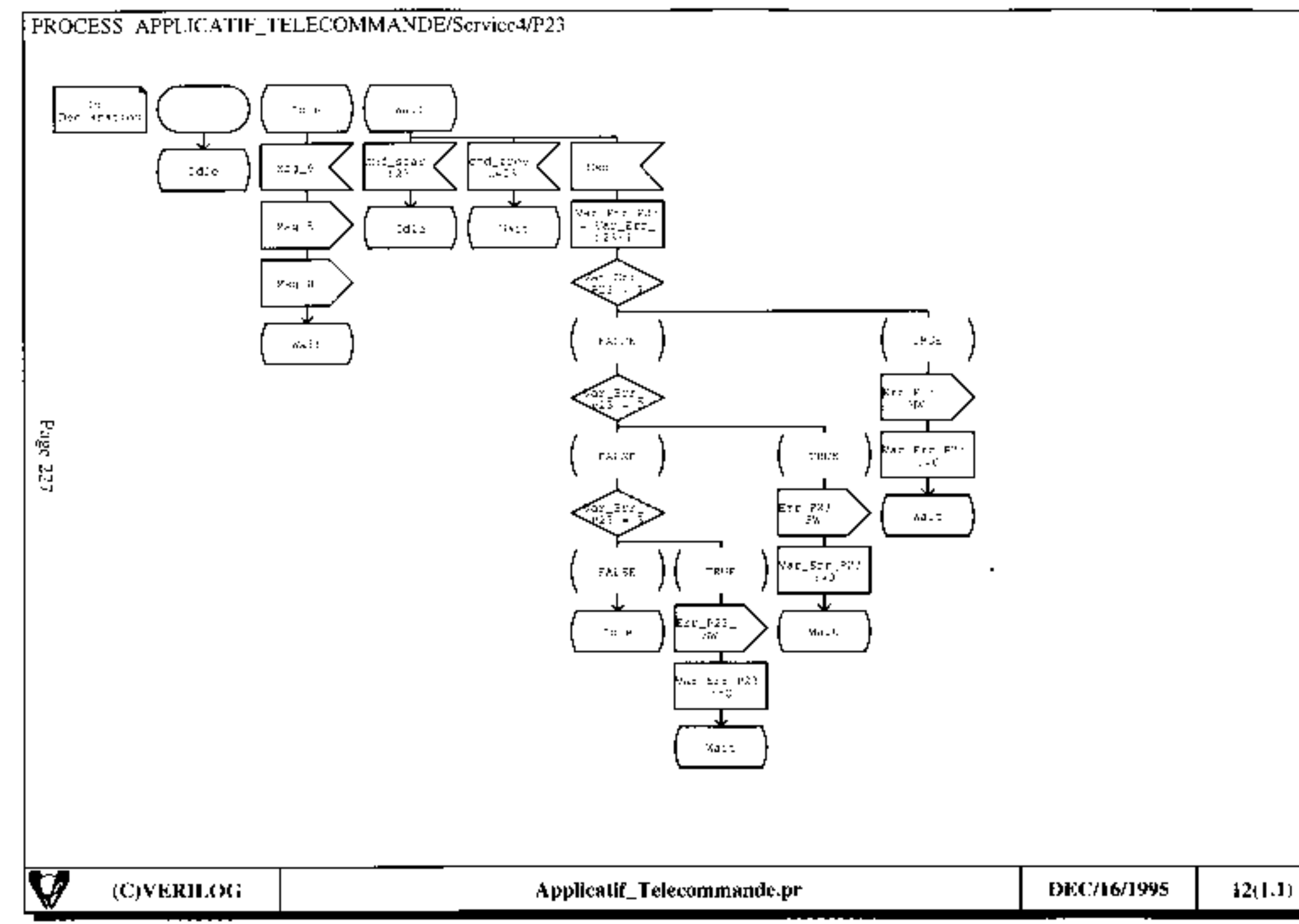


BLOCK APPLICATIF_TELECOMMANDE/Service4

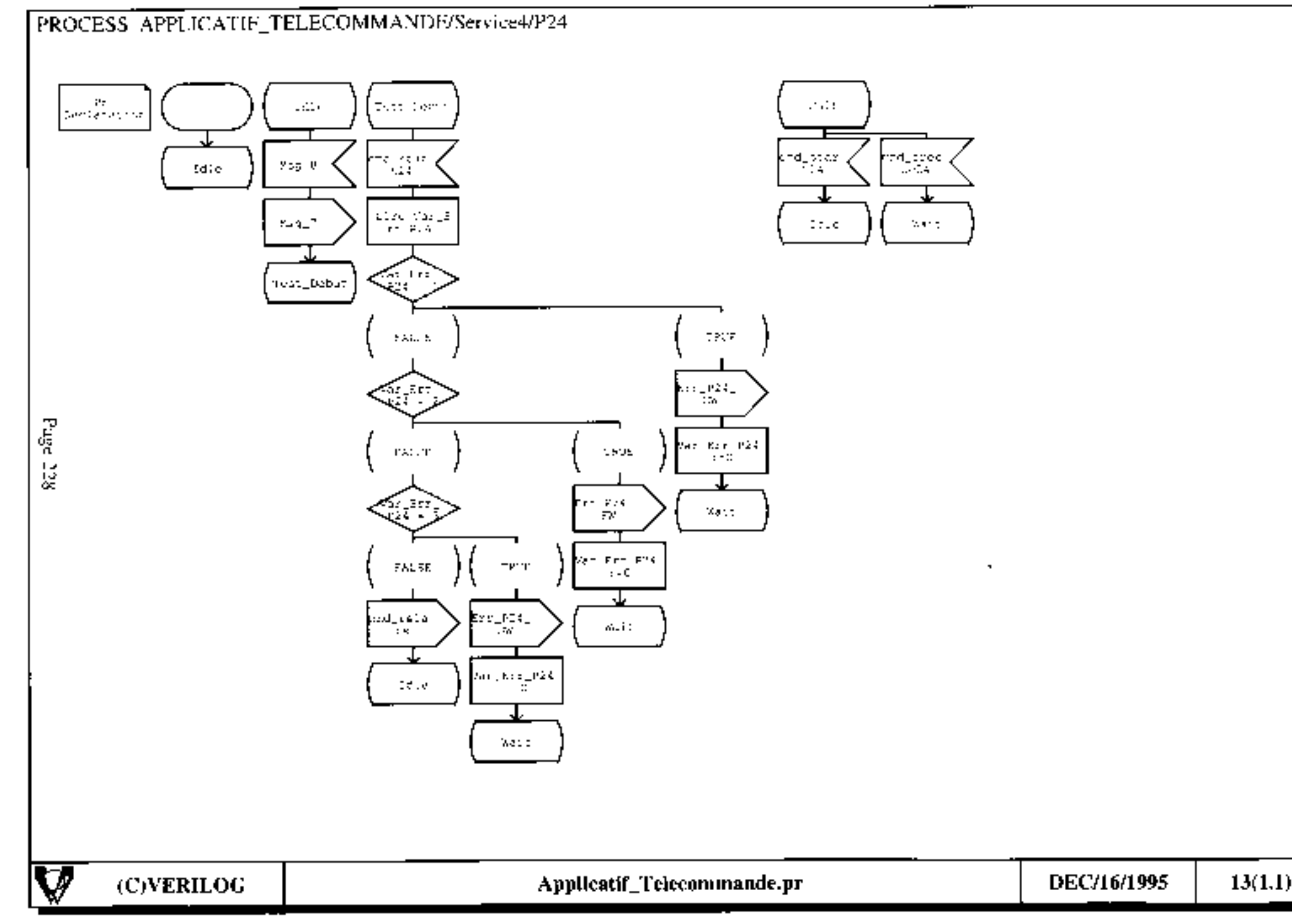


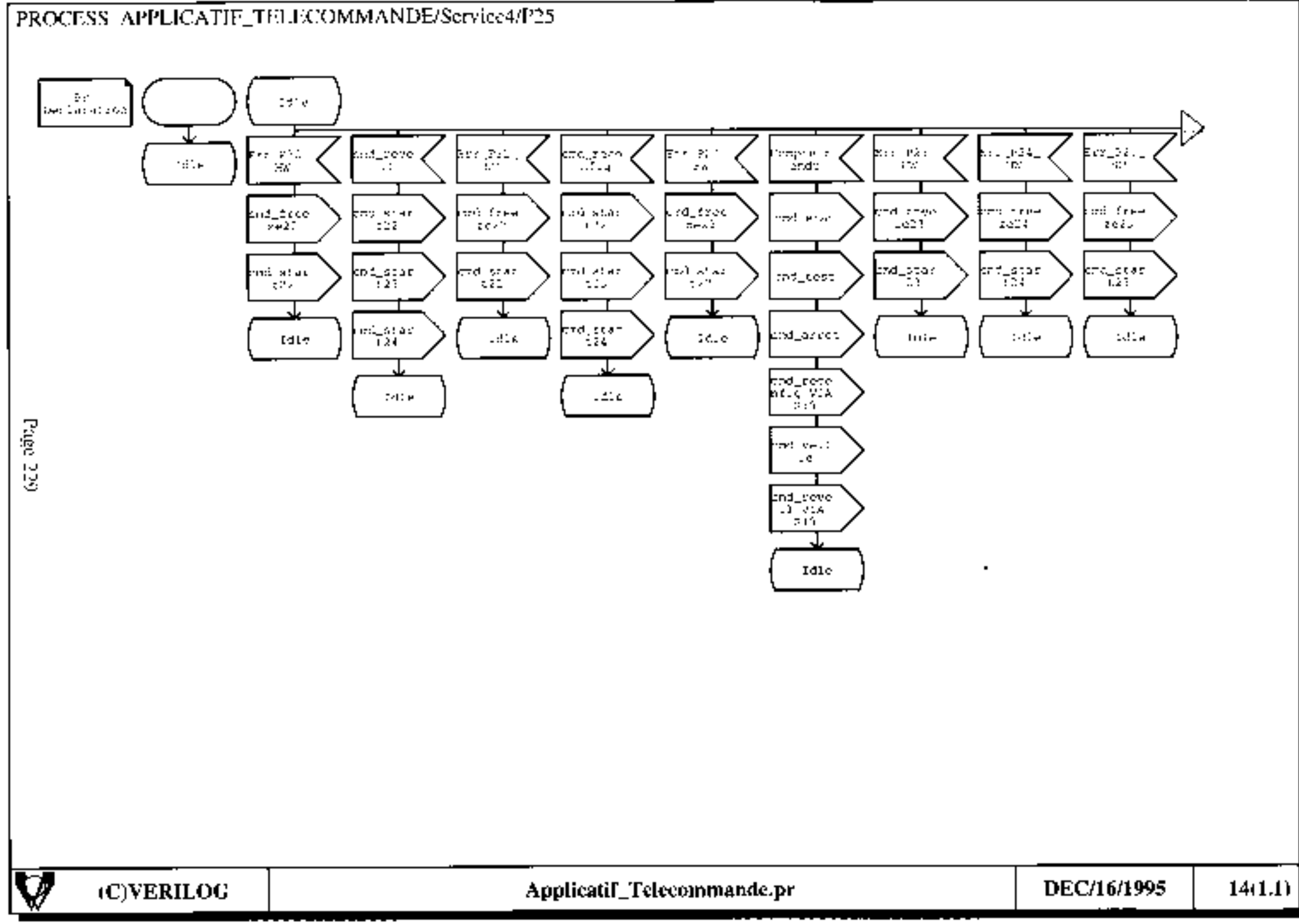
(C)VERILOG	Applicatif_Telecommande.pr	DEC/16/1995	11

PROCESS APPLICATIF_TELECOMMANDE/Service4/P23

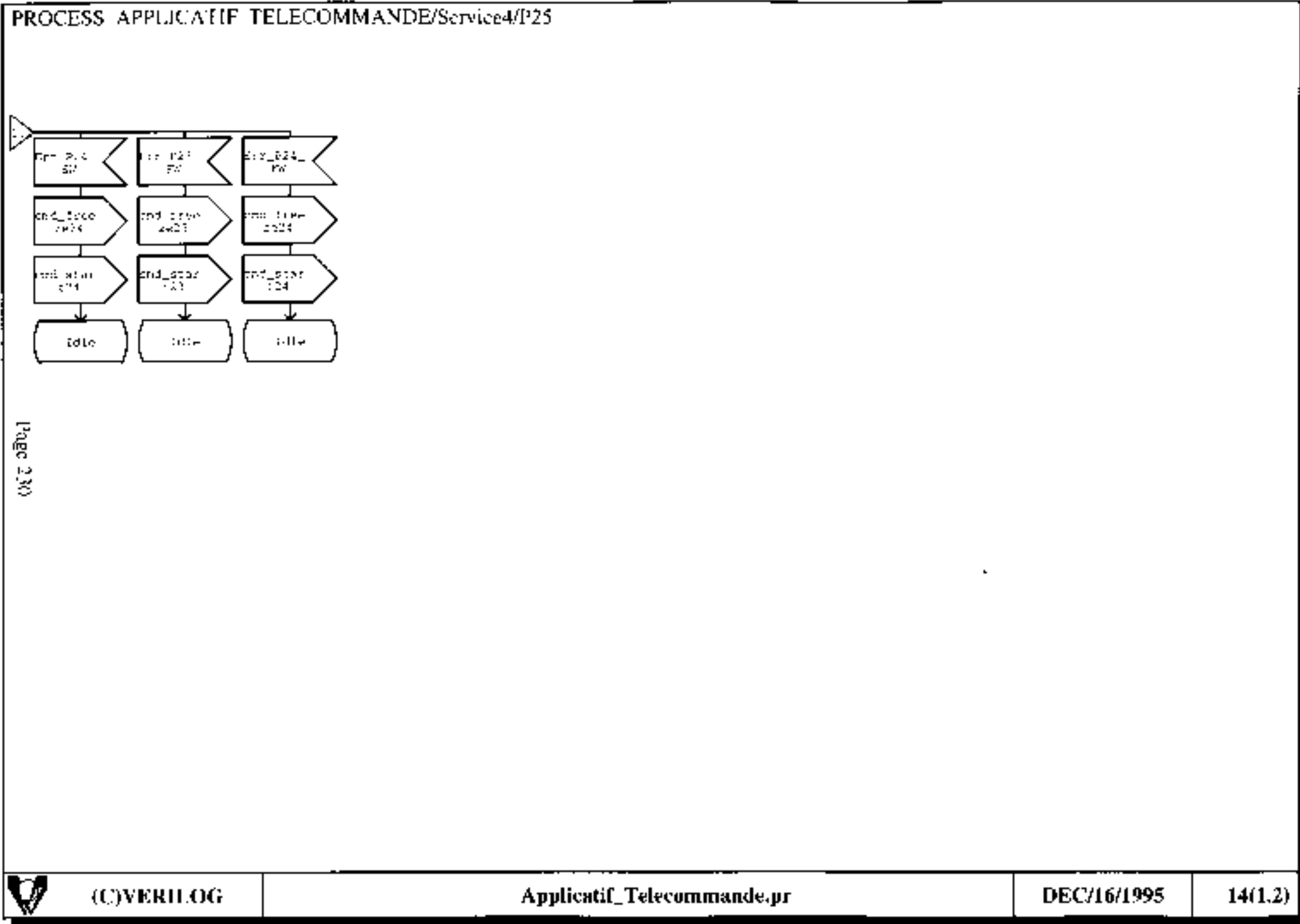


PROCESS APPLICATIF_TELECOMMANDE/Service4/P24

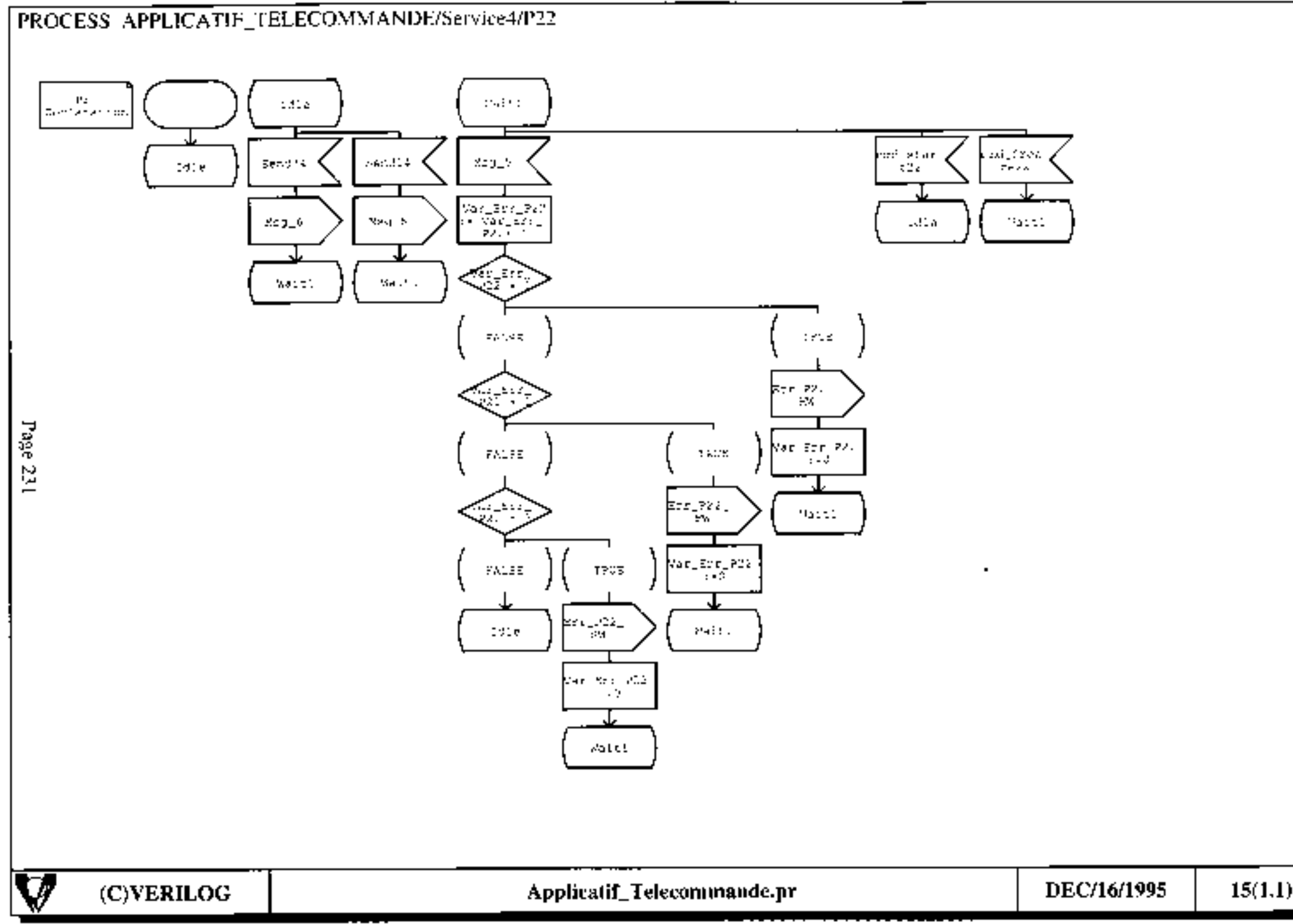



PROCESS APPLICATIF_TELECOMMANDE/Service4/P25
Idle
(C)VERILOG
Applicatif_Telecommande.pr
DEC/16/1995
14(1.1)

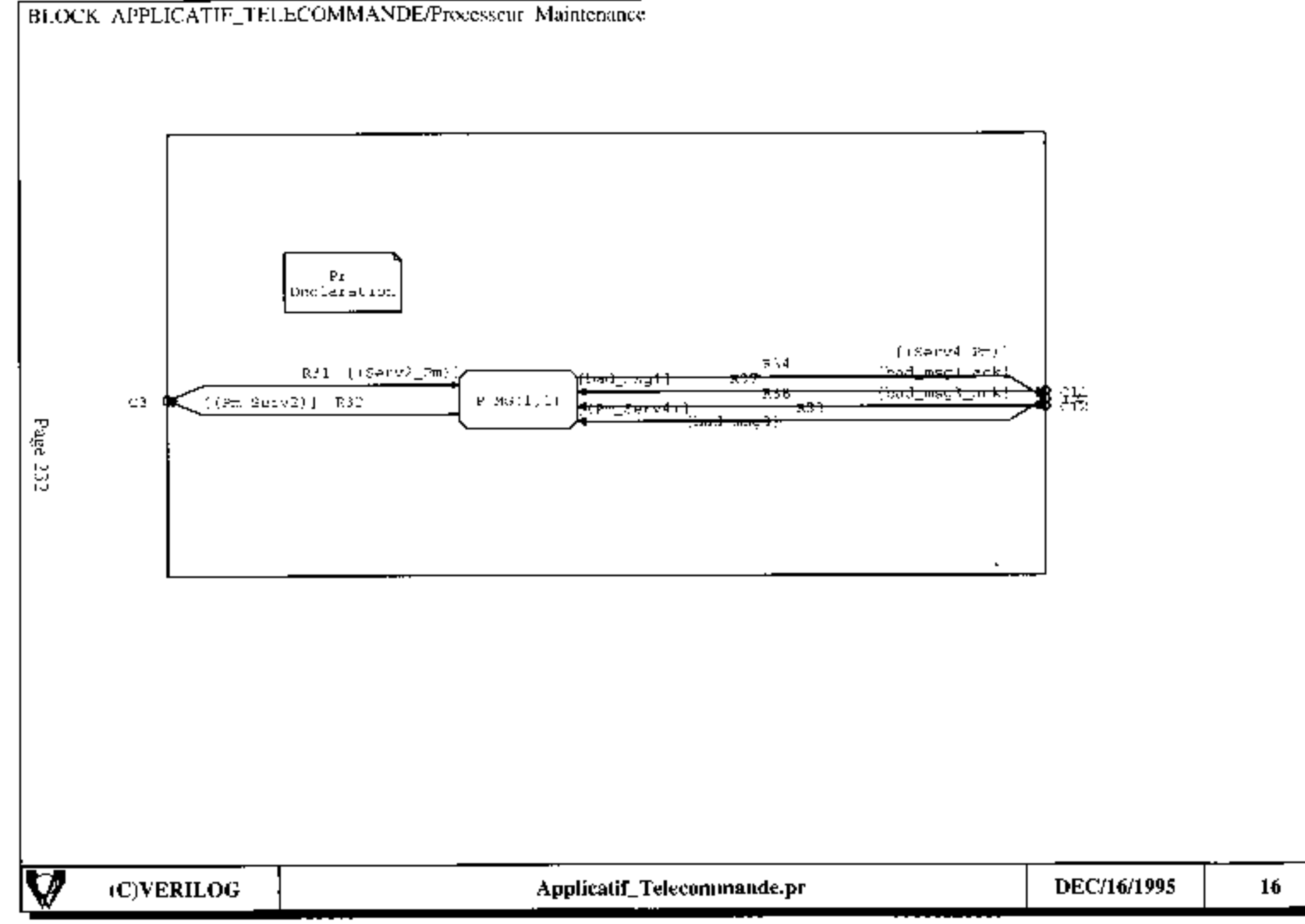
PROCESS APPLICATIF TELECOMMANDE/Service4/P25

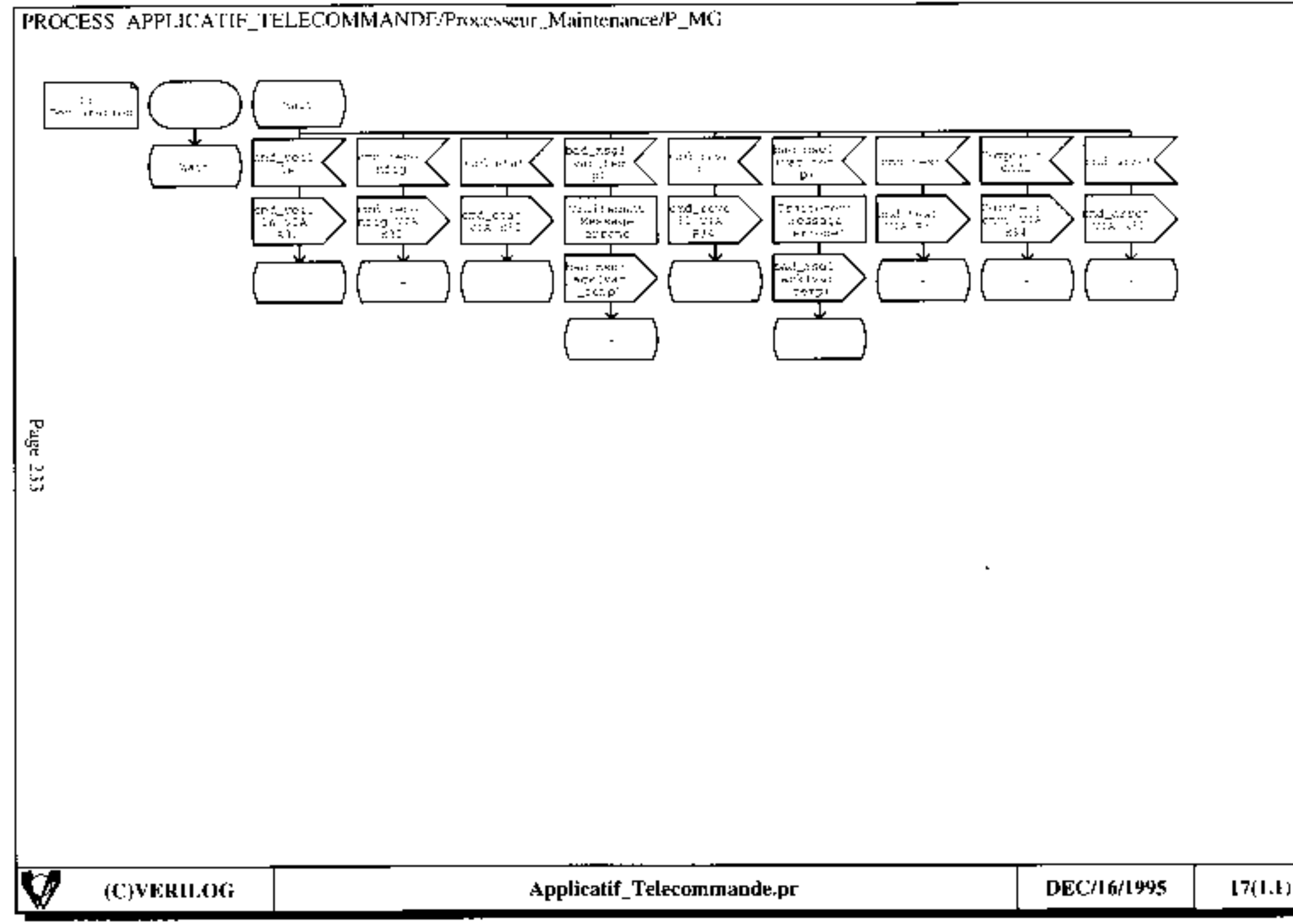


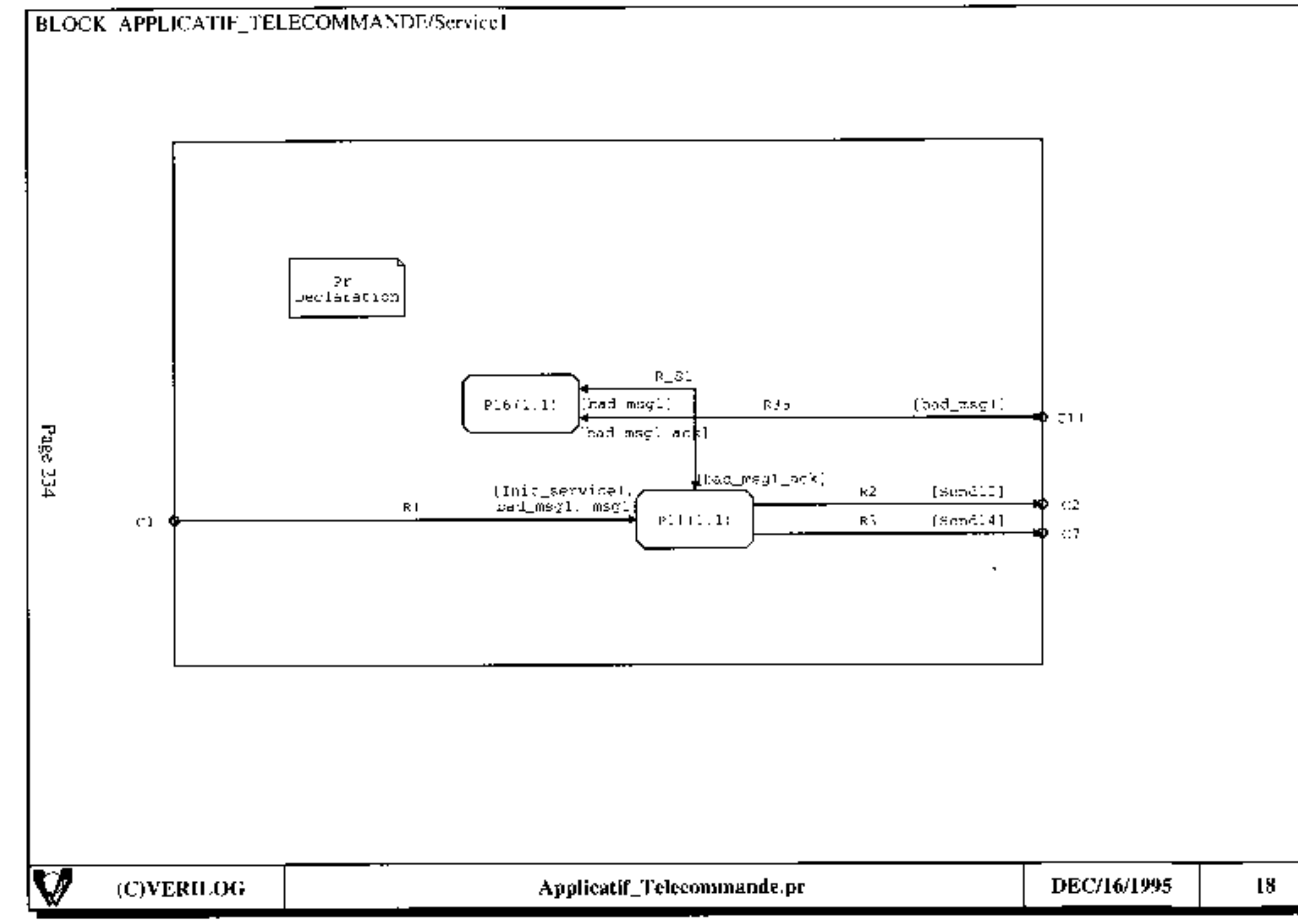
PROCESS APPLICATIF_TELECOMMANDE/Service4/P22



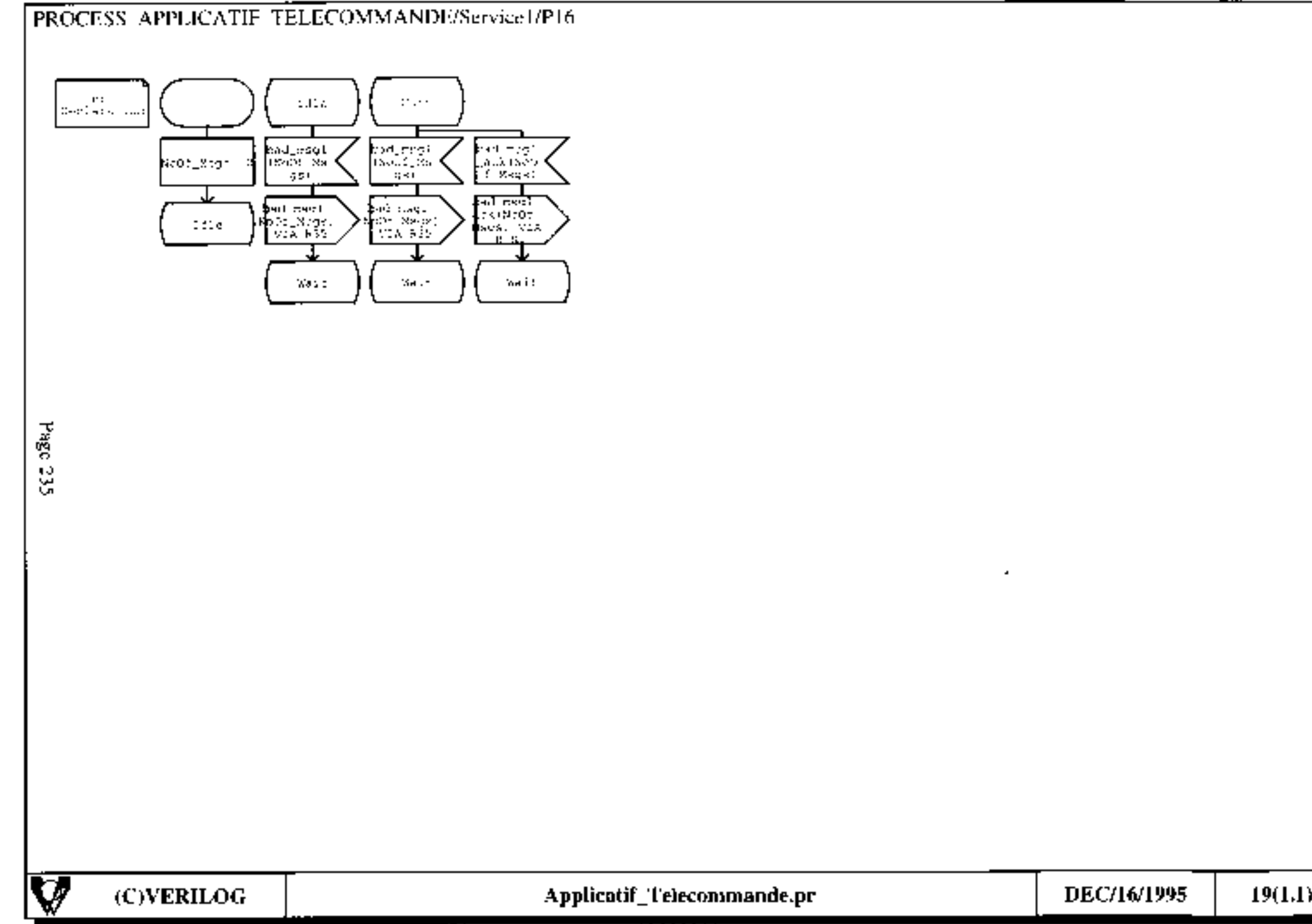
BLOCK APPLICATIF_TELECOMMANDE/Processeur Maintenance



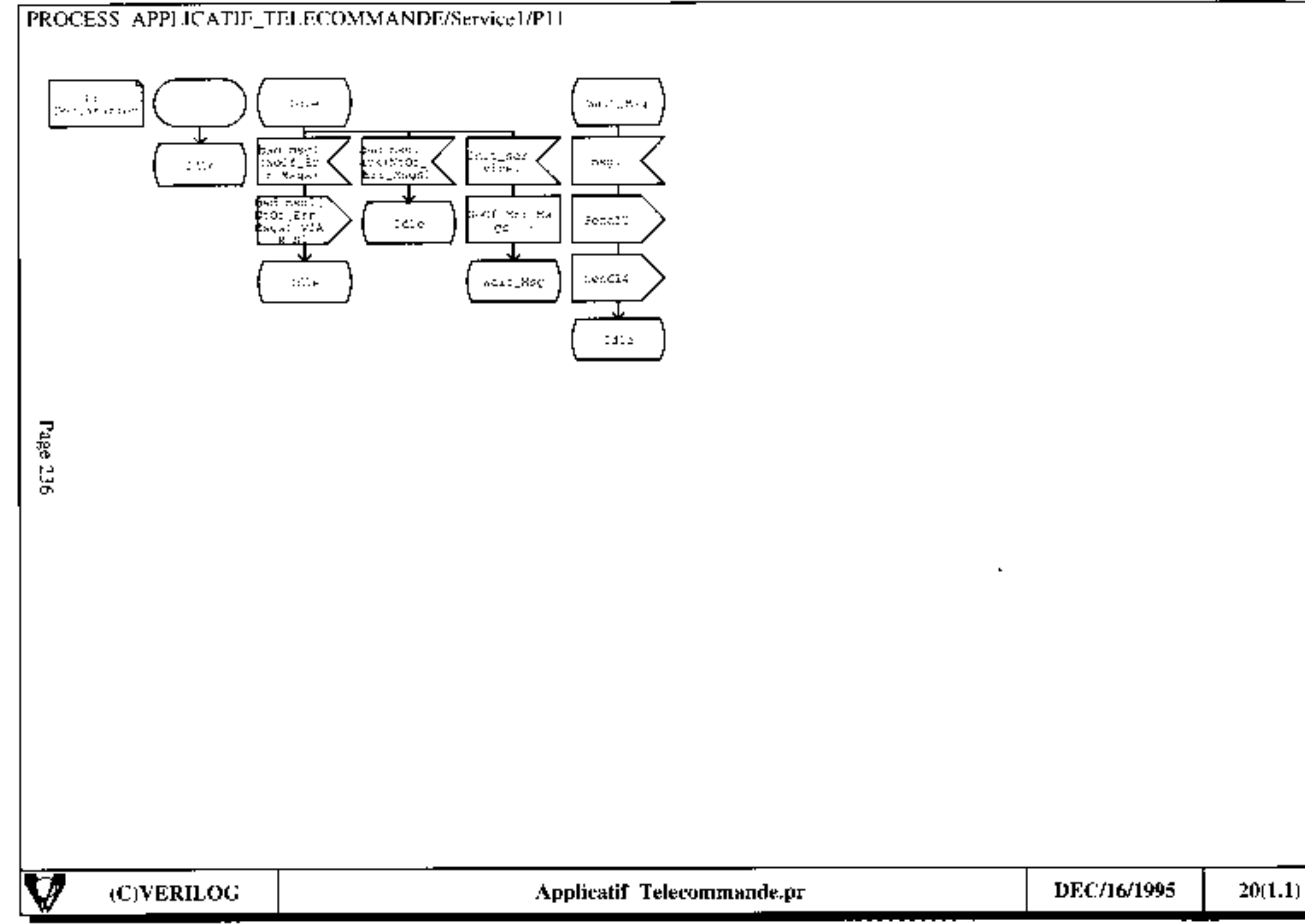

PROCESS APPLICATIF_TELECOMMANDE/Processeur_Maintenance/P_MG
(C)VERILOG
Applicatif_Telecommande.pr
DEC/16/1995
17(1.1)


BLOCK APPLICATIF_TELECOMMANDE/Service1
Pr
Declaration
R_S1
[bad_msg1]
[bad_msg1_ack]
[Init_service1,
bad_msg1, msg1]
R1
R2
R3
(C)VERILOG
Applicatif_Telecommande.pr
DEC/16/1995
18

PROCESS APPLICATIF TELECOMMANDE/Service1/P16



PROCESS APPLICATIF_TELECOMMANDE/Service1/P11



D.1. Système émetteur/récepteur

Ce système est composé de deux blocs; le premier, appelé “send_wait”, est composé des deux processus émetteur et récepteur; le deuxième, appelé “environement”, modélise l'environnement. Ce dernier bloc contient un processus, appelé “counter”, qui incrémente une variable à chaque fois que le processus récepteur reçoit dix messages.

Dans ce qui suit les notations utilisées dans SDL graphique sont décrites.

Déclarations

Bloc

Processus

État initial

État

Retour à l'état précédent

Émission

Réception

C.1. Introduction

L'objet de cet annexe est de présenter les algorithmes des différentes primitives de découpage et de synthèse de la communication qui ont été présentées dans les chapitres 5 et 6. Tout d'abord, les notations qui seront utilisées dans ces algorithmes seront proposées.

En Solar, un état (State) est défini par les attributs suivants :

Paraction (resp. *Seqaction*) est un constructeur qui définit des actions parallèles (resp. séquentielle). Le constructeur *Simpleaction* désigne une instruction simple tel qu'une affectation ou bien une instruction conditionnelle.

Notation : parent(A, B) est la machine qui contient les deux machines A et B.

A précède B dans parent(A, B) si l'entrée par défaut de parent(A, B) est A.

Le signe '+' sera utilisé pour représenter l'opérateur de concaténation et 'U' dénote l'opérateur d'union.

{ N_X est le nom de la machine X }

{ G_X est l'exception de la machine X }

{ S_X est la liste des états dans la machine X }

{ E_X est la liste des états d'entrée de la machine X }

{ D_X est l'état suivant par défaut dans la machine X }

{ R_X est l'état de réinitialisation de la machine X }

{ V_X est la liste des variables de la machine X }

{ $Block_X$ est la liste des états et tables d'états de la machine X }

C.2. Primitive Move

Algorithme :

Move (P, Q)
début
cas de chemin entre P et Q
Parallèle : $Par_Q := Par_Q \cup \{ P \}$
Enlever la machine P de $Par_{parent(P)}$
Séquentiel : $S_Q := S_Q \cup \{ N_P \}$
si $(E_Q = N_{parent(P)})$ et $(E_{parent(P)} = N_P)$
alors $E_Q := \{ P \} \cup E_Q$
fsi
$V_Q := V_Q \cup V_{parent(P)}$
$V_{parent(P)} := \emptyset$
$Block_Q := Block_Q \cup \{ P \}$
Enlever N_P de $E_{parent(P)}$
Enlever N_P de $S_{parent(P)}$
Enlever la machine P de $Block_{parent(P)}$
fcas
si P *table d'états* alors
{ héritage par P de *l'exception* de parent(P)}
$G_P := G_P \cup G_{parent(P)}$
fsi
fin

C.3. Primitive Merge

Algorithme :

Merge (A, B)
début
cas de B
table d'états : $N_{AB} := N_A + N_B$
$G_{AB} := G_A \cup G_B$
si A précède B dans parent(A,B) alors
$D_{AB} := D_A$
$E_{AB} := E_A \cup E_B$
sinon
$D_{AB} := D_B$
$E_{AB} := E_B \cup E_A$
fsi
$R_{AB} := R_A$
$S_{AB} := S_A \cup S_B$
$V_{AB} := V_A \cup V_B$
$Block_{AB} := Block_A \cup Block_B$
état : $N_{AB} := N_A$
$G_{AB} := G_A$
$D_{AB} := D_A$
si B précède A dans parent(A,B) alors
$E_{AB} := \{ B \} \cup E_A$
sinon
$E_{AB} := E_A$
fsi
$R_{AB} := R_A$
$S_{AB} := S_A \cup \{ N_B \}$
$V_{AB} := V_A$
$Block_{AB} := Block_A \cup \{ B \}$
fcas
fin

C.4. Primitive Split

La primitive *Split*. possède un seul paramètre qui représente une MEF étendue. On fait l'hypothèse que la machine à découper est séquentielle.

{ X : représente la machine à découper }
{ n : représente le nombre de machines dans la machine X }
{ X_i : représente l'une des n machines de X }
{ ctrl_X_i : représente le signal de contrôle créé pour contrôler la machine X_i}

Algorithme :

```
Split (X)
début
  pour i = 1 à  n  faire
        V_Parent(X) := V_Parent(X)  U  { ctrl_Xi }
  fpour
  pour i = 1 à  n  faire
        cas de Xi
              séquentielle : ActionSeq (Xi)
              parallèle    : ActionPar (Xi)
        fcas
  fpour
fin
```

Dans la première boucle les signaux de contrôle sont rajoutés dans la machine, notée Parent(X) et qui contient la machine à découper X. Les n sous-arbres de la machine hiérarchique X sont représentés par des machines X_i ($i \in [1..n]$). Dans la seconde boucle, chaque sous-arbre de X est traité, selon qu'il s'agit d'une machine séquentielle ou d'une machine parallèle. Les fonctions ActionSeq et ActionPar ont chacune un paramètre. Il représente la machine à traiter qui est séquentielle pour la fonction ActionSeq et parallèle pour la fonction ActionPar. Chaque fonction introduit un état de repos (*Idle*) et transforme les transitions externes en affectations de signaux de contrôle. Par exemple, si dans la machine X_i, on a une transition vers la machine X_j, on transforme cette transition en la séquence suivante : {ctrl_X_i := 0, ctrl_X_j := 1, état suivant := *Idle*}.

C.5. Primitive Cut

Soit M une machine parallèle sur laquelle la primitive *Cut* va être appliquée. On suppose que cette machine contient n machines (sous-arbres de M) M_i ($i \in [1..n]$) qui sont composées en parallèle. L'algorithme de la primitive *Cut* peut être défini comme suit:

Algorithme :

```
Cut ( M )
début
  pour chaque machine Mi dans M  faire
        Créer un bloc UCi (unité de conception)
        Mettre Mi dans UCi
        pour chaque Port qui est partagé et utilisé dans  Mi faire
              Créer un port dans Interface de UCi
        fpour
        pour chaque Constante qui est partagée et utilisée dans  Mi faire
              Déclarer la constante dans Mi
        fpour
        pour chaque Variable V qui est partagée et utilisée dans  Mi faire
              Ajouter dans la spécification, un canal relatif à V
              Ajouter dans Interface de UCi un accès au canal de V
              pour chaque utilisation de V faire
                    si  accès en lecture alors
                          appeler la méthode Read du canal
                    sinon
                          appeler la méthode Write du canal
                    fsi
              fpour
        fpour
  fpour
fin
```

C.6. Primitive Map

Soit un ensemble de sous-systèmes communicants sur lesquels la primitive *Map* va être appliquée.

Algorithme :

<u>*Map*</u>
<u>début</u>
  <u>pour</u> chaque bloc UC (Unité de Conception comportementale)
    <u>pour</u> chaque Accès dans l'interface
      Identifier le canal correspondant
      <u>pour</u> chaque appel à ce canal
        Déclarer la méthode comme procédure au niveau de l'UC
        Insérer les ports qu'elle utilise dans l'interface de l'UC
        Transformer l'appel à la méthode en un appel de procédure
      <u>fpour</u>
      Enlever l'accès de l'interface de l'UC
      Enlever toutes les méthodes du canal
      Déclarer le canal comme une UC
    <u>fpour</u>
  <u>fpour</u>
<u>fin</u>

B.1. Introduction

La présente étude propose une évaluation des principales méthodologies de conception de systèmes mixtes logiciels/matériels [BIsm94b]. Différentes approches sont actuellement explorées, dans des laboratoires de recherches, pour la conception de systèmes mixtes. Dans ce qui suit, les principales méthodologies pour ce domaine seront parcourues. L'accent est essentiellement mis sur les points suivants :

- **Le niveau système** : Une description brève du contexte système et du style de la description en entrée.

- **Le type d'application** : Les types d'applications envisagées au niveau système.

- **L'approche de conception** : Une présentation des étapes suivies pour la conception mixte.

- **L'architecture cible** : Une vue générale de la plate-forme d'implantation.

- **Notes** : Autres informations importantes relatives à la méthodologie.

- **Références** : Références bibliographiques adoptées.

B.2. Méthodologie de l'université de Californie/Berkeley: (Ptolemy)

Niveau système : Au niveau système, l'environnement utilisé est appelé Ptolemy. Les contraintes de conception pour un système peuvent inclure : Les contraintes temps-réel, performances requises, vitesse, surface, taille du code, consommation de puissance et la programmabilité.

Types d'applications : Applications de traitement du signal [Hilf85, PaLi94] (DSP : *Digital Signal Processing*) et systèmes communicants. Le logiciel est formé par un ou plusieurs programmes, souvent assez complexes, s'exécutant sur des composants programmables.

Approche de conception : L'approche est formée par une étape de découpage matériel/logiciel, une étape de synthèse de logiciel, une étape de synthèse du matériel et

une étape de synthèse d'interfaces matériel/logiciel. Une dernière étape concerne la simulation du système hétérogène.

Le découpage est fait manuellement. Il est guidé par les contraintes de surface, de vitesse et de flexibilité. Lors de la synthèse du logiciel pour une architecture multiprocesseurs, le découpage essai d'optimiser des fonctions coûts telles que : coût des communications, espaces mémoires locales et globales, etc. Des travaux récents [KaLe94b] parlent de découpage automatique.

La synthèse du logiciel consiste en une génération de code pour des processeurs programmables. Le code obtenu pourrait être du C ou de l'assembleur selon le processeur cible. Du côté synthèse du matériel, la description en entrée est un code SILAGE. La synthèse d'interfaces inclut :

- L'ajout de registres, files d'attentes FIFOs, et décodeurs d'adresses dans le matériel, et

- L'insertion du code pour les opérations d'entrée/sortie et les sémaphores de synchronisation dans le logiciel.

La simulation du système hétérogène a lieu une fois que la synthèse du logiciel et du matériel sont faites. Les résultats de la simulation permettent de vérifier si la conception est conforme à la spécification initiale. Cependant, à l'issue des étapes de synthèse du matériel et du logiciel, quelques estimations de performances sont faites afin de valider la conception. Les estimations concernent la surface, le chemin critique et l'utilisation de bus et de composants.

Architecture cible : Le logiciel peut être synthétisé pour une variété de processeurs cibles. Les processeurs cibles peuvent avoir une configuration parmi plusieurs :

1. Système mono-processeur,

2. Architecture parallèle à mémoire partagée,

3. Architecture parallèle avec un bus partagé,

4. Architecture avec passage de messages.

Dans [SrBr91], une implémentation basée sur une plate-forme d'architecture paramétrable est discutée. On utilise une bibliothèque de modules matériels et logiciels paramétrables.

Notes : Utilisation d'une représentation unifiée entre le matériel et le logiciel. Cette représentation facilite l'émigration de fonctions entre les implémentations.

Références : [KaLe93, KaLe94a, KaLe94b, SrBr91, Sriv92, SrBr93, ChGi93]

B.3. Méthodologie de l'université de Californie/Stanford: (VULCAN)

Niveau système : La description initiale est une description fonctionnelle donnée dans le langage HardwareC. L'approche de synthèse mixte utilise d'autres langages de description de matériel tels que VHDL et Verilog. Les descriptions HardwareC consistent en un ensemble de processus interagissants qui sont instanciés dans des blocs. Un processus s'exécute en parallèle avec d'autres processus de la spécification et ce ré-initialise automatiquement à sa terminaison. Il s'agit d'une extension au langage C afin d'inclure le parallélisme et des aspects pour la description de matériel.

Types d'applications : Systèmes temps-réel.

Approche de conception : Utilisation de contraintes temporelles. Ces contraintes sont de deux types :

1. Contraintes de délai Minimum/Maximum,

2. Contraintes de temps d'exécution.

L'estimation des délais d'opérations est basée sur la réalisation envisagée (type du matériel à utiliser et du processeur pour le logiciel). Ainsi, ce type de synthèse est orienté vers l'architecture. Les étapes suivantes sont suivies dans cette approche :

1. Prise en compte (saisie) du modèle,

2. Partitionnement matériel/logiciel,

3. Synthèse du modèle découpé en composants matériels et logiciels interconnectés dans une architecture cible.

L'algorithme de découpage commence avec une partition initiale où toutes les opérations, exceptées celles à délai non limité, sont affectées au matériel. La partition est raffinée par la migration d'opérations du matériel vers le logiciel afin d'obtenir une partition à moindre coût.

L'approche utilise un ensemble de modèles de graphes de séquencement avec des contraintes temporelles entre les opérations. Un ensemble de modèles de graphes de séquencement est réalisé en logiciel et un autre est réalisé en matériel. Le logiciel est composé par plusieurs programmes. Un ordonnancement est effectué sur le modèle du graphe relatif au logiciel. La concurrence entre les processus est effectuée à travers un modèle d'exécution supportant l'entrelacement.

L'interface matériel/logiciel consiste en des files de données et un contrôle qui maintient les identificateurs pour les processus activés dans l'ordre de l'arrivée de leurs données.

Architecture cible : L'architecture cible utilise un seul processeur qui est intégré avec un seul circuit intégré (ASIC), une mémoire et un bus. Le processeur utilise un seul niveau de mémoire et d'espace d'adressage pour les instructions et les données.

Notes : Utilisation des routines d'interruption pour les transferts de données d'un processus vers un ASIC. Ces interruptions sont associées à un temporisateur.

Références : [GuDM93, GuDM92, GuC92a, GuC92b, GuCo94]

B.4. Méthodologie de l'université de Californie/Irvine: (SpecSyn)

Niveau système : La spécification de niveau système qui est en entrée est exprimée en SpecCharts. Les composants matériels générés sont exprimés en VHDL. Le logiciel généré est décrit en langage C.

Types d'applications : Pas de restrictions sur le type des systèmes traités.

Approche de conception : La conception au niveau système est divisée en trois phases distinctes :

1. Prise en compte de la spécification,

2. Raffinement des spécifications,

3. Réalisation de la conception.

La phase de raffinement joue un rôle important dans le processus de conception. Elle inclut les tâches suivantes :

1. Groupement d'objets : Par exemple, les variables sont groupées pour former une unité de stockage qui pourrait être implantée par la suite comme une mémoire.

2. Association des groupes d'objets aux composants disponibles dans une bibliothèque.

3. Raffinement : il inclut les sous-tâches (1) Synthèse d'interfaces, (2) Insertion d'arbitrage pour les canaux de communication, (3) Fusion de protocoles et (4) Transposition des variables en adresses mémoire.

Architecture cible : L'architecture cible est une machine mono-processeur. Les processeurs à mémoire cache et/ou avec instructions pipelinées ne sont pas pris en considération.

Notes : L'évaluation des performances du système à développer est faite par les deux techniques de simulation et d'estimation. Un estimateur de logiciel permet de fournir des métriques à propos du logiciel généré en fonction du processeur cible. Ces métriques sont : Le temps d'exécution, la taille de l'exécutable et celle des données. Cette approche est basée sur l'identification des goulots d'étranglement dans le système. Il serait ainsi possible de repérer des fonctions où l'utilisation d'un circuit spécifique peut améliorer les performances.

Références : [GaVa95, GaVa94, GoGa94]

B.5. Méthodologie de l'université de Carnegie Mellon

Niveau système : Il s'agit d'une spécification fonctionnelle qui concerne aussi bien le logiciel que le matériel. Le modèle utilisé est celui des processus séquentiels et communiquants (CSP) où chaque processus est décrit d'une façon comportementale dans un langage de description de haut niveau. La caractéristique principale de ce modèle est le niveau d'abstraction dans lequel il représente l'interaction entre le matériel et le logiciel.

Types d'applications : Une classe de systèmes mixtes matériel/logiciel où l'on peut envisager l'accélération d'une fonction logicielle par l'extraction de certaines parties pour les réaliser en matériel. Les systèmes de contrôle complexes sont considérés dans cette approche.

Approche de conception : La méthodologie utilise les étapes suivantes :

1. Partitionnement matériel/logiciel,

2. Synthèse comportementale,

3. Compilation du logiciel,

4. Démonstration sur un banc de test composé par des CPUs, FPGAs et des interconnexions programmables.

La conception mixte permet de réaliser les tâches suivantes :

1. Caractérisation des performances du matériel et du logiciel,

2. Identification des partitions matériel/logiciel,

3. Transformation d'une description fonctionnelle en une partition, et

4. Synthèse du matériel et du logiciel résultants.

La simulation mixte matériel/logiciel est le moyen utilisé pour vérifier la fonctionnalité des descriptions matérielles/logicielles. Les processus à réaliser en matériel sont décrits en Verilog, le simulateur Verilog est utilisé pour les simulations comportementales. Les processus modélisant le logiciel s'exécutent comme des processus sous le système d'exploitation UNIX et communiquent avec le simulateur via des points de communication (les sockets). Les sockets sont utilisés ici pour permettre des simulations distribuées sur un réseau. La communication inter-processus est possible à travers le partage d'un espace mémoire commun.

Architecture cible : Utilisation d'un banc de test composé par des FPGAs et des interconnexions existantes dans la carte mère d'un ordinateur à usage général. Les composants pour applications spécifiques (ASICs) sont conçus pour coopérer avec le logiciel qui s'exécute sur un seul CPU. Ils sont connectés au système entier à travers des

bus de connexion. Le prototypage est fait sur un PC-AT avec une carte composée de FPGAs.

Notes : Le but de cette approche est de cacher les détails de l'architecture matérielle, être explicite concernant le niveau de concurrence et le partitionnement matériel/logiciel, et de faciliter le transformation du niveau de concurrence et des partitions.

Références : [AdSc93, ThAd93]

B.6. Méthodologie de Siemens: (CODES)

Niveau système : La spécification en entrée peut être donnée en langage SDL ou en StateCharts. La sortie générée est composée de codes C et VHDL. StateCharts utilise une communication par diffusion (*broadcasting*), alors que SDL utilise une communication point-à-point. L'utilisation de ces deux langages permet d'établir des modèles hiérarchiques et fonctionnels du système à développer. SDL est plutôt efficace pour la description des protocoles de communications alors que StateCharts convient pour la spécification des fonctions dominées par le contrôle.

Types d'applications : Les systèmes de communication. Un modèle abstrait pour les machines à accès aléatoire (PRAMs : *Parallel Random Access Machines*) est adopté. Ce modèle supporte la communication point-a-point et celle par diffusion. Un système est représenté à travers des unités communicantes, chacune d'elles représente un comportement.

Approche de conception : Un environnemnt de conception de systèmes et appelé CODES a été développé. Les différentes phases de CODES sont les suivantes :

1. Modélisation,
2. Conception de composants,
3. Simulation,
4. Integration dans un prototype.

Architecture cible : La plate-forme qui supporte le logiciel est formée par un processeur, une mémoire, des composants standards et des circuits spécifiques issus de la synthèse.

Notes : L'environnement CODES offre la possibilité de synthèse d'un contrôleur d'interruption ce qui permet la conception de systèmes multiprocesseurs.

Références : [BuVe92, BuSe93, Buch94]

B.7. Méthodologie de l'université de Tübingen

Niveau système : Le langage de spécification utilisé s'appelle UNITY. Dans ce langage, uniquement les comportements synchrones et asynchrones d'un système sont spécifiés.

Types d'applications : Systèmes synchrones et asynchrones (selon les dépendances de données).

Approche de conception : Un découpage de la conception est effectué. Ce découpage vise essentiellement à maximiser le parallélisme entre les composants, supporter la réutilisation des structures matérielles, et minimiser les coûts de communication et l'utilisation des ressources. La méthode de vérification est basée sur la simulation. Les composants matériels sont spécifiés en VHDL, alors que les composants logiciels sont spécifiés en langage C. Les interfaces matériel/logiciel peuvent être décrites en C, VHDL, ou mixtes C/VHDL.

Architecture cible : Pour des raisons de simplification, une architecture de réalisation prédéfinie est utilisée. L'architecture est formée par un noyau RISC (*Reduced Instruction Set Computer*), une mémoire principale, un ensemble de circuits spécifiques et un circuit d'interface pour le contrôle de la communication entre le processeur RISC et les ASICs. Il s'agit ici d'une architecture du type maître/esclave où le RISC contrôle l'activation de chaque ASIC.

Références : [BaRo94, BaRo92]

B.8. Méthodologie de l'université de Cincinnati: (RAPID)

Niveau système : La description en entrée combine le texte et le graphique. Un environnement de conception système appelé RAPID est développé et utilisé. Il est intégré d'une part avec le langage de description de matériel VHDL et d'autre part avec le langage de programmation ADA.

Types d'applications : Pas de restriction sur les types d'applications. Les auteurs illustrent dans [ReWi93] l'outil RAPID avec un système de contrôle numérique.

Approche de conception : Dans cette approche, le concepteur essaye différentes alternatives à partir d'une réalisation purement logicielle jusqu'à une réalisation purement matérielle. Les résultats de temps (*timing*) obtenus après simulation aident le concepteur à décider quelles composantes logicielles sont à réaliser en matériel.

Architecture cible : Aucune architecture cible n'est adoptée. Cependant un processeur intégré est utilisé pour l'exécution du logiciel.

Notes : La technique de simulation est utilisée pour tester les performances de la conception. La simulation est réalisée par un simulateur VHDL. Tous les modules logiciels doivent être écrits en langage assembleur.

Références : [ReWi93]

B.9. Méthodologie de l'université de Linkoping

Niveau système : La spécification acceptée en entrée est une description comportementale. Elle est exprimée dans un langage proche du Pascal appelé ADDL (*Algorithm Design Description Language*). Ce langage est un sous-ensemble de Pascal avec certaines extensions pour supporter le parallélisme et une sémantique matérielle. La spécification en entrée peut être aussi donnée en VHDL.

Types d'applications : Toute les spécifications comportementales.

Approche de conception : Le processus de partitionnement est formulé comme un problème de partitionnement de graphes. L'approche réalise la transposition d'un graphe de flux de contrôle et de données sur un graphe unique de partitionnement. Différents types d'arcs sont utilisés entre les noeuds pour refléter les dépendances. Le

partitionnement divise la fonctionnalité d'un système en un ensemble de modules, chacun correspondant à une unité physique (puce) ou à un paquetage logiciel. Une fonction coût guide le processus de partitionnement. Elle tient compte des coûts de communication et de synchronisation.

Cette approche utilise les résultats de simulation pour identifier les parties critiques du système. Ces parties critiques sont alors réalisées en matériel.

Architecture cible : Les résultats de partitionnement sont transposés sur un ensemble de représentations d'une conception avec une interface bien définie. Aucune architecture cible n'est adoptée.

Notes : La méthode est basée sur un modèle de réseaux de Petri.

Références : [PeKu93]

B.10. Méthodologie de l'université de Manchester

Niveau système : La description en entrée est un programme en langage C avec quelques restrictions pour permettre la traduction vers le langage HardwareC.

Types d'applications : Systèmes complexes.

Approche de conception : Utilisation d'un outil d'analyse de performances à posteriori (*Profiling*). Cet outil de profilage permet d'identifier les parties critiques d'un programme en C.

Le partitionnement matériel/logiciel est effectué sur la base de l'analyse des performances. Les composants logiciels sont formés par du code C et des appels au co-processeur alors que les parties matérielles sont traduites en HardwareC.

Le logiciel est compilé en code machine i960. Le compilateur du matériel produit un réseau logique pour des FPGAs Xilinx de la famille 40XX.

Architecture cible : Un PC-AT avec un banc de développement incluant un processeur, une mémoire, une section d'entrées/sorties (un port série, un FPGA, jusqu'à 15 interruptions externes) et des ASICs.

Notes : Les variables globales ne sont pas permises dans le code source C. Ceci limite les possibilités de communication entre processus.

Références : [EdFo94]

B.11. Méthodologie de ITALTEL (TOSCA)

Niveau système : La description en entrée utilise SpeedChart [Belh94] qui permet de combiner une représentation graphique et textuelle. Le formalisme fourni par SpeedChart appartient à la famille de StateCharts et permet de décrire des représentations textuelle du type VHDL.

Types d'applications : Les circuits (ASICs) dominés par le contrôle tels que les applications de télécommunication, par exemple, les sous-systèmes autocommutateurs numériques.

Approche de conception : Cette approche comporte trois activités étroitement liées. Il s'agit de (1) la restructuration qui réalise des transformations locales à un processus, (2) l'allocation de composants logiciels et matériels, et (3) l'affectation de chaque partition à une réalisation logicielle ou matérielle.

Architecture cible : Une puce comprenant un seul processeur avec un ensemble de coprocesseurs (accélérateurs).

Notes : Le choix de la stratégie adoptée durant le cycle d'exploration de l'espace des solutions est à la charge de l'utilisateur. Ce dernier effectue manuellement les activités de restructuration et d'affectation au logiciel et au matériel.

Références : [AnMa90, AnBa94]

B.12. Méthodologie de l'université de Braunschweig: (COSYMA)

Niveau système : La description en entrée est une spécification textuelle en langage C^X qui est une extension au langage C afin d'inclure des contraintes de temps et le parallélisme entre processus.

Types d'applications : Systèmes complexes.

Approche de conception : Traduction de la spécification donnée en C^X dans une représentation interne appelée graphe syntaxique étendu. Ensuite, une simulation préliminaire et un profilage (*profiling*) sont réalisés. Cet environnement possède un outil de découpage logiciel/matériel qui est automatique et basé sur un algorithme de recuit simulé.

La stratégie adaptée pour le découpage matériel/logiciel consiste à partir d'une solution purement logicielle et d'extraire les parties critiques pour les réaliser en matériel en exploitant les informations obtenues par le profilage. Les parties à réaliser en logiciel sont traduites en langage C et les parties à réaliser en matériel sont traduites en langage HardwareC.

Architecture cible : L'architecture cible utilise un seul processeur avec un seul circuit intégré (ASIC), une mémoire et un bus.

Notes : Le système de découpage vérifie si les contraintes sur le temps sont satisfaites et localise les fautes.

Références : [ErHe95, HeEr94, HeHe94]

CREW	*Concurrent Read Exclusive Write* Lectures parallèles et écritures exclusives
CSP	*Communicating Sequential Processes* Langage de description de processus séquentiels communicants
CU	*Channel Unit* Canal de communication
DSP	*Digital Signal Processing* Traitement de signal
DU	*Design Unit* Unité de conception
EDIF	*Electronic Design Interchange Format* Format d'échange pour la conception de systèmes électroniques
EREW	*Exclusive Read Exclusive Write* Lectures exclusives et écritures exclusives
ESTEREL	Langage parallèle synchrone ayant une sémantique mathématique
ESTELLE	Langage standard OSI pour la spécification de protocoles de communication
Ethernet	Réseau local de communication
FDDI	*Fiber Distributed Data Interface* Protocole standard pour réseau local basé sur une structure d'anneau à jeton et utilisant des fibres optiques pour le câblage
FPGA	*Field Programmable Gate Array* Circuit programmable pour l'émulation
HardwareC	Langage basé sur une extension au langage C pour intégrer le parallèlisme et des aspects pour la description de matériel
IEEE	*Institute of Electrical and Electronics Engineers* Organisation de normalisation dans le domaine électronique
IPC	*Inter Process Communication* Communication inter processus sous UNIX
ISO	Organisme de Standardisation Internationale

LOTOS *LOgical Temporal Ordering Specification*
Langage standard OSI de description formelle de protocoles et de systèmes distribués

MEF Machine à États Finis

Niveau-système Le plus haut niveau d'abstraction pour la description de systèmes

PARTIF *PARTItioning of extended Finite state machines*
Outil de découpage au niveau système développé au cours de cette thèse

Partition Entité formée d'une ou plusieurs fonctions d'un système et qui seront réalisées sur le même processeur physique

PLD *Programmable Logic Device*
Composant logique programmable

Ptolemy Environnement de simulation et de conception logiciel/matériel développé à l'université de Berkeley (Californie, USA)

RAPID Outil de conception logicielle/matérielle à l'Université de Cincinnati

RISC *Reduced Instruction Set Computer*
Ordinateur à jeu d'instructions réduit

Réseaux de Petri Formalisme puissant permettant de représenter les systèmes à événements discrets

RTL *Register Transfer Level*
Représentation d'une fonctionnalité au niveau transfert de registres

RPC *Remote Procedure Call*
Appel de procédures à distance

SART *Structured Analysis for Real-Time systems*
Langage de description d'automates qui permet de représenter les aspects dynamiques d'un système

SDL *Specification and Description Language*
Langage de spécification et de description

SILAGE Langage de description de matériel

SOLAR Modèle unifié de représentation des systèmes (développé à TIMA)

SpecCharts Langage de spécification au niveau système et du type StateCharts avec instructions très semblables à celles décrites en VHDL comportemental. Ce langage a été développé à l'Université d'Irvine

SpecSyn Outil de conception logicielle/matérielle à l'Université d'Irvine

SpeedChart Langage de description graphique et textuel et qui utilise le formalisme de StateCharts

ST *State Table*

Table d'États

StateCharts Langage synchrone ayant un formalisme visuel et permettant de décrire des MEFs ainsi que leur contrôle (arrêt et relance de processus) en fonction du temps

Statemate Produit logiciel commercialisé par i-Logix et qui combine les langages StateCharts, ActivityCharts et ModuleCharts

Synthèse Procédé de raffinement d'une spécification qui permet de rajouter des détails en vue de sa réalisation

Système Collection de sous-systèmes qui coopérent entre-eux

TOSCA Outil de conception logicielle/matérielle à Italtel (Italie)

UNITY Langage de spécification développé à l'Université de Tübingen en Allemagne

UNIX Système d'exploitation

Validation Procédé de contrôle permettant de s'assurer qu'une conception est conforme aux attentes de l'utilisateur

VDM *Vienna Development Method*

Langage de spécification standard OSI basé à la fois sur la théorie des ensembles et la logique des prédicats

Vérification Procédé de contrôle permettant de s'assurer qu'une conception est conforme à sa spécification

VERILOG Langage de description du matériel

VHDL *VHSIC Hardware Description Language*

Langage standard IEEE de description du matériel

VLSI *Very Large Scale Integration*

Intégration à très grande Echelle

VULCAN Outil de conception logicielle/matérielle à l'Université de Stanford

Z Langage de spécification basé sur la théorie des ensembles

CREW *Concurrent Read Exclusive Write*
Lectures parallèles et écritures exclusives

CSP *Communicating Sequential Processes*
Langage de description de processus séquentiels communicants

CU *Channel Unit*
Canal de communication

DSP *Digital Signal Processing*
Traitement de signal

DU *Design Unit*
Unité de conception

EDIF *Electronic Design Interchange Format*
Format d'échange pour la conception de systèmes électroniques

EREW *Exclusive Read Exclusive Write*
Lectures exclusives et écritures exclusives

ESTEREL Langage parallèle synchrone ayant une sémantique mathématique

ESTELLE Langage standard OSI pour la spécification de protocoles de communication

Ethernet Réseau local de communication

FDDI *Fiber Distributed Data Interface*
Protocole standard pour réseau local basé sur une structure d'anneau à jeton et utilisant des fibres optiques pour le câblage

FPGA *Field Programmable Gate Array*
Circuit programmable pour l'émulation

HardwareC Langage basé sur une extension au langage C pour intégrer le parallèlisme et des aspects pour la description de matériel

IEEE *Institute of Electrical and Electronics Engineers*
Organisation de normalisation dans le domaine électronique

IPC *Inter Process Communication*
Communication inter processus sous UNIX

ISO Organisme de Standardisation Internationale

LOTOS *LOgical Temporal Ordering Specification*
Langage standard OSI de description formelle de protocoles et de systèmes distribués

MEF Machine à États Finis

Niveau-système Le plus haut niveau d'abstraction pour la description de systèmes

PARTIF *PARTItioning of extended Finite state machines*
Outil de découpage au niveau système développé au cours de cette thèse

Partition Entité formée d'une ou plusieurs fonctions d'un système et qui seront réalisées sur le même processeur physique

PLD *Programmable Logic Device*
Composant logique programmable

Ptolemy Environnement de simulation et de conception logiciel/matériel développé à l'université de Berkeley (Californie, USA)

RAPID Outil de conception logicielle/matérielle à l'Université de Cincinnati

RISC *Reduced Instruction Set Computer*
Ordinateur à jeu d'instructions réduit

Réseaux de Petri Formalisme puissant permettant de représenter les systèmes à événements discrets

RTL *Register Transfer Level*
Représentation d'une fonctionnalité au niveau transfert de registres

RPC *Remote Procedure Call*
Appel de procédures à distance

SART *Structured Analysis for Real-Time systems*
Langage de description d'automates qui permet de représenter les aspects dynamiques d'un système

SDL *Specification and Description Language*
Langage de spécification et de description

SILAGE Langage de description de matériel

SOLAR Modèle unifié de représentation des systèmes (développé à TIMA)

SpecCharts Langage de spécification au niveau système et du type StateCharts avec instructions très semblables à celles décrites en VHDL comportemental. Ce langage a été développé à l'Université d'Irvine

SpecSyn	Outil de conception logicielle/matérielle à l'Université d'Irvine
SpeedChart	Langage de description graphique et textuel et qui utilise le formalisme de StateCharts
ST	*State Table* Table d'États
StateCharts	Langage synchrone ayant un formalisme visuel et permettant de décrire des MEFs ainsi que leur contrôle (arrêt et relance de processus) en fonction du temps
Statemate	Produit logiciel commercialisé par i-Logix et qui combine les langages StateCharts, ActivityCharts et ModuleCharts
Synthèse	Procédé de raffinement d'une spécification qui permet de rajouter des détails en vue de sa réalisation
Système	Collection de sous-systèmes qui coopérent entre-eux
TOSCA	Outil de conception logicielle/matérielle à Italtel (Italie)
UNITY	Langage de spécification développé à l'Université de Tübingen en Allemagne
UNIX	Système d'exploitation
Validation	Procédé de contrôle permettant de s'assurer qu'une conception est conforme aux attentes de l'utilisateur
VDM	*Vienna Development Method* Langage de spécification standard OSI basé à la fois sur la théorie des ensembles et la logique des prédicats
Vérification	Procédé de contrôle permettant de s'assurer qu'une conception est conforme à sa spécification
VERILOG	Langage de description du matériel
VHDL	*VHSIC Hardware Description Language* Langage standard IEEE de description du matériel
VLSI	*Very Large Scale Integration* Intégration à très grande Echelle
VULCAN	Outil de conception logicielle/matérielle à l'Université de Stanford
Z	Langage de spécification basé sur la théorie des ensembles

[BeJe95] T. Ben Ismail, and A.A. Jerraya, “Design Models and Steps for CoDesign,” Invited paper, Second Colloquium on Verification of Hardware-Software Codesigns, London, England, IEE Press, October 1995.

[VoBI94] M. Voss, T. Ben Ismail, A.A. Jerraya, and K-H. Kapp, “Towards a Theory for Hardware/Software Codesign,” Proc. Int'l Wshp on Hardware/Software Codesign (CODES/CASHE), Grenoble, France, IEEE CS Press, September 1994, pp. 173-180.

[VaCh95] C.A. Valderrama, A. Changuel, P.V. Raghavan, M. Abid, T. Ben Ismail, and A.A. Jerraya, “A Unified Model for Co-simulation and Co-synthesis of Mixed Hardware/Software Systems,” Proc. European Design & Test Conference (ED&TC), Paris, France, IEEE CS Press, March 1995.

[DaBI95] J-M. Daveau, T. Ben Ismail, and A.A. Jerraya, “Synthesis of System-Level Communication by an allocation-Based Approach,” Proc. Int'l Symposium on System Synthesis (ISSS), Cannes, France, IEEE CS Press, September 1995, pp. 150-155.

[OBBI93] K. O'Brien, T. Ben Ismail, and A.A. Jerraya, “A Flexible Communication Modelling Paradigm for System-Level Synthesis,” Handouts of Int'l Wshp on Hardware-Software Co-Design, Cambridge, Massachusetts, IEEE CS Press, October 1993.

Publications dans des conférences nationales

[BIsm94a] T. Ben Ismail, “A Method for Hardware-Software Codesign and Rapid Prototyping,” Actes des Journées scientifiques sur les Technologies et les Architectures, Paris, France, Mars 1994, pp. 25-30.

[BIA94c] T. Ben Ismail, and M. Abid, “Architecture Generation for Hardware/Software Codesign,” Actes des Journées des Jeunes Chercheurs en Architectures de Machines et Systèmes, Monastir, Tunisie, Décembre 1994, pp. 45-54.

Rapports internes

[BIsm94b] T. Ben Ismail, “A Survey of Hardware-Software Codesign Methodologies,” Internal Report, TIMA/INPG, Grenoble, France, March 1994.

[MaBI95] G. Marchioro, T. Ben Ismail, J-M Daveau, A.A. Jerraya, “Interactive Codesign Environment,” Internal Report, TIMA/INPG, Grenoble, France, November 1995.

[BeCo84] G. Berry, and L. Cosserat, “The Esterel Synchronous Programming Language and its Mathematical Semantics,” Tech Report, Ecole Nationale Supérieure des Mines de Paris, 1984.

[BeCo87] G. Berry, Ph. Courroné, G. Gonthier, “Programmation Synchrone des Systèmes Réactifs,” Techniques et Sciences Informatiques, Vol. 6, N° 4, pp. 305-316, 1987.

[Belh94] H. Belhadj, “Overview on Graphical Entries for High-Level Description,” IFIP Workshop on Logic and Architecture Synthesis, December 1994.

[BeSi92] D. Becker, R. Singh, and S. Tell, “An Engineering Environment for Hardware/Software Co-Simulation,” Proc. 29th Design Automation Conference (DAC), IEEE CS Press, pp. 129-134, 1992.

[BeJe95] T. Ben Ismail, and A.A. Jerraya, “Design Models and Steps for CoDesign,” Invited paper, Second Colloquium on Verification of Hardware-Software Codesigns, London, England, IEE Press, October 1995.

[BenI92] T. Ben Ismail, “Découpage de Systèmes de Contrôle en vue d'une Réalisation Sur Silicium,” DEA Dissertation, INPG/TIM3, June 1992.

[BIJe95] T. Ben Ismail, and A.A. Jerraya, “Synthesis Steps and Design Models for CoDesign,” IEEE Computer, special issue on rapid-prototyping of microelectronic systems, Vol. 28, N° 2, pp. 44-52, February 1995.

[BIOB95] T. Ben Ismail, K. O'Brien, and A.A. Jerraya, “PARTIF: An Interactive System-level Partitioning,” to appear in VLSI Design, Special issue on Decomposition systems, Publ. Gordon Breach & Science Publishers, 1995.

[BIDa95] T. Ben Ismail, J-M. Daveau, K. O'Brien, and A.A. Jerraya, “A System-Level Communication Synthesis Approach for Hardware/Software Systems,” to appear in Int'l Journal Microprocessors and Microsystems, special issue on Hardware/Software Codesign, Butterworth-Heinemann Publishers, 1995.

[BIMa96] T. Ben Ismail, G. Marchioro, and A.A. Jerraya, “Découpage de Systèmes VLSI à partir d'une Spécification de haut niveau,” accepté à Technique et Science Informatiques (TSI), Hermes (eds), 1996.

[BIOJ94] T. Ben Ismail, K. O'Brien, and A.A. Jerraya, “Interactive System-level Partitioning with PARTIF,” Proc. European Design & Test Conference (EDAC-ETC-EuroASIC), Paris, France, IEEE CS Press, pp. 464-468, March 1994.

[BIA94a] T. Ben Ismail, M. Abid, K. O'Brien, and A.A. Jerraya, “An Approach for Hardware-Software Codesign,” Proc. Int'l Wshp on Rapid System Prototyping (RSP), Grenoble, France, IEEE CS Press, pp. 73-80, June 1994.

[BIA94b] T. Ben Ismail, M. Abid, and A.A. Jerraya, “COSMOS: A CoDesign Approach for Communicating Systems,” Proc. Third Int'l Wshp on Hardware/Software Codesign (CODES/CASHE), Grenoble, France, IEEE CS Press, pp. 17-24, September 1994.

[BIA94c] T. Ben Ismail, M. Abid, “Architecture Generation for Hardware/Software Codesign,” Actes des Journées des Jeunes Chercheurs en Architectures de Machines et Systèmes, Monastir, Tunisie, pp. 45-54, Décembre 1994.

[BIJe94] T. Ben Ismail, and A.A. Jerraya, “Tutorial: Hardware/Software Codesign,” Invited paper, Eighth Brasilian Symposium on Integrated Circuits Design, Gramado, Brasil, pp. 17, November 1994.

[BIsm94a] T. Ben Ismail, “A Method for Hardware-Software Codesign and Rapid Prototyping,” Actes des Journées scientifiques sur les Technologies et les Architectures, Paris, France, pp. 25-30, Mars 1994.

[BIsm94b] T. Ben Ismail, “A Survey of Hardware-Software Codesign Methodologies,” Internal Report, TIMA/INPG, Grenoble, France, March 1994.

[BiNe84] A.D. Birrell, and B.J. Nelson, “Implementing Remote Procedure Calls,” ACM Trans. on Computer Systems, Vol 2, N° 1, pp. 39-59, February 1984.

[BoBu93] G. Boriello, K. Buchenrieder, R. Camposano, E. Lee, R. Waxman, and W. Wolf, “Hardware/Software Codesign,” IEEE Design & Test of Computers, RoundTable, pp. 83-91, March 1993.

[BoKa87] G. Boriello, and R. Katz, “Synthesis of Interface Transducer Logic,” Proc. Int'l Conf. on Computer-Aided Design (ICCAD), IEEE CS Press, 1987.

[BoBr87] T. Bolognesi, and E. Brinksma, “Introduction to the ISO Specification Language LOTOS,” Computer Networks & ISDN Systems, Vol. 14, N° 1, North-Holland, pp. 25-29, 1987.

[BoSt93] J. Bowen, and V. Stavridou, “Systèmes à Sécurité de Fonctionnement Critique, Méthodes Formelles et Normes,” Génie logiciel & Systèmes experts, Mars 1993.

[Bram83] G.W. Bram, “Réseaux de Pétri: Théorie et Pratique,” Éditions Masson, Paris, 1983.

[BrSc87] Ed. Brinksma, G. Scollo, and C. Steenbergen, “LOTOS Specifications, Their Implementations and Their Tests,” in "Protocol Specification, Testing and Verification VI", G. Bochmann, B. Sarikaya (Eds), North Holland, pp. 349-360, 1987.

[Brui87] J. Bruijning, “Evaluation and Integration of Specification Languages,” Computer Networks & ISDN Systems, Vol. 13, pp. 75-89, 1987.

[BuVe92] K. Buchenrieder, and C. Veith, “CODES: A practical Concurrent design Environment,” Int'l Wshp on Hardware/software Co-design, Estes Park, Colorado, IEEE CS Press, October 1992.

[BuSe93] K. Buchenrieder, A. Sedlmeier, and C. Veith, “HW/SW Co-Design with PRAMs using CODES,” Proc. IFIP Conf. Hardware Description Languages (CHDL), Publ. Elsevier Science, Ottawa, Canada, April 1993.

[Buch94] K. Buchenrieder, “A Prototyping Environment for Control-Oriented HW/SW Systems using State-Charts, Activiy-Charts and FPGA's,” Proc. Euro-DAC with Euro-VHDL, Grenoble, France, IEEE CS Press, pp. 60-65, September 1994.

[CaCo94] P. Camurati, F. Corno, P. Prinetto, C. Bayol, and B. Soulas, “System-Level Modeling and Verification: A Comprehensive Design Methodology,” Proc. European Design & Test Conference (EDAC-ETC-EuroASIC), Paris, France, IEEE CS Press, February 1994.

[Calv93] J.P. Calvez, “Spécification et conception des ASICs,” Éditions Masson, Paris, Juin 1993, 580 pages.

[ChGi93] M. Chiodo, P. Giusto, A. Jurecska, L. Lavagno, H. Hsieh, A. Sangiovanni-Vincentelli, “A Formal Specification Model for Hardware/Software Codesign,” Wshp Handouts of Int'l Wshp on Hardware-Software Co-design, Cambridge, Massachusetts, IEEE CS Press, October 1993.

[ChJe95] A. Changuel, and A.A. Jerraya, “Taxonomy of Architectures for Codesign,” Internal Report, TIMA/INPG, Grenoble, France, January 1995.

[ClLo91] E.M. Clarke, D.E. Long, and K.L. McMillan, “A Language for Compositional Specification and Verification of Finite State Hardware Controllers,” Proceedings of the IEEE, Vol. 79, pp. 1283-1292, September 1991.

[Clar89] E.M. Clarke et al., “A Language for Compositional Specification and Verification of Finite State Hardware Controllers,” Tech. Rpt. CMU-CS-89-110, January 1989.

[DaBI95] J-M. Daveau, T. Ben Ismail, and A.A. Jerraya, “Synthesis of System-Level Communication by an allocation-Based Approach,” Proc. Int'l Symposium on System Synthesis (ISSS), Cannes, France, IEEE CS Press, September 1995.

[DaBI96] J-M. Daveau, T. Ben Ismail, G. Marchioro, and A.A. Jerraya, “Protocol Selection and Interface Generation for HW-SW Codesign,” submitted to IEEE Transactions on VLSI Systems, Special issue on Design Automation of complex integrated systems, September 1996.

[DeBo95] H. De Man, I. Bolsens, B. Lin, K. Van Rompaey, S. Vercauteren, and D. Verkest, “Co-design of DSP systems,” NATO ASI Hardware/Software Co-Design, Tremezzo, June 1995.

[DoPi93] A. Dorseuil, and P. Pillot, “Le Temps Réel en Milieu Industriel,” DUNOD Publishers, Paris, 1993, 296 pages.

[DrHa89] D. Drusinsky, and D. Harel, “Using StateCharts for Hardware Description and Synthesis,” IEEE Trans. CAD, Vol. 8, N° 7, pp. 798-807, July 1989.

[Dutt90] N. Dutt et al., “An Intermediate Presentation for Behavioral Synthesis,” Proc. 27th Design Automation Conf. (DAC), IEEE CS Press, pp. 14-19, Orlando, June 1990.

[Dzer90] D. Dzerzgowski, “Quatre Exemples de Langages et Environnements où le Temps Intervient,” Techniques et Sciences Informatiques (TSI), Avril 1990.

[EDIF88] Electronic Design Interchange Format Version 2 0 0, *Electronic Industries Association*, May 1988.

[EdFo94] M. Edwards, and J. Forrest, “A Development Environment for the Cosynthesis of Embedded Software/Hardware Systems,” Proc. European Design & Test Conference (EDAC-ETC-EuroASIC), Paris, France, IEEE CS Press, pp. 469-473, February 1994.

[ErHe93] R. Ernst, J. Henkel, and Th. Benner, “Hardware-Software Cosynthesis for Microcontrollers,” IEEE Design & Test of Computers, Vol. 10, N° 4, pp. 64-75, December 1993.

[FiKu93] D. Filo, D.C. Ku, C.N. Coelho, and G. DeMicheli, “Interface Optimisation for Comcurrent Systems Under Timing Constraints,” IEEE Trans. on VLSI Systems, Vol. 1, pp. 268-281, September 1993.

[Fish94] G.E. Fisher, "Rapid System Prototyping in an Open System Environment," Proc. Int'l Wshp on Rapid System Prototyping (RSP), Grenoble, France, IEEE CS Press, June 1994.

[FlJe93] H. Fleukers, and J.A. Jess, "ESCAPE: A Flexible Design and Simulation Environment," Proc. The Synthesis and Simulation Meeting and International Interchange, SASIMI'93, pp. 277-288, October 1993.

[Fros93] B.K. Fross, "Modeling Techniques using VHDL/C-language Interfacing," March 30, 1993.

[GaDu92] D. Gajski, N. Dutt, and A. Wu, "HIGH-LEVEL SYNTHESIS - Introduction to Chip and System Design," Kluwer Academic Publishers, Boston, 1992.

[GaVa95] D. Gajski, and F. Vahid, "Specification and Design of Embedded Systems," IEEE Design & Test of Computers, pp. 53-67, Spring 1995.

[GaVa94] D. Gajski, F. Vahid, and S. Narayan, "A System-Design Methodology: Executable-Specification Refinement," Proc. European Design & Test Conference (EDAC-ETC-EuroASIC), Paris, France, IEEE CS Press, pp. 458-463, February 1994.

[GlVe91] W. Glunz, and G. Venzl, "Hardwre Design using CASE Tools," Proc. VLSI'91, Edinburgh, Scotland, 1991.

[GlKr93] W. Glunz, T. Kruse, T. Rossel, and D. Monjau, "Integrating SDL and VHDL for System-Level Hardware Design," Proc. IFIP Conf. Hardware Description Languages (CHDL), Publ. Elsevier Science, Ottawa, Canada, April 1993.

[Goer92] R. Goering, "Emerging Tools and High-Level Design," Electronic Engineering Times, 10 August 1992.

[GoBo94] G. Goossens, I. Bolsens, B. Lin, and F. Catthoor, "Design of heterogeneous ICs for mobile and personal communication systems," Proc. Int'l Conf. on Computer-Aided Design (ICCAD), San Jose, California, IEEE CS Press, pp. 524-531, November 1994.

[GoGa94] J. Gong, D. Gajski, and S. Narayan, "Software Estimation from Executable Specifications," Proc. European Design Automation Conf. (EuroDAC), IEEE CS Press, Grenoble, France, September 1994.

[GuCo94] R.K. Gupta, C.N. Coelho Jr., and G. DeMicheli, "Program Implementation Schemes for Hardware-Software Systems," IEEE Computer, Vol. 27, N° 1, pp. 48-55, January 1994.

[GuDM93] R.K. Gupta, and G. DeMicheli, "Hardware-Software Cosynthesis for Digital Systems," IEEE Design & Test of Computers, Vol. 10, N° 3, pp. 29-41, September 1993.

[GuDM92] R.K. Gupta, and G. DeMicheli, "System-level Synthesis using Re-programmable Components," Proc. Third European Conference on Design Automation, IEEE CS Press, pp. 2-7, 1992.

[GuC92a] R.K. Gupta, C.N. Coelho Jr., and G. DeMicheli, "Progrm Implementation Schemes for Hardware-Software Systems," Wshp Handouts of Int'l Wshp on Hardware-Software Co-design, IEEE CS Press, October 1992.

[GuC92b] R.K. Gupta, C.N. Coelho Jr., and G. DeMicheli, “Synthesis and Simulation of Digital Systems Containing Interacting Hardware and Software Components,” Proc. 29th Design Automation Conference (DAC), IEEE CS Press, pp. 225-230, 1992.

[Halb93] N. Halbwachs, “Synchronous Programming of Reactive Systems,” Netherlands, Kluwer Academic Publishers, 1993, 174 pages.

[HaMe93] K. Ten Hagen, and H. Meyer, “Timed and Untimed Hardware/Software Cosimulation: Application and Efficient Implementation,” Int'l Wshp on Hardware-Software Codesign, Cambridge, IEEE CS Press, October 1993.

[HaPi88] D. Hatley, and I. Pirbhai, “Strategies for Real-Time Systems Specification,” Dorset House, 1988.

[HaPn83] D. Harel, and A. Pnueli, “On the Development of Reactive Systems,” Logics and Models of Concurrents Systems, ASI Series, Springer, N.Y., pp. 477-498, 1983.

[Hare90] D. Harel et al., “STATEMATE: A Working Environment for the Development of Complex Reactive Systems,” IEEE Trans. on Software Engineering, Vol. 16, N° 4, pp. 403-413, April 1990.

[Hare87] D. Harel, “Statecharts : A Visual Formalism for Complex Systems,” Science of Computer Programming 8, North-Holland, pp. 231-274, 1987.

[Hart66] J. Hartmanis, and R.E. Stearns, “Algebraic Structure Theory of Sequential Machines,” Prentice-Hall, 1966.

[HeEr94] J. Henkel, R. Ernst, U. Holtman, and T. Benner, “Adaptation of Partitioning and High-Level Synthesis in Hardware/Software Co-Synthesis,” Proc. Int'l Conf. on Computer-Aided Design (ICCAD), IEEE CS Press, pp. 96-100, 1994.

[HePa92] J.L. Hennessy, and D.A. Patterson, “Architecture des ordinateurs: Une approche quantitative,” Édition française par D. Etiemble, M. Israël, McGRAW-HILL Publishers, Paris, 1992, 751 pages.

[HePa94] J.L. Hennessy, and D.A. Patterson, “Organisation et conception des ordinateurs: L'interface Matériel/Logiciel,” Édition française par P. Klein, DUNOD Publishers, Paris, 1994, 660 pages.

[HeHe94] D. Herman, J. Henkel, and R. Ernst, “An Approach to the Adaptation of Estimated Cost Parameters in the Cosyma System,” Proc. Third Int'l Wshp on Hardware/Software Codesign (CODES/CASHE), Grenoble, France, IEEE CS Press, pp. 100-107, September 1994.

[Hilf85] P. Hilfinger, “A High-Level Language and Silicon Compiler for Digital Signal Processing,” Proc. IEEE CICC, Portlant, pp. 213-216, May 1985.

[Hoar74] C. Hoare, “Monitors: An Operating System Structuring Concept,” Comm. ACM, Vol. 17, N° 10, October 1974.

[Hoar78] C. Hoare, “Communicating Sequential Processes,” Comm. ACM, Vol. 21, N° 8, pp. 666-677, August 1978.

[Holz91] G.J. Holzmann, “Design and Validation of Computer Protocols,” Prentice-Hall, Englewood Cliffs, N.J., 1991.

[ISO87] International Standard, ESTELLE (Formal description technique based on an extended state transition model), ISO/DIS 9074, 1987.

[Jaul90] P. Jaulent, "Génie logiciel : les méthodes SADT, SA, E-A, SA-RT, SREM," Armand Colin Publishers, Paris, 1990, 288 pages.

[JeOB94] A.A. Jerraya, and K. O'Brien, "SOLAR: An Intermediate Format for System-Level Modeling and Synthesis," in "Computer Aided Software/Hardware Engineering," J. Rozenblit, K. Buchenrieder (eds), IEEE Press, Piscataway, N.J., pp 147-175, 1994.

[JeOB93] A.A. Jerraya, K. O'Brien, and T. Ben Ismail, "Linking System Design Tools and Hardware Design Tools," Proc. IFIP Conference on Hardware Description Languages (CHDL), Ed. D. Agnew, L. Claesen, and R. Composano, Publ. Elsevier Science, Ottawa, Canada, pp. 331, April 1993.

[JeOB92] A.A. Jerraya, and K. O'Brien, "SOLAR: An Intermediate Format for System-Level Design and Specification," Wshp on Hardware/Software Co-Design, Germany, IEEE CS Press, 1992.

[JePO93] A.A. Jerraya, I. Park, and K. O'Brien, "AMICAL: An Interactive High-Level Synthesis Environment," Proc. European Design & Test Conference (EDAC-ETC-EuroASIC), Paris, France, IEEE CS Press, February 1993

[Jone87] G. Jones, "Programming in Occam," Prentice-Hall, Englewood Cliffs, N.J., 1987.

[Jone90] C.B. Jones, "Systematic Software Development Using VDM," C.A.R Hoare Series, Prentice Hall International Series in Computer Science, 1990.

[Józw90] L. Józwiak, "Simultaneous Decomposition of Sequential Machines," Microprocessing & Microprogramming, North-Holland, Vol. 30, pp. 305-312, August 1990.

[JóKo91] L. Józwiak, and J. Kolsteren, "An Efficient Method for the Sequential General Decomposition of Sequential Machines," Microprocessing & Microprogramming, North-Holland, Vol. 32, pp. 657-664, August 1991.

[KaLe94a] A. Kalavade, and E.A. Lee, "Manifestations of Heterogeneity in Hardware/Software Codesign," Proc. 31st Design Automation Conference (DAC), IEEE CS Press, pp. 437-438, June 1994.

[KaLe94b] A. Kalavade, and E.A. Lee, "A Global Criticality/Local Phase Driven Algorithm for the Constrained Hardware/Software Partitioning," Proc. Third Int'l Wshp on Hardware/Software Codesign (CODES/CASHE), Grenoble, France, IEEE CS Press, pp. 42-48, September 1994.

[KaLe93] A. Kalavade, and E.A. Lee, "A Hardware-Software Codesign Methodology for DSP Applications," IEEE Design & Test of Computers, Vol. 10, N° 3, pp. 16-28, September 1993.

[Keut94] K. Keutzer, "Hardware-Software Co-Design and ESDA," Proc. 31st Design Automation Conference (DAC), IEEE CS Press, pp. 435-436, June 1994.

[KuD88] D.C. Ku, and G. DeMicheli, "HardwareC - A Language for Hardware Design," Tech. Rpt. N° CSL-TR-88-362, Computer Systems Lab, Stanford University, August 1988.

[Lamp89] L. Lamport, "A Simple Approach to Specifying Concurrent Systems," Comm. ACM, Vol. 32, N° 1, January 1989.

[Lapl92] Ph. Laplante, "Real-Time Systems design and Analysis, An Engineer's Handbook," IEEE CS Press, 1992.

[LaSe91] C. Lavarenne, O. Seghrouchni, Y. Sorel, and M. Sorine, "The SynDex software environment for real-time distributed systems design and implementation," Proc. European Control Conference, July 1991.

[LeRa93] S. Lee, and J. Rabaey, "A Hardware Software Cosimulation Environment," Handouts Int'l Wshp on Hardware-Software Codesign, Cambridge, IEEE CS Press, October 1993.

[LiVe94] B. Lin, and S. Vercauteren, "Synthesis of Concurrent System Interface Modules with Automatic Protocol Conversion Generation," Proc. Int'l Conf. on Computer-Aided Design (ICCAD), San Jose, California, IEEE CS Press, pp. 101-108, November 1994.

[LoDo93] W.M. Loucks, B.J. Doray, and D.G. Agnew, "Experiences in Real Time Hardware-Software Cosimulation," Proc. VHDL Int'l Users Forum (VIUF), Ottawa, Canada, pp.47-57, April 1993.

[Loto89] International Standard Organisation, "LOTOS: A Formal Description Technique Based on the Temporal Ordering of Observational Behavior," ISO, IS 8807, February 1989.

[MaBI95] G. Marchioro, T. Ben Ismail, J-M Daveau, A.A. Jerraya, "Interactive Codesign Environment," Internal Report, TIMA/INPG, Grenoble, France, November 1995.

[Mali93] L. Maliniak, "ESDA Boosts CAE Technology to Higher Levels," Electonic Design Report, Electronic Design, pp. 61-72, December 1993.

[Mara90] F. Maraninchi, "Argos : Un langage graphique pour la conception, la description et la validation des systèmes réactifs," Thèse d'informatique, Université Joseph Fourier Grenoble, 1990.

[Mara89] F. Maraninchi, "ARGONAUTE: Graphical Description Semantics and Verification of Reactive Systems by using a Process Algebra," Workshop on Automatic Verification of Finite State Systems, Grenoble, 1989.

[Mart90] A.J. Martin, "Synthesis of Asynchronous VLSI Circuits," Formal Methods for VLSI Design, J. Staunstrup (eds), North-Holland, 1990.

[MiLa92] P. Michel, U. Lauther, and P. Duzy, "The Synthesis Approach to Digital System Design," Kluwer Academic Publishers, 1992.

[Miln83] R. Milner, "Calculi for Synchrony and Asynchrony," Theoretical Computer Science, Vol. 23, pp. 267-310, 1983.

[MFK90] M. McFarland, and T. Kowalski, "Incorporating Bottom-Up Design into Hardware Synthesis," IEEE Trans. Computer Aided Design, September 1990.

[Moal81] M. Moalla, "Spécification et conception d'automatismes discrets complexes, basées sur l'utilisation du GRAFCET et des réseaux de Petri," Thèse Es Science informatique-automatique, INPG, Grenoble, 1981.

[MoEv95] D. Morris, G. Evans, and S. Schofield, "Simulating the Behaviour of Computer System Models – Cosimulation of Hardware/Software," submitted to the Computer Journal, 1995.

[NaBa86] E. Najm, and J. Barre, "LOTOS, Un Nouveau Langage de Spécification des Systèmes Distribués," 3ème Congrés des Nouvelles Architectures pour les Communications, 1986.

[NaGa94a] S. Narayan, and D. Gajski, “Synthesis of System-Level Bus Interfaces,” Proc. European Design & Test Conference (EDAC-ETC-EuroASIC), Paris, France, IEEE CS Press, pp. 395-399, February 1994.

[NaGa94b] S. Narayan, and D. Gajski, “Protocol Generation for Communication Channels,” Proc. 31st Design Automation Conference (DAC), IEEE CS Press, pp. 547-551, 1994.

[NaGa95] S. Narayan, and D. Gajski, “Interfacing Incompatible Protocols Using Interface Process Generation,” Proc. Design Automation Conference (DAC), IEEE CS Press, pp. 468-473, June 1995.

[NaVa91] S. Narayan, F. Vahid, and D. Gajski. "System Specification and Synthesis with the SpecCharts Language,” Proc. Int'l Conf. on Computer-Aided Design (ICCAD), IEEE CS Press, pp. 226-269, November 1991.

[OBBI93] K. O'Brien, T. Ben Ismail, and A.A. Jerraya, “A Flexible Communication Modelling Paradigm for System-Level Synthesis,” Handouts of Int'l Wshp on Hardware-Software Co-Design, Cambridge, Massachusetts, IEEE CS Press, October 1993.

[OBPa92] K. O'Brien, I. Park, and A.A. Jerraya, “Communication Modelling for the System-Level Synthesis of Mixed Hardware/Software Designs,” Proc. European Design & Test Conference (EDAC-ETC-EuroASIC), Brussels, Belgium, IEEE CS Press, March 1992.

[OBri93] K. O'Brien, “Compilation de silicium: du circuit au système,” Thèse de microélectronique, Institut National Polytechnique de Grenoble (INPG), Mars 1993.

[Ostr89] J.S. Ostroff, “Temporal Logic for Real-Time Systems,” Research Study Press Ltd, Advanced Software Developpment Series, 1989.

[OuJe95] G. Ouillade, A.A. Jerraya, et J.M. Fleurant, “Modélisation dans un Espace Virtuel de Systèmes Électroniques Maintenables,” Rapport ASTER Ingénierie, France, Avril 1995.

[PaLi94] P. Paulin, C. Liem, T. May, and S. Sutarwala, “DSP Design Tool Requirements for Embedded Systems: A Telecommunications Industrial Perspective,” in Journal of VLSI Signal Processing (special issue on synthesis for real-time DSP), Kluwer Academic Publishers, 1994.

[PeKu93] Z. Peng, and K. Kuchcinki, “An Algorithm for Partitionning of Application Specific Systems,” Proc. European Design & Test Conference (EDAC-ETC-EuroASIC), IEEE CS Press, February 1993.

[Peng91] W. Peng, “Data Flow Analysis of Communicating Finite State Machines,” ACM Transactions on Programming Languages and Systems, Vol. 13, N° 3, pp. 399-442, July 1991.

[Pnue77] A. Pnueli, “The Temporal logic for Programs,” Proc. 18th IEEE Symposium on Foundations of Computer Science, pp. 46-57, 1977.

[PuKr92] O. Pulkkinen, and K. Kronlöf, “Integration of SDL and VHDL for high level digital system,” Proc. Euro-VHDL, Hamburg, Germany, pp. 624-629, 1992.

[Quin94] P. Quinton, “Architectures Spécialisées,” École d'été des Jeunes Chercheurs en Architectures de Machines et Systèmes, Couiza, France, Juillet 1994.

[ReWi93] N.L. Rethman, and P.A. Wilsey, “RAPID: A Tool for Hardware/Software Tradeoff Analysis,” Proc. IFIP Conf. Hardware Description Languages (CHDL), Publ. Elsevier Science, Ottawa, Canada, April 1993.

[Rich94] M.A. Richards, “The Rapid Prototyping of Application Specific Signal Processors (RASSP) Program: Overview and Status,” Proc. Int'l Wshp on Rapid System Prototyping (RSP), Grenoble, France, IEEE CS Press, pp. 1-6, June 1994.

[Romd92] M. Romdhani, “Compilation du Langage SDL en vue d'une Réalisation Matérielle,” Final Year Project Report, INPG/TIM3, June 1992.

[RoHa95] M. Romdhani, R.P. Hautbois, A. Jeffroy, P. de Chazelles, and A.A. Jerraya, “Evaluation and Composition of Specification Languages, an Industrial Point of View,” Proc. IFIP Conf. Hardware Description Languages (CHDL), Japan, pp. 519-523, September 1995.

[Rows94] J.A. Rowson, “Hardware/Software Co-Simulation,” Proc. 31st Design Automation Conference (DAC), IEEE CS Press, pp. 439-440, June 1994.

[SaPr90] K. Salah, and R. Probert, “A Service-Based Method for the Synthesis of Communication Protocols,” Int'l Journal of Mini and Microcomputers, Vol. 12, N° 3, pp. 97-103, 1990.

[SaTi87] R. Saracco, and P.A.J. Tilanus, “CCITT SDL: An Overview of the Language and its Applications,” Computer Networks & ISDN Systems, Special Issue on CCITT SDL, Vol. 13, N° 2, pp. 65-74, 1987.

[SDL88] CCITT, Livres bleux, recommandations, Volumes X (X.135), Fascicules: X.1, X.2, X.3 et X.4, Genève, 1988.

[Somm89] I. Sommerville, “Software Engineering,” 4th edition, Addison-Wesley Publishers, London, Amsterdam, International computer science series, 1989, 290 pages.

[Spiv89] J.M. Spivey, “An Introduction to Z Formal Specifications,” Software Engineering Journal, pp. 40-50, January 1989.

[SrBr93] M.B. Srivastava, and R.B. Brodersen, “Using VHDL for High-level, Mixed-Mode Simulation,” IEEE Design & Test of Computers, pp. 31-40, September 1993.

[SrBr91] M.B. Srivastava, and R.B. Brodersen, “Rapid-prototyping of Hardware and Software in a Unified Framework,” Proc. Int'l Conf. on Computer-Aided Design (ICCAD), IEEE CS Press, pp. 152-155, 1991.

[Sriv92] M.B. Srivastava, “Rapid-prototyping of Hardware and Software in a Unified Framework,” Ph.D. thesis, University of Calif. Berkeley, June 1992.

[Syno93] “Synopsys VHDL System Simulator Interfaces Manual: C-language Interface,” Synopsys Inc., Version 3.0b, June 1993.

[ThAd93] D.E. Thomas, J.K. Adams, and H. Schmit, “A Model and Methodology for Hardware-Software Codesign,” IEEE Design & Test of Computers, Vol. 10, N° 3, pp. 6-15, September 1993.

[ThLW90] D.E. Thomas, E. Dirkes-Lagnese, R.A. Walker, J.A. Nestor, J.V. Rajan, and R.L. Blackburn, “Algorithmic and Register-Transfer Level Synthesis: The System's Architect's Workbench,” Boston, Kluwer Academic Publishers, 1990.

[ThMo91] D.E. Thomas, and P.R. Moorby, "The Verilog Hardware Description Language," Boston, Kluwer Academic Publishers, pp. 146-167, 1991.

[Tier88] R.A. Tierney, "Modelling Complex Systems," VLSI System Design, May 1988.

[UnSm92] D. Ungar, R.B. Smith, C. Chambers, and U. Holzle, "Object, Message, and Perfomance: How they Coexist in Self," IEEE Computer, October 1992.

[VaCh95] C.A. Valderrama, A. Changuel, P.V. Raghavan, M. Abid, T. Ben Ismail, and A.A. Jerraya, "A Unified Model for Co-simulation and Co-synthesis of Mixed Hardware/Software Systems," Proc. European Design & Test Conference (EDAC-ETC-EuroASIC), Paris, France, IEEE CS Press, March 1995.

[VaGa92] F.Vahid, and D. Gajski, "Specification Partitioning for System Design," Proc. 29th Design Automation Conf. (DAC), IEEE CS Press, June 1992.

[Vahi91] F. Vahid, "A Survey of Behavioral-level Partitioning Systems," UC Irvine, TR ICS 91-71, October 1991.

[VaNa91] F. Vahid, S. Narayan, and D. Gajski, "SpecCharts: A Language For System-Level Synthesis," Proc. IFIP Conf. Hardware Description Languages (CHDL), Publ. Elsevier Science, pp. 145-154, April 1991.

[VaVa93] J. VanHoof, K. VanRompaey, I. Bolsens, G. Goossens and H. DeMan, High-Level Synthesis for Real-Time Digital Signal Processing, Kluwer Academic Publishers, 1993.

[VeMa94] R. Venkateswaran, and P. Mazumder, "A Survey of DA Techniques for PLD and FPGA based Systems," Integration, the VLSI Journal, Elsevier Science Publishers, Vol. 17(3), pp. 191-240, 1994.

[Vhdl88] "IEEE Standard VHDL Language Reference Manual," IEEE, N.Y., 1988.

[VoBI94] M. Voss, T. Ben Ismail, A.A. Jerraya, and K.H. Kapp, "Towards a Theory for Hardware/Software Codesign," Proc. Third Int'l Wshp on Hardware/Software Codesign (CODES/CASHE), Grenoble, France, IEEE CS Press, pp. 173-180, September 1994.

[WaBo93] E.A. Walkup, and G. Boriello, "Automatic Synthesis of Device Drivers for Hardware/Software Co-design," Int'l Wshp on Hardware-Software Codesign, Cambridge, IEEE CS Press, October 1993.

[WaMe85] P. Ward, and S. Mellor, "Structured Development for Real-Time Systems," Yourdon Press, 1985.

[Wolf90a] W. Wolf, "An Algorithm for Nearly-Minimal Collapsing of Finite-State Machine Networks," Proc. Int'l Conf. on Computer-Aided Design (ICCAD), IEEE CS Press, pp. 80-83, 1990.

[Wolf90b] W. Wolf, "The FSM Network Model For Behavioral Synthesis of Control-Dominated Machines," Proc. 27th Design Automation Conf. (DAC), IEEE CS Press, pp. 692-697, 1990.

[Wolf93] W. Wolf, "Guest Editor's Introduction: Hardware-Software Codesign," IEEE Design & Test of Computers, Vol. 10, N° 3, pp. 5, September 1993.

[Wolf94] W. Wolf, "Hardware-Software Codesign of Embedded Systems," Proc. IEEE, Vol. 82, N° 7, pp. 967-989, 1994.

[WoM93] W. Wolf, and R. Manno, “High-level Modeling and Synthesis of Communicating Processes using VHDL,” IEICE Trans. Information & Systems, E76-D(9), pp. 1039-1046, September 1993.

[WoTa91] W. Wolf, A. Takach, and T-C. Lee, “Architectural Optimization Methods for Control-Dominated Machines,” Eds. R. Camposano & W. Wolf, High-Level VLSI Synthesis, Kluwer Academic Publishers, pp. 231-254, 1991.

[WoDu94] N.S. Woo, A.E. Dunlop, and W. Wolf, “Codesign from Cospecification,” IEEE Computer, Vol. 27, N° 1, pp. 42-47, January 1994.

[YeWo95] T. Yen, and W. Wolf, “Communication Synthesis for Distributed Embedded Systems,” Proc. Int'l Conf. on Computer-Aided Design (ICCAD), IEEE CS Press, pp. 288-294, November 1995.

[ZaLi93] P. Zanella, and Y. Ligier, “Architecture et technologie des ordinateurs,” DUNOD Publishers, Paris, 1993, 425 pages.

7.1. Conclusions

Dans cette thèse, l'objectif a été de proposer une approche et un environnement de synthèse de systèmes à partir d'un très haut niveau d'abstraction qui est le niveau système. L'approche considère particulièrement le problème de la conception conjointe logiciel/matériel. Ce travail a pris une vue globale d'un système, dans le processus de développement, afin d'identifier les vrais problèmes qui doivent être résolus. Le premier problème dans la conception de tels systèmes a été le développement simultané des aspects logiciels et matériels d'un système. Ce problème de conception de logiciel/matériel est présent à tous les niveaux de la conception d'un système - la spécification, la simulation, la génération d'architecture, et la réalisation physique. Basé sur ces constatations, cette thèse a permis l'investigation du problème de découpage logiciel/matériel en partant d'un langage de niveau système, la synthèse de la communication entre des composants logiciels et autres matériels, la co-simulation des descriptions mixtes logiciel/matériel, et la génération d'architecture.

Le chapitre 2 a permis de présenter l'état de l'art de la modélisation à travers les modèles les plus connus. Il a aussi caractérisé les langages de spécification au niveau système, et comparé les outils existants pour la conception conjointe logicielle/matérielle. Il s'agit des outils les plus avancés dans différents centres de recherches. Il a été montré que les approches concernées de conception différent par la spécification donnée en entrée, les domaines d'application, la méthode de synthèse, et l'architecture cible qui est considérée.

Le chapitre 3 a donné une description du modèle de représentation, appelé Solar, pour la synthèse. Ce modèle sert comme représentation interne utilisable par les outils de synthèse au niveau système. Le format Solar est basé sur une extension au modèle des machines à états finis. Il est à noter que le modèle d'exécution de Solar n'est pas défini sur la base d'une sémantique formelle. Néanmoins, Solar a l'avantage de pouvoir accommoder plusieurs langages de spécification et de rendre l'approche de conception conjointe logicielle/matérielle indépendante de ces langages.

Le chapitre 4 a proposé une vue globale des étapes nécessaires à la conception conjointe de systèmes logiciels/matériels qui sont décrits à l'aide d'un langage de spécification au niveau système. Dans ce chapitre, chaque étape de raffinement est décrite à travers les modèles qu'elle utilise et les transformations qu'elle est amenée à réaliser. L'approche de conception proposée dans cette thèse est interactive et se base sur la réutilisation d'éléments de bibliothèque.

Le chapitre 5 a expliqué l'activité de découpage d'une spécification de haut niveau en la distribuant sur un ensemble de partitions. Chaque partition renferme une partie des fonctions qui se trouvent dans la spécification initiale. Le découpage logiciel/matériel permet d'affecter chaque partition à une réalisation logicielle ou matérielle. L'approche de découpage qui a été décrite se base sur l'application, de manière interactive, d'un certain nombre de primitives qui réalisent des opérations de transformation d'une spécification.

Il est clair d'après les discussions et les exemples présentés dans cette thèse que la synthèse de niveau système présente des problèmes autres que ceux qui sont connus dans le niveau comportemental. Parmi ces problèmes, il est possible de citer la synthèse de communication, décrite dans le chapitre 6. Cette synthèse de communication présente deux aspects importants; Le premier est la sélection (choix) des protocoles de communication et de synchronisation qui permettent de communiquer entre modules logiciels, entre composants matériels, ou bien entre des modules logiciels et des composants matériels; Le second aspect consiste à adapter l'interface des sous-systèmes communicants. Cette adaptation peut avoir divers degrés de complexité selon que les interfaces de ces sous-systèmes sont déjà fixées (e.g. interface standard) ou bien qu'elles restent à définir. D'autre part, le logiciel et le matériel sont traités de manière symétrique afin de faciliter la migration de fonctionnalités entre le logiciel et le matériel sans aucune restriction. Bien entendu, la limitation se situe au niveau de l'architecture cible qui existe déjà ou bien qu'il reste à définir.

Aujourd'hui, les outils de compilation d'architecture appliqués aux circuits existent et sont relativement bien maîtrisés. Par contre, les études dans le domaine de la spécification, modélisation, et la synthèse de systèmes entiers, correspondant à un besoin réel des industriels, ne font qu'émerger. Ce travail constitue ce qu'on pourrait appeler une première génération de systèmes de conception mixte logiciel/matériel. Avant que ce type de techniques soit réellement utilisable en milieu industriel il faudrait encore une ou deux générations pour bien cerner et maîtriser les différents aspects de ce problème. Une comparaison avec les autres niveaux de synthèse montre que pour la synthèse comportementale, par exemple, qui est apparue au début des années 80, il a fallu trois ou quatre générations successives d'outils avant la maturation du domaine et l'acceptation de ce type d'outils en milieu industriel.

Vu l'effort, de recherche, consacré à la conception conjointe logiciel/matériel, il est possible d'espérer que les recherches avancent plus vite. D'une part, cette avancée sera d'autant plus rapide que plusieurs techniques utilisées dans la conception conjointe logiciel/matériel sont déjà utilisées dans d'autres domaines tels que les systèmes

distribués [Lapl92]. D'autre part, les industriels accordent une attention particulière à ce domaine de recherches en souhaitant que ces dernières puissent les aider à résoudre les problèmes engendrés par de nouvelles applications e.g. les applications multimédia portables.

7.2. Perspectives

Même si, au cours de cette thèse, une approche complète de conception descendante a été adoptée et présentée, une série de travaux sont encore à considérer à court terme. Parmi ces travaux, certains font partie de la liste non exhaustive suivante :

• La réutilisation massive de composants (paramétrables) de bibliothèque pouvant être des sous-systèmes logiciels ou matériels. Ceci permettra de réduire le temps de développement et d'attirer d'avantage l'attention et l'intérêt des industriels.

• L'estimation précise des facteurs, influants sur les performances d'un système, tels que la rapidité des temps de calcul et de communication, la surface de réalisation, le nombre d'interconnexions, la consommation de puissance, la fiabilité, et la maintenabilité. L'un des points qu'il reste à développer est la fonction d'évaluation, plus particulièrement l'estimation des performances de chaque partition en supposant sa réalisation sur un processeur (logiciel ou matériel) spécifique.

• La vérification précise des contraintes de temps réel à tous les niveaux d'une conception mixte logicielle/matérielle.

• La simulation mixte qui prend en compte des performances du niveau physique et qui intègre le logiciel et le matériel avec ses aspects numériques et analogiques. Il s'agit de pouvoir rétro-annoter (*backannotation*) la spécification système avec des paramètres de la réalisation.

Pour le long terme, d'autres travaux pourront venir compléter ceux qui ont été mentionnés plus haut :

• Le découpage automatique d'un système entre logiciel et matériel en utilisant les estimations de performances citées précédemment et en se basant sur une architecture spécifique. Ceci va permettre de réaliser une exploration plus large et plus rapide de l'espace de conception.

• La vérification formelle des transformations réalisées sur une description et qui servent à améliorer les performances de la réalisation. Il s'agit de vérifier d'une part que ces transformations (e.g. fusion et découpage) préservent le comportement initial, et d'autre part que les décisions prises par le concepteur sont valides vis à vis des contraintes initiales imposées sur la conception.

• Le développement d'outils d'expertise et d'exploration qui permettent par exemple, lors de la phase de découpage, de suggérer au concepteur d'utiliser un certain type de processeur, non disponible en bibliothèque, afin d'améliorer le niveau de performances avec un certain pourcentage. Ce genre d'outils est très souhaitable en milieu industriel pour que le concepteur, même expérimenté, ait le sentiment d'être réellement assisté par des outils de conception.

6.1. Introduction

Le problème de la synthèse de communication se pose généralement à la suite du découpage d'un système [BIJe95]. En effet, le découpage d'un système produit un ensemble de sous-systèmes communicants. Chaque sous-système aura à communiquer avec un ou plusieurs sous-systèmes à travers un ou plusieurs types de protocoles (standards ou spécifiques).

Ce chapitre présente une approche de synthèse de la communication qui est formulée comme un problème d'allocation. Le but est de pouvoir sélectionner un ensemble de composants de communication à partir des bibliothèques prévus à cet effet. Ensuite, l'interface de chaque sous-système sera adaptée aux protocoles préalablement alloués. Par comparaison, les approches classiques [FiKu93, NaGa94b] sont réalisées en une seule étape qui effectue la synthèse des interfaces en utilisant un réseau de communication fixe.

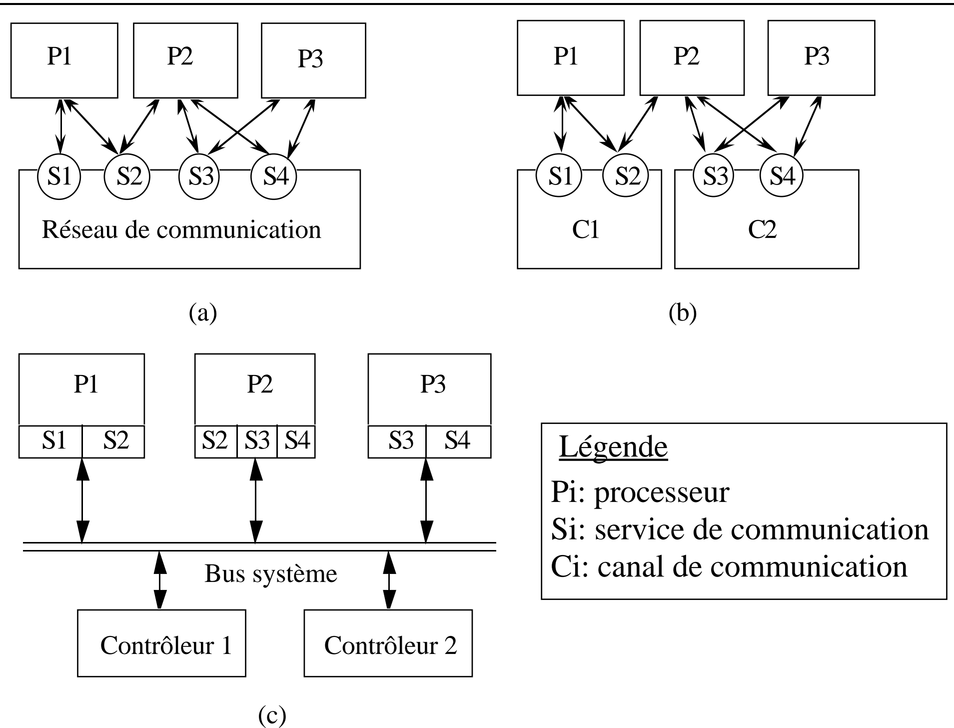
Les travaux réalisés jusqu'ici se sont concentrés, soit sur la synthèse de protocoles [SaPr90], soit sur la synthèse d'interfaces [NaGa94a, NaGa94b] mais rarement les deux à la fois [YeWo95]. Il faut noter qu'il y a deux niveaux de détails concernés. La synthèse de protocoles se préoccupe uniquement du type de protocole et donc de la spécification du comportement e.g. protocole synchrone ou asynchrone. Cette étape fixe aussi la structure de communication tel que le nombre de bus ou le nombre d'unités de communication en général. En revanche, la synthèse d'interfaces se concentre sur les détails de réalisation des interfaces (signaux d'E/S) des systèmes communicants. Ces systèmes peuvent communiquer à travers des canaux où chaque canal peut être une collection de fils, un bus (e.g. bus de données sur une puce ou bus VME) ou même un réseau (Ethernet, ATM, etc.).

Dans cette approche, la synthèse de communication est réalisée en deux étapes qui sont la sélection de protocoles et la synthèse (génération et adaptation) d'interfaces. L'objectif est de combiner à la fois la synthèse de protocoles et la synthèse d'interfaces pour avoir une synthèse intégrale de la communication [DaBI96]. L'avantage de procéder en deux étapes est d'avoir plus de contrôle dans cette activité de raffinement ce qui permet de faire des choix, par exemple de protocoles, et voir l'effet immédiat de l'incidence des choix réalisés. Cette synthèse procède donc par étapes interactives qui évitent de prendre des décisions très tôt lesquels risquent d'écarter plusieurs alternatives de réalisation. L'idée est donc de retarder autant que possible les choix concernant la réalisation

physique de la communication puisqu'une décision prise prématurément peut restreindre l'espace des solutions possibles pour la communication.

6.2. Les étapes de la synthèse de la communication

Cette section définit les étapes nécessaires pour réaliser la synthèse de la communication [BIDa95]. Le point de départ de cette synthèse est un ensemble de processeurs qui coopèrent entre eux à travers un réseau conceptuel (canal abstrait) de communication. Rappelons que dans le modèle utilisé, la spécification de la communication est séparée du reste de la conception (voir § 3.5, chapitre 3). L'objet de la synthèse de la communication d'un système est de transformer un système composé d'un ensemble de processeurs communicants, via des RPC [Andr91], par l'intermédiaire de primitives de haut niveau (figure 6.1(a)) en un ensemble de processeurs interconnectés (figure 6.1(c)). Ces derniers processeurs communiquent via des signaux et partagent le contrôle de la communication.



(a) processeurs communicant au niveau conceptuel, via des canaux abstraits,
(b) communication au niveau spécification après la synthèse des protocoles,
(c) communication au niveau physique après la synthèse d'interfaces.

Figure 6.1. *Approche de synthèse de la communication.*

La figure 6.1 montre l'activité de synthèse de communication et qui comporte deux étapes :

• La synthèse de protocoles, illustrée par le passage de la figure 6.1(a) à la figure 6.1(b), et

• La synthèse d'interfaces, qui est illustrée par le passage de la figure 6.1(b) à la figure 6.1(c).

Exemple 1: Dans cet exemple, il s'agit de partir d'un système, appelé Prod/Cons, préalablement découpé et qui est composé des deux processeurs P1 et P2. Ces processeurs communiquent entre eux à travers un canal logique noté par CU (voir figure 6.2). La synthèse de protocoles commence par allouer une unité de communication qui réalise ce canal logique. Comme il est montré dans la figure 6.2, cette allocation s'effectue à partir d'une bibliothèque composée de canaux de communication qui suivent le principe RPC. Chaque canal de communication de cette bibliothèque offre des services (par exemple A et B dans la figure 6.2) et un protocole donné (décrit dans le contrôleur). Ce canal de communication est une abstraction de plusieurs composants physiques de communication.

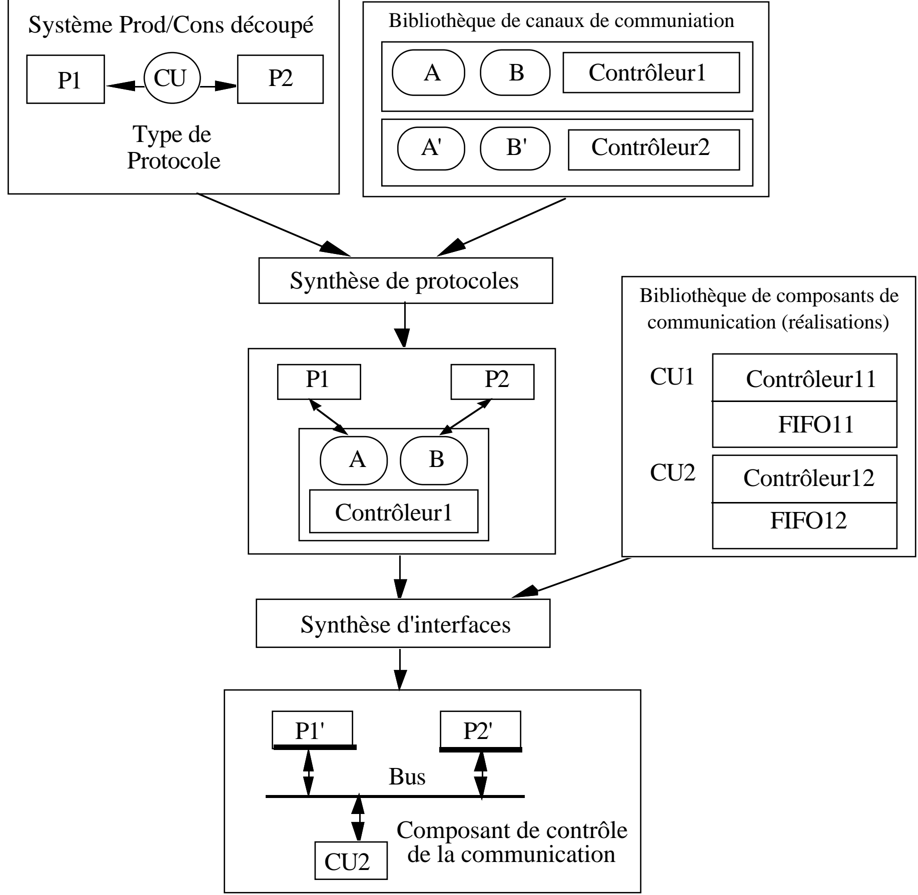


Figure 6.2. *Étapes de la synthèse de communication du système Prod/Cons.*

L'étape suivante est la synthèse d'interfaces. Cette étape va utiliser une autre bibliothèque de réalisation des composants de communication. En fait, on suppose que chaque unité de la première bibliothèque peut avoir plusieurs réalisations dans la seconde. L'objet de cette synthèse d'interfaces est de raffiner et d'adapter les interfaces des deux processeurs pour qu'ils puissent communiquer à travers un bus physique et éventuellement au moyen d'un contrôleur de communication. Les deux processeurs raffinés, P1' et P2' de la figure 6.2, contiennent en plus de la description du comportement initial des services et une interface spécifique pour cette communication. L'enchaînement des deux étapes est présenté dans la figure 6.2. Le résultat de cette synthèse de communication est composé des deux processeurs raffinés, d'un contrôleur de communication et d'un bus.

6.3. La synthèse de protocoles

6.3.1. Formulation du problème

La description en entrée pour la synthèse de protocoles est modulaire et composée de processeurs et de canaux logiques. Étant donnée une bibliothèque de canaux de réalisations physiques, le problème de sélection automatique de composants pour la communication peut être vu comme un problème classique d'allocation. Il s'agit d'allouer parmi les composants physiques de communication, pris dans une bibliothèque, les instances appropriées à chacun des canaux logiques. Au cours de cette thèse, on s'est intéressé à deux cas de figure. Le premier cas consiste à allouer pour chaque canal logique un canal physique qui peut assurer la communication. Cette allocation s'effectue moyennant une fonction coût. Le deuxième cas permet de réaliser plusieurs canaux logiques, en les multiplexant, par une seule instance de canal physique (voir figure 6.3). Deux algorithmes d'allocation ont alors été développés; le premier, appelé SELECT, permet de résoudre le cas de figure (1); le deuxième, appelé ALLOC, permet de résoudre le cas de figure (2). Comme le montre la figure 6.3, le cas (1) permet à un seul canal logique d'être réalisé par une seule instance de canal physique. Dans le cas (2), plusieurs canaux logiques (L2 et L3) peuvent aller sur une seule instance de canal physique (R4). Ceci permet le multiplexage de plusieurs canaux de mêmes protocoles sur un seul support physique de communication. Cette alternative a les avantages d'augmenter le taux d'utilisation d'un canal physique et de réduire le nombre de ports d'E/S nécessaires. En effet, en utilisant une seule instance de canal physique pour plusieurs canaux logiques, l'interface nécessaire sera inférieure en général à celle de plusieurs instances de canaux

physiques où chacun est relatif à un seul canal logique. Par contre, l'inconvénient d'une telle solution est qu'elle peut entraîner une dégradation (diminution) des performances.

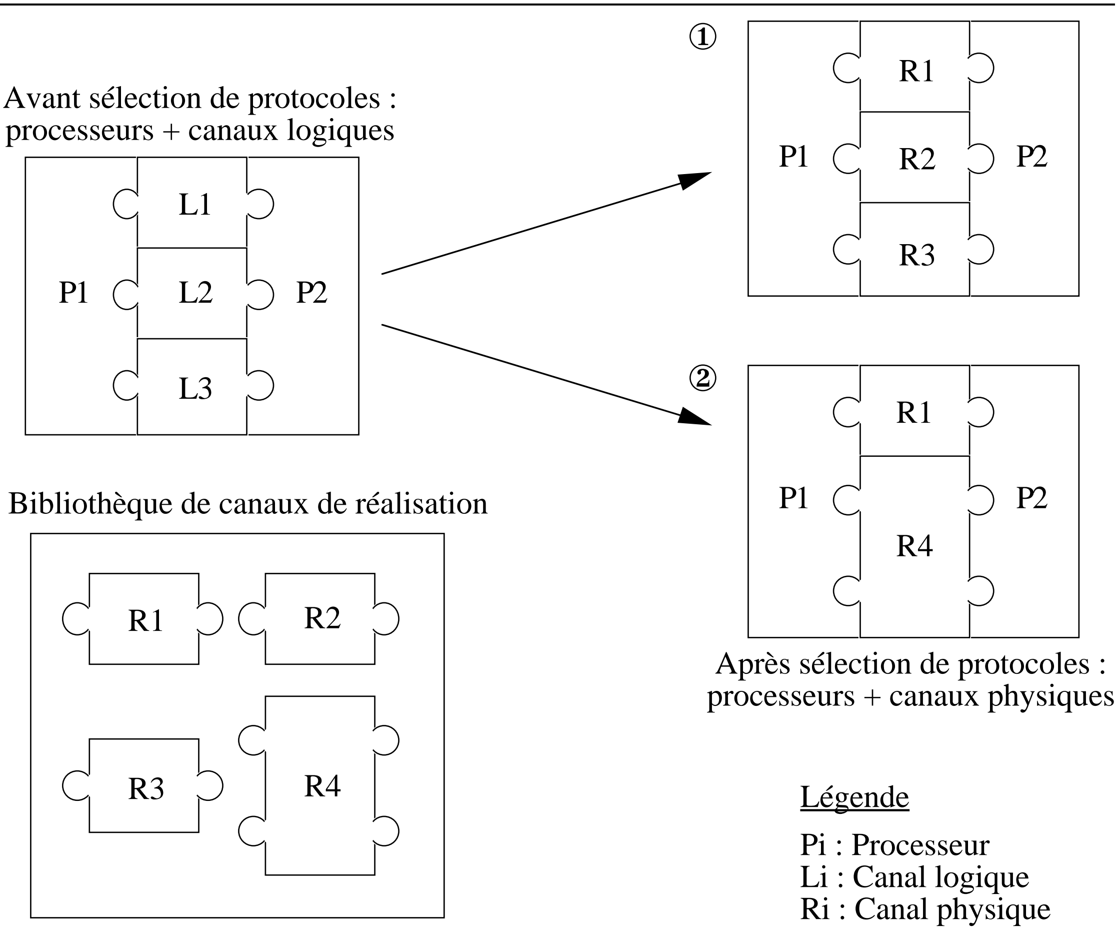


Figure 6.3. *Sélection de composants de communication.*

6.3.2. Algorithmes

Cette section définit les deux algorithmes d'allocation qui sont proposés dans cette thèse. Chaque algorithme correspond à la résolution de l'un des deux cas de figure qui ont été présentés dans la section précédente.

L'objet d'un algorithme d'allocation est de choisir à partir d'une bibliothèque de modèles de communication les unités de communication appropriées, selon une fonction coût déterminée. Les unités de communication allouées sont des canaux qui offrent l'ensemble des services nécessaires aux différents processeurs pour établir des communications. La communication entre différents sous-systèmes (ou processeurs) peut suivre l'un des protocoles de communication (e.g. synchrone, asynchrone, série, ou

parallèle) décrits dans la bibliothèque. Le choix d'une instance de canal parmi ceux décrits dans la bibliothèque va dépendre d'une part du protocole de communication, et d'autre part des performances souhaitées ainsi que de la technologie de réalisation des divers processeurs communicants. L'ensemble de ces caractéristiques peut être pris en compte dans une fonction de coût qui sera à minimiser par l'algorithme d'allocation [DaBI95]. Ceci est similaire à l'allocation d'unités fonctionnelles dans les outils de synthèse de haut niveau [MiLa92]. La plupart des algorithmes d'allocation dans la synthèse de haut niveau peuvent être appliqués afin de résoudre ce problème, moyennant une modification de la fonction coût.

Au cours de cette thèse, les fonctions coût qui sont considérées se basent sur des coefficients. Ces derniers permettent non seulement de pondérer les différents termes d'une fonction coût mais aussi de hiérarchiser les priorités. Ainsi, parmi les facteurs de performance : temps, surface, temps de conception, fiabilité, etc., on n'est pas en mesure de dire a priori lequel est le plus important et donc plus prioritaire que les autres. Ceci dépend de l'application en cours. L'utilisateur peut intervenir en modifiant les coefficients de la fonction coût afin de favoriser les facteurs les plus prioritaires selon son point de vue.

Notations

Soit M_i une unité canal logique qui offre un ensemble de méthodes (services) qui doivent être exécutées par la même unité canal physique. On définit Méthodes(M_i) par $\{m1, m2, ..., m_{n_i}\}$, avec cardinal de M_i noté par Card(M_i) = n_i. Soit M l'ensemble des M_i, donc $M = \{M_1, M_2, ..., M_a\}$. Cet ensemble M représente un réseau logique (conceptuel) de communication.

Soit B la bibliothèque de canaux physiques et qui comporte l'ensemble des B_j avec B_j une unité canal qui offre un ensemble de méthodes. On a alors Méthodes(B_j) = $\{b_1, b_2, ..., b_{c_j}\}$ et Card(B_j) = c_j. Soit A le résultat partiel d'un algorithme d'allocation, A représente l'ensemble des unités canal physiques qui ont été allouées en utilisant un algorithme d'allocation constructif. Soient Var une variable temporaire qui représente la meilleure unité canal allouée pour M_i et Nouveau_Coût son coût. On note par Coût_Total le coût de toutes les instances d'unités canal physiques allouées.

La section qui suit définit les caractéristiques des canaux physiques qui se trouvent dans une bibliothèque utilisée par un algorithme d'allocation.

Caractéristiques des éléments de bibliothèque

Pour chaque unité de communication physique, qui se trouve dans une bibliothèque, l'ensemble de propriétés suivant est associé :

- Chaque canal de la bibliothèque possède un coût intrinsèque noté par Coût (B_j). Ce coût peut dépendre de la complexité, de la surface sur silicium, de la capacité de stockage des messages, de la consommation d'énergie, du coût monétaire, etc.
- Le type de protocole, noté par Protocole, offert par le canal.
- L'ensemble des méthodes (services) offertes par le canal.
- La largeur maximale d'un bus est représentée par MaxLargeurBus(B_j) en bits.
- Le délai d'un transfert, qui représente le temps nécessaire pour effectuer un accès au bus. Il est noté par Délai(B_j) et donné en cycles (impulsions) d'horloge.

Ainsi, MaxDébitBus est défini par :

```
MaxDébitBus = MaxLargeurBus / Délai
```

La section suivante donne l'ensemble de contraintes utilisées lors de l'allocation de canaux physiques à partir d'une bibliothèque.

Contraintes

Soient DébitMin(M_i) une contrainte sur le débit minimal de transfert qui doit être assuré par l'unité canal allouée et DébitMoyen(M_i) une autre contrainte sur le débit moyen de transfert. DébitMin(M_i) et DébitMoyen(M_i) sont spécifiés en bits par cycles (impulsions) d'horloge. Ces deux contraintes peuvent être fournies par le concepteur ou bien calculées par un outil d'estimation.

6.3.2.1. Algorithme SELECT

Comme il a été mentionné précédemment (§ 6.3.1), l'algorithme SELECT permet d'allouer pour une spécification donnée, composée de processeurs communicant à travers des canaux logiques, une instance de canal physique pour réaliser chaque canal logique. Cet algorithme considère uniquement le cas où chaque canal logique est à réaliser par une instance de canal physique.

L'objectif est de choisir l'unité canal physique qui offre le débit de transfert maximum au-dessus du débit moyen de transfert. Cette contrainte sur le débit moyen de transfert peut ne pas être respectée au quel cas la fonction coût va croître.

Fonction coût

La fonction coût qui est à minimiser par l'algorithme d'allocation SELECT prend en considération une unité de communication sélectionnée (B_j) à partir de la bibliothèque et une unité canal M_i à allouer. Cette fonction coût est définie comme suit :

$$\text{Fonction_Coût} = \{\text{DébitMoyen}(M_i)/\text{MaxDébitBus}(B_j)\}^2 * \text{Coût}(B_j) * \text{Card}(B_j)/\text{Card}(M_i)$$

Le terme DébitMoyen(M_i) / MaxDébitBus(B_j) est un coefficient de pondération de cette fonction coût.

Il est à noter que lors d'une allocation une deuxième contrainte sur le débit minimal de transfert, DébitMin, doit être obligatoirement respectée. Autrement dit, l'algorithme SELECT vérifie, avant d'allouer une unité de communication B_j pour réaliser un canal logique M_i, que MaxDébitBus(B_j) est supérieur à DébitMin(M_i). L'autre contrainte sur DébitMoyen est prise en considération au niveau de la fonction coût qui vient d'être présentée.

Pour chaque canal M_i à réaliser, son protocole peut être indéfini (quelconque) ou prédéfini (connu à l'avance) e.g. synchrone. Dans ce dernier cas l'algorithme d'allocation SELECT ne choisit de la bibliothèque de canaux de communication B que les canaux B_j ayant le même protocole que celui de M_i e.g. synchrone.

L'algorithme SELECT est un algorithme glouton. Il considère les canaux logiques un à un et pour chaque canal logique il parcourt les éléments de la bibliothèque de canaux physiques et choisit le meilleur élément qui peut le réaliser. Ce choix est guidé par la fonction coût donnée plus haut.

ALGORITHME SELECT

Entrée : un ensemble de canaux logiques M_i et un ensemble de canaux physique B_j.

Sorties : A qui est l'ensemble des unités canal physiques allouées et Coût_Total leur coût.

```
début
A := {Ø}
Coût_Total := 0
Pour chaque élément Mi de M faire
        Var := Ø ;
        Nouveau_Coût := +∞ ;
        Pour chaque élément Bj de B faire
                MaxDébitBus(Bj) := MaxLargeurBus(Bj) / Délai(Bj)
                Si (Méthodes(Mi) ⊆ Méthodes(Bj)) et (MaxDébitBus(Bj) > DébitMin(Mi))
                alors
                        Coût_actuel := Coût (Bj) ∗ Card(Bj) / Card(Mi) ∗
                                        { DébitMoyen(Mi) / MaxDébitBus(Bj) }^2 ;
                        Si (Coût_actuel < Nouveau_Coût ) alors
                                Nouveau_Coût := Coût_actuel ;
                                Var := Bj ;
                        FinSi
                FinSi
        FinPour
        Si (Var = Ø) alors
                "Aucun canal physique Bj de la bibliothèque B ne peut réaliser le canal
                logique Mi"
        Sinon
                A := A + {Var} ;
                Coût_Total := Coût_Total + Nouveau_Coût ;
                Mi := Var ; /* Rattacher Mi à Var */
        FinSi
FinPour
Fin
```

6.3.2.2. Algorithme ALLOC

Comme il a été mentionné dans § 6.3.1, l'algorithme ALLOC permet d'allouer pour une spécification donnée, composée de processeurs communicant à travers des canaux logiques, une ou plusieurs instances de canaux physiques. Contrairement à l'algorithme SELECT, la particularité de l'algorithme ALLOC est que chaque canal physique peut être alloué pour réaliser un ou plusieurs canaux logiques en même temps. Ceci permet d'augmenter le taux d'utilisation d'un canal physique.

Cet algorithme commence par construire un arbre de décision sur toutes les réalisations possibles. L'arbre de décision sert à énumérer pour chaque canal logique toutes les instances de canaux physiques, de la bibliothèque, qui peuvent être candidates à l'allocation. Les noeuds de cet arbre représentent les canaux logiques et les arcs, sortant de chaque noeud, représentent les instances de canaux physiques qui sont candidates à l'allocation. Les feuilles de l'arbre sont utilisées simplement pour marquer le coût d'une allocation. Chaque chemin de la racine jusqu'à un noeud feuille correspond à une solution possible.

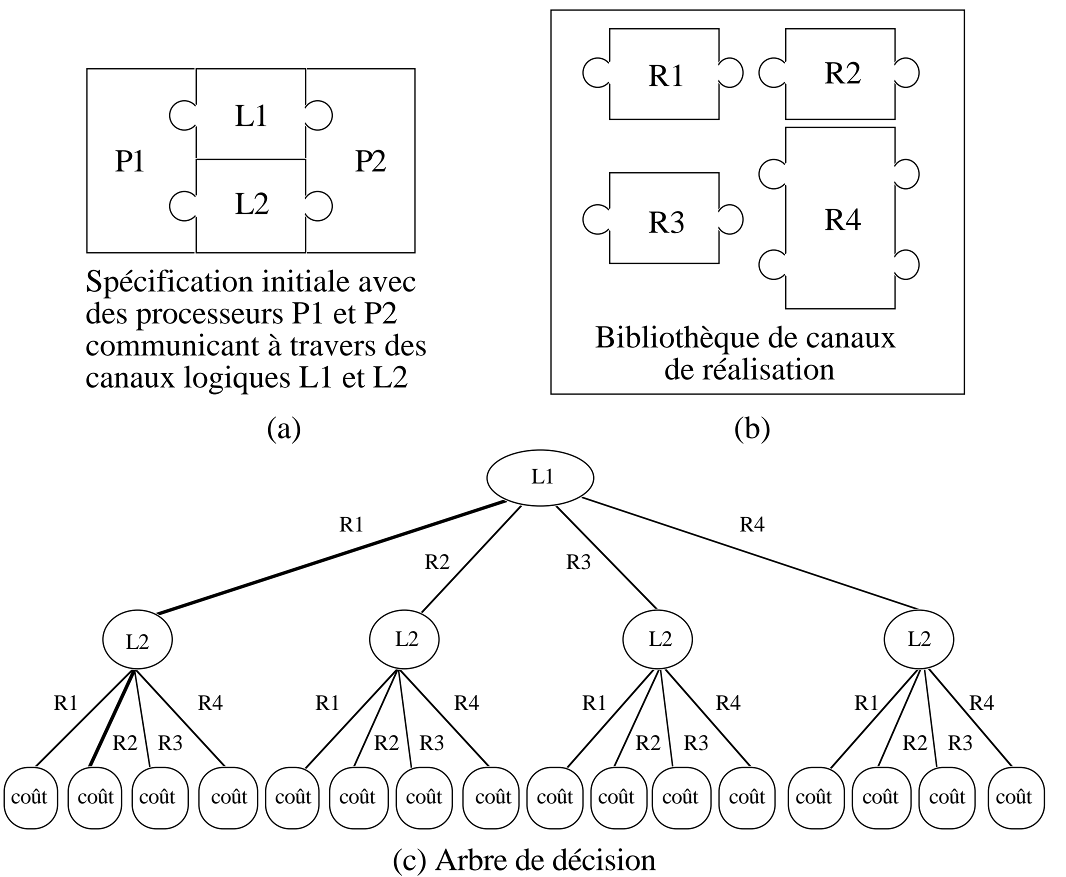


Figure 6.4. *Sélection de composants de communication avec l'algorithme ALLOC.*

Soit l'exemple de la figure 6.4(a) dans lequel deux processeurs P1 et P2 communiquent à travers deux canaux logiques L1 et L2. La figure 6.4(b) donne le contenu de la bibliothèque de canaux physiques. Cette bibliothèque renferme les quatre canaux de réalisation : R1, R2, R3, et R4. L'arbre de décision de toutes les réalisations possibles est présenté dans la figure 6.4(c); cet arbre montre seize solutions possibles. Les canaux logique L1 et L2 peuvent être réalisés par l'un des quatre éléments de la bibliothèque. Dans cet arbre de décision, une des solutions serait de réaliser le canal logique L1 par le canal physique R1 et le canal logique L2 par le canal physique R2. Cette solution est montrée par un chemin, en trait gras, de la racine à l'une des feuilles de l'arbre.

Chaque unité canal logique (M_i) possède un ensemble de contraintes qui portent sur son :

- Protocole de communication noté par Protocole
- DébitMoyen (débit moyen de transfert),
- Débit de communication pour un seul transfert à travers le bus et noté par Débit_de_Pointe.

Il est à noter qu'une contrainte sur DébitMoyen doit être obligatoirement respectée par l'instance de canal qui est à allouer. Par contre, une contrainte sur Débit_de_Pointe peut ne pas être satisfaite, au quel cas il est prévu que la fonction coût associée augmente [NaGa94a]. Ainsi, une définition possible de la fonction coût peut être comme décrit plus bas.

Fonction coût

La fonction coût qui est utilisée par l'algorithme ALLOC est exprimée comme suit:

$$\text{Fonction_Coût} = K_1 \, \text{Coût}(B_j) + K_2 * \sum_i \{\text{Débit_de_Pointe}(M_i) - \text{MaxDébitBus}(B_j)\}^2$$

K_1 et K_2 sont des facteurs de pondération des termes de cette fonction coût. Ces coefficients sont donnés par l'utilisateur et permettent de faire des compromis entre le coût des composants de communication et leurs performances.

La liste d'unités de communication déjà allouées le long d'un chemin dans l'arbre de décision est notée par $\mathfrak{I}$. On a $\mathfrak{I} = \{I_1, I_2, \dots, I_\varphi\}$ avec $1 \leq \varphi \leq$ profondeur de l'arbre. L'instance I_k possède l'ensemble de caractéristiques suivantes :

• Le débit courant du bus noté par DébitBus(I_k) et qui est la somme des valeurs de débit, DébitMoyen, de tous les canaux logiques alloués sur cette instance de canal physique. On suppose que les débits s'additionnent sans occasionner de surcoût dû au partage d'un canal physique.

• Le nombre de communications indépendantes actuelles, noté par ComActuel(I_k).

Définition : Le concept de communications indépendantes veut dire que plusieurs communications entre différents processeurs peuvent avoir lieu en même temps sur le même support physique de communication, tout comme dans le cas d'un réseau informatique de communication entre plusieurs ordinateurs.

Pour une solution réalisable, la condition suivante doit être absolument respectée :

$$\text{MaxDébitBus} \geq \sum_i \text{DébitMoyen}_i$$

Une autre condition, qui exprime le fait qu'un seul transfert de données doit être possible à tout moment, est formulée par :

$$\text{MaxDébitBus} \geq \text{Débit_de_pointe}_i , \forall i$$

Il faut aussi s'assurer que le nombre de canaux logiques réalisés sur la même instance de canal physique ne dépasse pas le nombre de communications indépendantes possibles sur ce même canal physique, noté par ComMax. D'où la condition suivante :

$$\text{ComMax} \geq \text{ComActuel}$$

L'algorithme ALLOC essaye de parcourir l'arbre en profondeur d'abord au moyen de la procédure récursive appelée Parcours. Avec la procédure appelée Fusion, il tente d'utiliser, à chaque fois, l'une des instances préalablement allouées, avant de décider d'allouer une nouvelle instance de canal physique à partir de la bibliothèque.

Pour chaque noeud N de l'arbre de décision, un canal logique, noté par CanalLogique(N), est associé et pour chaque arc A_s sortant une unité canal physique notée par UnitéCommunication(A_s) est également associée. Au cours de l'allocation, l'algorithme ALLOC va essayer de réaliser plusieurs canaux logiques sur la même instance de canal physique. Ces canaux logiques regroupés doivent pouvoir assurer tout de même le débit moyen de transfert qu'ils auraient fourni s'ils étaient réalisés séparément sur plusieurs canaux physiques [GaVa95].

ALGORITHME ALLOC

Entrée : un ensemble de canaux logiques M_i et une bibliothèque B de canaux physiques.
Sorties : A qui est l'ensemble des unités canal physiques allouées et Coût_Total leur coût.

```
début
  Construire l'arbre de décision
  A := {Ø}
  Coût_Total := 0
  Parcours (/, ℑ, Coût_actuel)
  /* Résultat : A et Coût_Total */
Fin
```

La procédure Parcours fait appel à chaque fois, à la procédure Fusion afin d'essayer d'utiliser, l'une des instances préalablement allouées, avant d'allouer une nouvelle instance de canal physique à partir de la bibliothèque B.

```
Procédure Fusion (ListInstance ℑ, Instance CU, Instance Var, Entier Coût_fusion)
/* Entrées : ℑ et CU ; Sorties : Var et Coût_fusion */
début
  Coût_fusion := + ∞
  Pour (chaque instance Ik = CU, Ik ∈ ℑ) faire
   Si  (Protocole(CU) = Protocole(Ik) et (DébitBus(Ik) + DébitMoyen(Mi) ≤ MaxDébitBus(Ik))
        et (ComActuel(Ik) + 1 ≤ ComMax(Ik)) alors
      Si  (Débit_de_Pointe(Mi) > MaxDébitBus(Ik)) alors
         Coût_actuel_fusion := K2 * [Débit_de_Pointe(Mi) - MaxDébitBus(Ik)]^2
      Sinon
         Coût_actuel_fusion := 0
      FinSi
      Si  (Coût_actuel_fusion < Coût_fusion) alors
         Coût_fusion := Coût_actuel_fusion
         Var := Ik
      FinSi
   FinSi
  FinPour
  /* Retourner Var et Coût_fusion */
FinFusion
```

```
Procédure Parcours (Noeud N, ListInstance ℑ, Entier Coût_actuel)
/* Entrée : N ; Sorties : ℑ et Coût_actuel */
début
  Si (N est un noeud feuille) alors
   Si (Coût_actuel < Coût_Total) alors
      A := ℑ
      Coût_Total := Coût_actuel
   FinSi
  Sinon  Var := Ø ; M_i := CanalLogique(N)
   Pour chaque arc A_s partant de N faire
      CU := UnitéCommunication(A_s)
      Fusion (ℑ, CU, Var, Coût_fusion)
      Si (Var ≠ Ø) alors /* fusion réussie */
         Rattacher M_i à Var
         DébitBus(Var) := DébitBus(Var) + DébitMoyen(M_i)
         ComActuel(Var) := ComActuel(Var) + 1
         Coût_actuel := Coût_actuel + Coût_fusion
         Parcours (A_s.noeudsuivant, ℑ, Coût_actuel)
      Sinon /* créer une nouvelle instance de canal */
         Si (DébitMoyen(M_i) ≤ MaxDébitBus(CU)) alors
            Si (Débit_de_Pointe(M_i) > MaxDébitBus(CU)) alors
               Coût_alloc:=K1* Coût(CU) + K2 * [Débit_de_Pointe(M_i)-MaxDébitBus(CU)]^2
            Sinon  /* Pas de contraintes violées */
               Coût _alloc := K1 * Coût(CU)
            FinSi
            Rattacher M_i à CU
            Coût_actuel := Coût_actuel + Coût_alloc
            DébitBus(CU) := DébitMoyen(M_i)
            ComActuel(CU) := ComActuel(CU) + 1
            ℑ := ℑ + {CU}
            Parcours (A_s.noeudsuivant, ℑ, Coût_actuel)
         Sinon  /* Pas de solution */
            Coût_actuel := + ∞
         FinSi
      FinSi
   FinPour
  FinSi
FinParcours
```

6.4. La synthèse d'interfaces

La synthèse d'interfaces consiste à la fois en des transformations [OBBI93] et des optimisations [FiKu93] de la description résultat de l'étape d'allocation de protocoles. Les transformations permettent d'adapter l'interface des sous-systèmes communicants; les optimisations permettent de prendre en considération les contraintes de temps (*timing*) tels que le temps nécessaire pour établir une connexion, ou bien le temps de communication lui-même qui est fonction du débit des données. La synthèse d'interfaces tient compte de la taille des données à transférer et de la largeur du bus de données. Ainsi, selon ces deux paramètres, une même donnée peut être acheminée soit en série, soit en parallèle (si la largeur du bus de données le permet) tout en gardant le même protocole de communication préalablement alloué. Par exemple, un entier sur 16 bits peut être acheminé en série par un bus de données de 8 bits, et en parallèle par un bus de données de 16 bits. En fait, comme il a été vu dans la section définissant les caractéristiques des éléments de bibliothèque (§ 6.3.2), chaque unité de communication utilisée lors de la phase d'allocation de protocoles possède une largeur maximale de bus, notée par MaxLargeurBus. C'est au niveau de la phase de synthèse d'interfaces que cette largeur de bus sera fixée. Il s'agit donc d'une opération de raffinement de l'interface des processeurs communicant à travers un ou plusieurs canaux physiques de communication.

6.4.1. Formulation du problème

L'étape de synthèse d'interfaces remplace toutes les unités de communication en distribuant le protocole de communication entre les sous-systèmes communicants et éventuellement des contrôleurs spécifiques de communication. Ces contrôleurs sont choisis à partir d'une bibliothèque de réalisation des canaux. Cette sélection s'effectue en prenant des considérations de (1) débit de transfert des données, de (2) capacité de stockage des files d'attente, et (3) et du nombre de lignes de donnée et de contrôle. Cette activité consiste à transposer une structure logique de communication sur une structure physique. Le résultat obtenu à la suite d'une synthèse d'interfaces est un ensemble d'unités communicantes à travers des bus. Parmi ces unités, il peut y avoir des composants tels que les contrôleurs de communication ou les arbitres de bus. Chaque unité représente soit un processeur abstrait soit un composant de la bibliothèque de canaux. La figure 6.5 présente des alternatives possibles à la synthèse d'interfaces. La première alternative génère un bus de huit bits et un contrôleur de FIFO; la deuxième

alternative génère un bus de seize bits avec une FIFO de plus grande taille. D'autres travaux [BoKa87] se sont penchés sur le problème d'interface entre des composants standards ayant des protocoles incompatibles. Deux approches de génération d'interfaces avec une conversion automatique de protocoles sont présentées dans [LiVe94, NaGa95]. Dans ces approches, les unités à faire communiquer peuvent avoir des interfaces fixes.

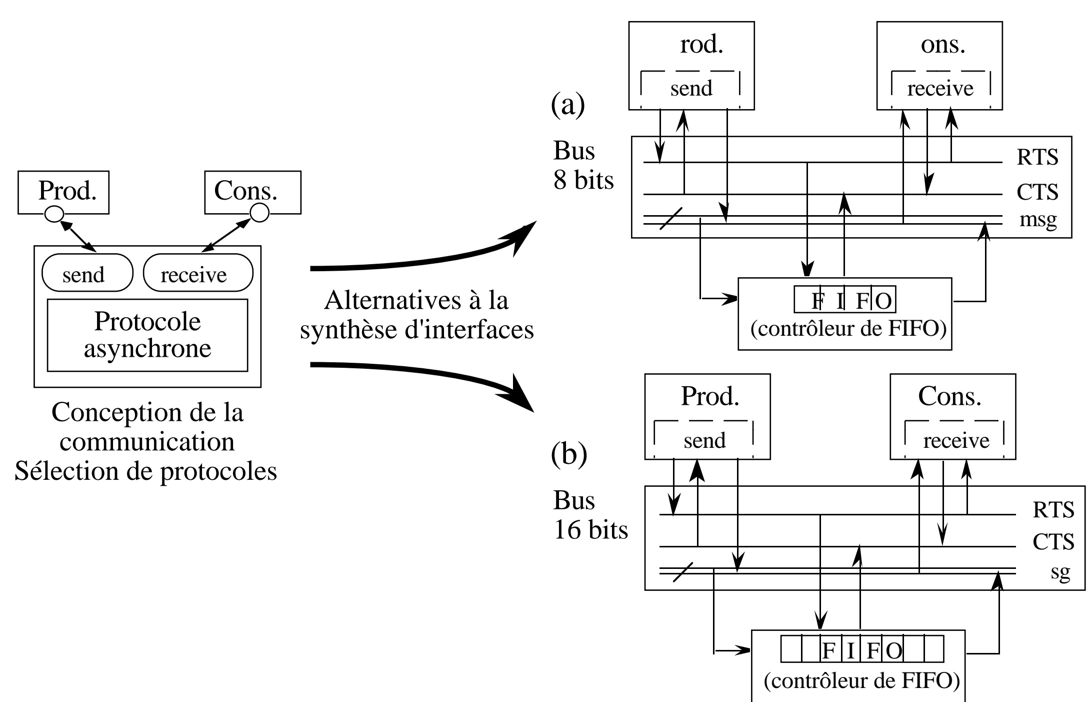


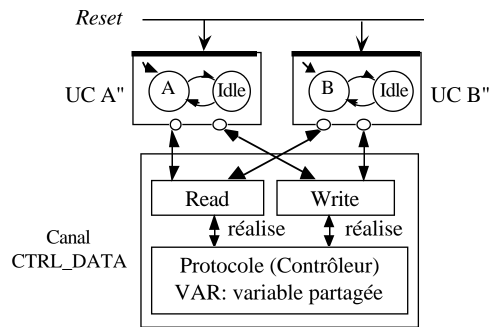
Figure 6.5. *Alternatives à la synthèse d'interfaces.*

6.4.2. La primitive Map

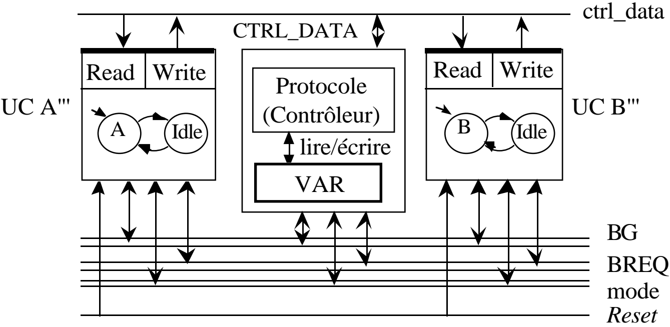
Principe: Cette primitive a pour objet de transformer des sous-systèmes communicants en sous-systèmes interconnectés. Elle réalise donc la synthèse (adaptation) d'interfaces de ces sous-systèmes, notamment ceux générés à la suite du découpage par la primitive *Cut*. Pour cela une expansion en ligne des méthodes (procédures) appelées à distance est effectuée. La primitive *Map* remplace les accès aux canaux par des signaux d'interface (ports d'entrée/sortie).

Exemple 2: Soit la machine parallèle AB de la figure 6.6(a), après application de la primitive *Map*, on obtient les sous-systèmes interconnectés A''', B''', et CTRL_DATA (voir figure 6.6(b)). Le canal CTRL_DATA de la figure 6.6(a) a été transformé en une unité de conception dont le rôle est de contrôler l'accès à la ressource partagée. Dans cet

exemple *BG*, *BREQ* et *mode* représentent des signaux de contrôle. Le signal *ctrl_data* est relatif à la donnée partagée “VAR”.



(a) Machine AB avant l'application de Map



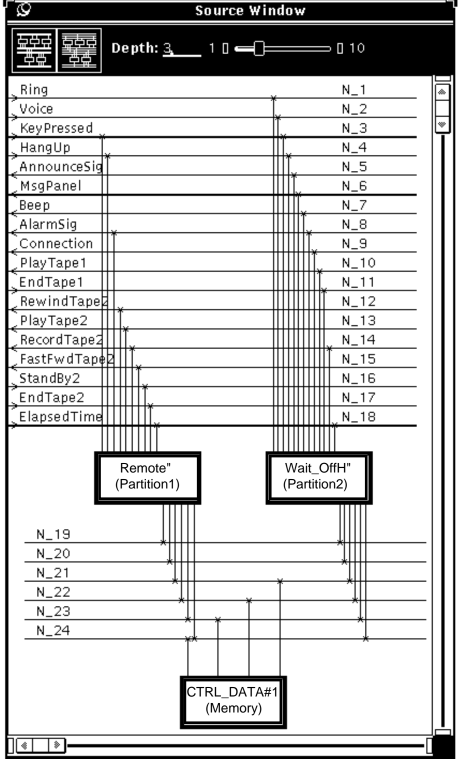
(b) Machine AB après l'application de Map

Figure 6.6. *Machine AB avant et après l'application de Map.*

C'est le sous-système CTRL_DATA qui va contrôler l'accès à la donnée partagée “VAR”, en autorisant à chaque fois un (ou deux) processus pour lire ou écrire dans la variable “VAR”. Par exemple, si la donnée partagée est dans le mode EREW (lectures et écritures exclusives), alors à tout moment un seul processus peut accéder à la donnée en lecture ou bien en écriture (voir exemple de canal en annexe A.2).

La primitive Map a été programmée puis intégrée dans l'outil PARTIF [BIOB95]. Pour l'application du répondeur téléphonique (présentée au chapitre 5, §5.5), la figure 6.7 montre la partie contrôle de ce répondeur téléphonique, avant et après l'application de la primitive Map. Dans la figure 6.7(a), N_21 représente le canal de communication *“CTRL_DATA#1”* qui gère l'accès aux ressources partagées. Le résultat après l'application de Map(*CTRL_DATA#1*) est un ensemble de deux processeurs qui sont interconnectés avec un composant qui protège l'accès à une mémoire partagée. Comme il est présenté dans la figure 6.7(b), il s'agit des deux partitions obtenues à la suite d'un découpage et du contrôleur de communication *CTRL_DATA#1*. L'algorithme de la primitive *Map* est présenté en annexe C.6.

(a) Répondeur téléphonique avant l'application de Map



(b) Répondeur téléphonique après l'application de Map

Figure 6.7. *Répondeur téléphonique avant et après l'application de Map.*

6.5. Exemple

Dans cette section les résultats des algorithmes d'allocation et de synthèse d'interfaces, appliqués sur un exemple classique de producteur/consommateur, seront présentés. Il s'agit d'une application comportant un émetteur de messages et un récepteur qui renvoi les acquittements. Le système est composé de deux processus d'émission et de réception et de deux canaux logiques utilisés pour véhiculer des communications uni-directionnelles. Ce modèle correspond au modèle de communication utilisé dans le langage SDL. La figure 6.8 montre une représentation de l'entrée à la synthèse de communication. A chacun des deux canaux logiques, l'ensemble de contraintes suivant sera associé :

• DébitMoyen(CU_émetteur) = 12 bits / cycle d'horloge,

• DébitMin(CU_émetteur) = 10 bits / cycle d'horloge,

• DébitMoyen(CU_récepteur) = 4 bits / cycle d'horloge,

• DébitMin(CU_récepteur) = 2 bits / cycle d'horloge,

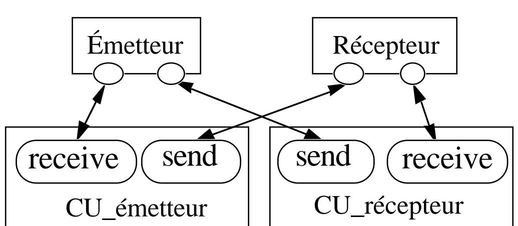


Figure 6.8. *Système Émetteur/Récepteur.*

6.5.1. Application de l'algorithme SELECT

La bibliothèque d'unités de communication qui est considérée avant d'appliquer l'algorithme SELECT comporte :

(1) Un canal ayant un protocole rendez-vous (RDV),

(2) Un canal ayant un protocole asynchrone et possédant une file d'attente de messages.

Les caractéristiques de ces unités de communication sont présentées dans la figure 6.9. Chacune de ces unités est définie par (1) son type de protocole, (2) la largeur maximale de son bus, (3) son délai de transfert, exprimé en cycles d'horloge, et (4) son coût intrinsèque. Le délai de transfert représente le temps nécessaire pour effectuer un accès au bus. Le dernier paramètre relatif au coût intrinsèque peut dépendre de plusieurs facteurs tels que la complexité de l'unité de communication ou sa capacité de stockage des messages.

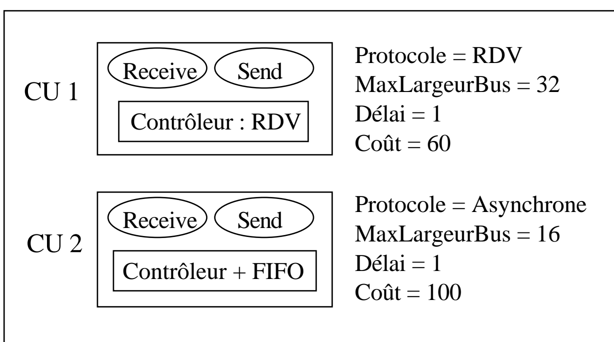


Figure 6.9. *Bibliothèque des unités de communication pour l'algorithme SELECT.*

Les valeurs de Coût_Total sont obtenues en appliquant la fonction coût qui a été présentée pour l'algorithme SELECT. Cet algorithme SELECT essaye de trouver la solution réalisable avec le coût minimum. Les quatre possibilités d'allocation de canaux sont décrites ci-dessous et présentées dans la figure 6.10 :

(a) - CU_émetteur et CU_récepteur réalisés chacun avec un protocole RDV (CU 1). Le coût total dans ce cas est Coût_Total = 10.

(b) - CU_émetteur et CU_récepteur réalisés chacun avec un protocole asynchrone et possédant sa propre file de messages (CU 2). Le coût total dans ce cas est Coût_Total = 62.

(c) - CU_émetteur avec le protocole RDV (CU 1) et CU_récepteur avec le protocole asynchrone (CU 2). Le coût total dans ce cas est Coût_Total = 14.

(d) - CU_émetteur avec le protocole asynchrone (CU 2) et CU_récepteur avec le protocole RDV (CU 1). Le coût total dans ce cas est Coût_Total = 57.

Le résultat de l'algorithme SELECT est donc l'alternative (a) qui est retenue parce qu'elle possède le coût minimum.

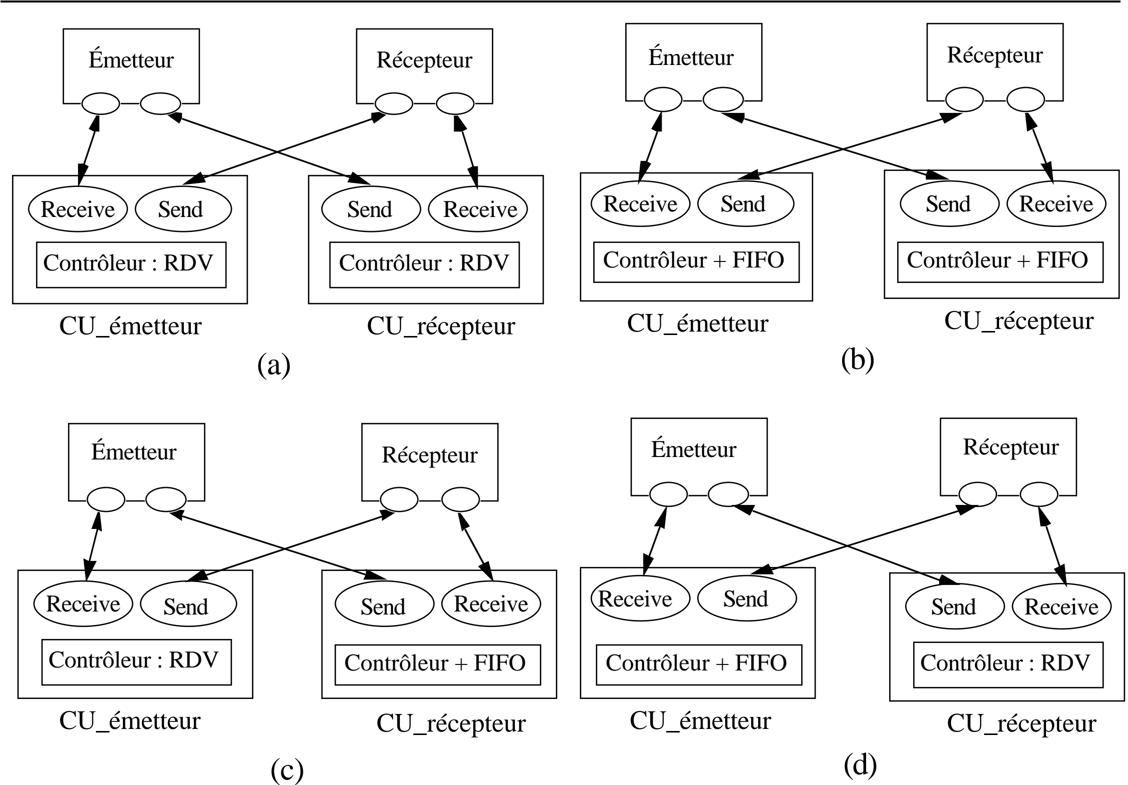


Figure 6.10. *Alternatives d'allocation par l'algorithme SELECT.*

L'étape de synthèse d'interfaces génère automatiquement le système présenté dans la figure 6.11. Cette étape réalise une expansion en ligne des procédures de communication et transpose une interface générique pré-définie sur l'émetteur et le récepteur. La largeur de chaque bus dépend du débit de transfert de données. Elle est dans tous les cas inférieure à la largeur MaxLargeurBus qui est décrite dans la bibliothèque.

```
Bus 1   ===================================  Données
        -----------------------------------  Lignes de contrôle

        Interface           Interface
        Émetteur            Récepteur
        Interface           Interface

Bus 2   ===================================  Données
        -----------------------------------  Lignes de contrôle
```

Figure 6.11. *Synthèse d'interfaces du résultat d'allocation par l'algorithme SELECT.*

6.5.2. Application de l'algorithme ALLOC

L'intérêt d'appliquer l'algorithme ALLOC est de pouvoir allouer une seule instance de la bibliothèque pour réaliser les deux canaux logiques CU_émetteur et CU_récepteur. Cette solution sera trouvée uniquement avec l'algorithme ALLOC. A priori, cette solution peut réduire la taille des bus (en utilisant un seul bus) avec éventuellement une légère perte de performances. Une approche qui détermine la largeur d'un bus qui va réaliser un groupe de canaux a été présentée dans [FiKu93].

Pour l'algorithme ALLOC, en plus des contraintes déjà prises pour l'algorithme SELECT, d'autres contraintes concernant les débits de pointe seront considérées. Ainsi, l'ensemble de contraintes suivant est à considérer :

- DébitMoyen(CU_émetteur) = 12 bits / cycle d'horloge,
- Débit_de_Pointe(CU_émetteur) = 18 bits / cycle d'horloge,
- DébitMoyen(CU_récepteur) = 4 bits / cycle d'horloge,
- Débit_de_Pointe(CU_récepteur) = 8 bits / cycle d'horloge,
- Protocole(CU_émetteur) = Protocole(CU_récepteur) = quelconque.

Dans la fonction coût qui est utlisée par l'algorithme ALLOC, les valeurs choisies pour les coefficients K_1 et K_2 sont 1 et 10 respectivement. La raison qui a conduit à ce choix est de privilégier les performances au dépend du coût des composants de communication. La bibliothèque d'unités physiques de communication comprend la liste suivante :

(1) Un canal ayant un protocole RDV, qui ne sauvegarde pas les messages et qui ne gère qu'une seule communication à la fois,

(2) Un canal ayant un protocole RDV, qui ne sauvegarde pas les messages et qui peut gérer deux RDV à la fois,

(3) Un canal ayant un protocole asynchrone et qui gère une seule file d'attente de messages,

(4) Un canal ayant un protocole asynchrone et qui gère deux files d'attente de messages.

Les différentes unités de cette bibliothèque sont décrites dans la figure 6.12. Chacune de ces unités est définie par (1) son type de protocole, (2) le débit maximal de son bus, (3) le nombre de communications indépendantes possibles, noté par ComMax, et (4) son coût intrinsèque. Rappelons que le débit maximal d'un bus est la valeur obtenue en divisant la largeur maximale du bus par le délai de transfert qui est nécessaire pour effectuer un accès au bus (§ 6.3.2).

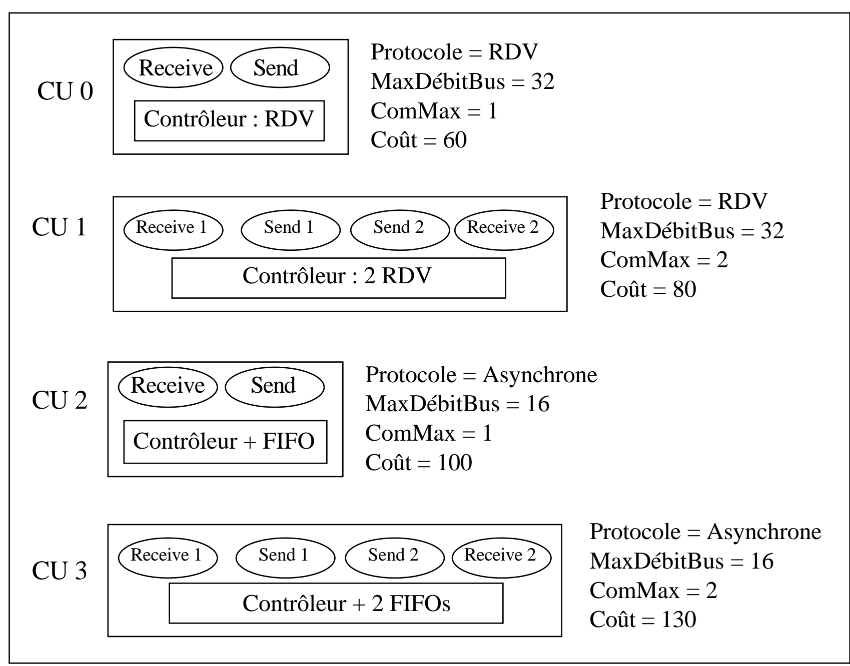


Figure 6.12. *Bibliothèque des unités de communication pour l'algorithme ALLOC.*

Comme il a été dit plus haut (§ 6.3.2.2), l'algorithme ALLOC commence par construire l'arbre de décision qui sert à énumérer toutes les réalisations possibles. La figure 6.13 présente l'arbre de décision relatif à ce système producteur/consommateur. Toutes les alternatives d'allocation sont évaluées par l'algorithme ALLOC en fonction de leurs coûts. Certaines feuilles de l'arbre indiquent le coût Avant/Après fusion. Par exemple, 300/170 signifie que le coût avant fusion est égal à 300 et que le coût après fusion est égal à 170.

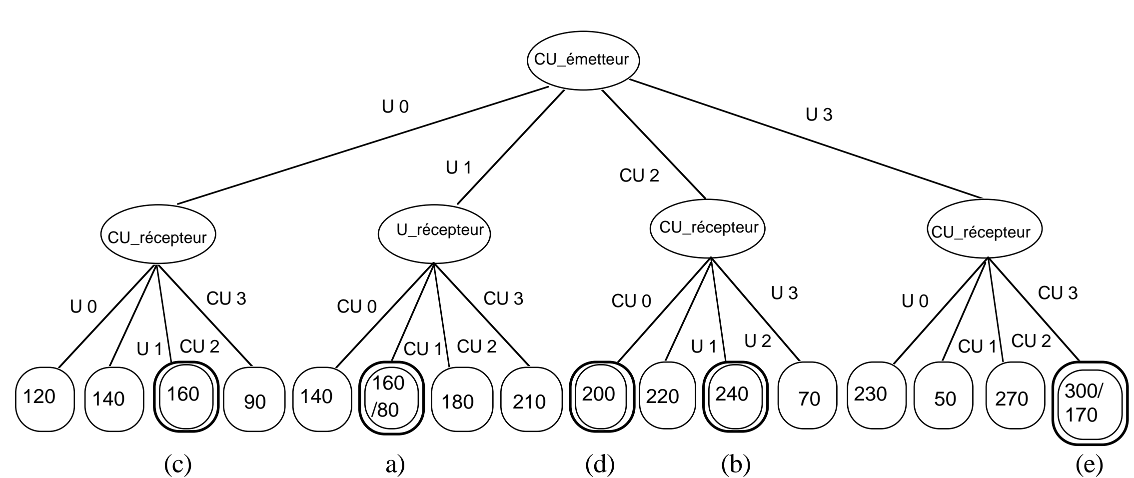


Figure 6.13. *Arbre de décision obtenu par l'algorithme ALLOC.*

Ce système producteur/consommateur présente dix-huit réalisations possibles. Comme indiqué dans la figure 6.13, chaque feuille de l'arbre représente une solution. Cependant, en raison de la possibilité de réaliser plusieurs canaux logiques par un seul canal physique (caractéristique de l'algorithme ALLOC) il y a, en plus des seize allocations un à un (un canal physique qui réalise un canal logique), deux autres alternatives. La première permet de réaliser les deux canaux logiques par le canal physique “CU 1”. La deuxième permet de réaliser les deux canaux logiques par le seul canal physique “CU 3”. Toutes ses alternatives d'allocation (dix-huit) ne seront pas énumérées explicitement. A titre d'exemple, cinq alternatives possibles d'allocation sont décrites ci-dessous et présentées dans la figure 6.14 (elles sont indiquées et marquées par deux cercles dans la figure 6.13) :

(a) - CU_émetteur et CU_récepteur réalisés par le protocole RDV (CU 1) qui peut gérer deux communications à la fois. Il est important de préciser qu'il n'est nécessaire d'allouer qu'une seule instance de “CU 1”. Le coût total dans ce cas est Coût_Total = 80.

(b) - CU_émetteur et CU_récepteur réalisés chacun avec un protocole asynchrone et possédant sa propre file de messages (CU 2). Le coût total dans ce cas est Coût_Total = 240.

(c) - CU_émetteur avec le protocole RDV (CU 0) et CU_récepteur avec le protocole asynchrone ayant une file d'attente (CU 2). Le coût total dans ce cas est Coût_Total = 160.

(d) - CU_émetteur avec le protocole asynchrone ayant une file d'attente (CU 2) et CU_récepteur avec le protocole RDV (CU 0). Le coût total dans ce cas est Coût_Total = 200.

(e) - CU_émetteur et CU_récepteur réalisés avec le protocole asynchrone (CU 3) qui gère deux files de messages. Dans ce cas, une seule instance de “CU 3” est nécessaire. Le coût total est alors Coût_Total = 170.

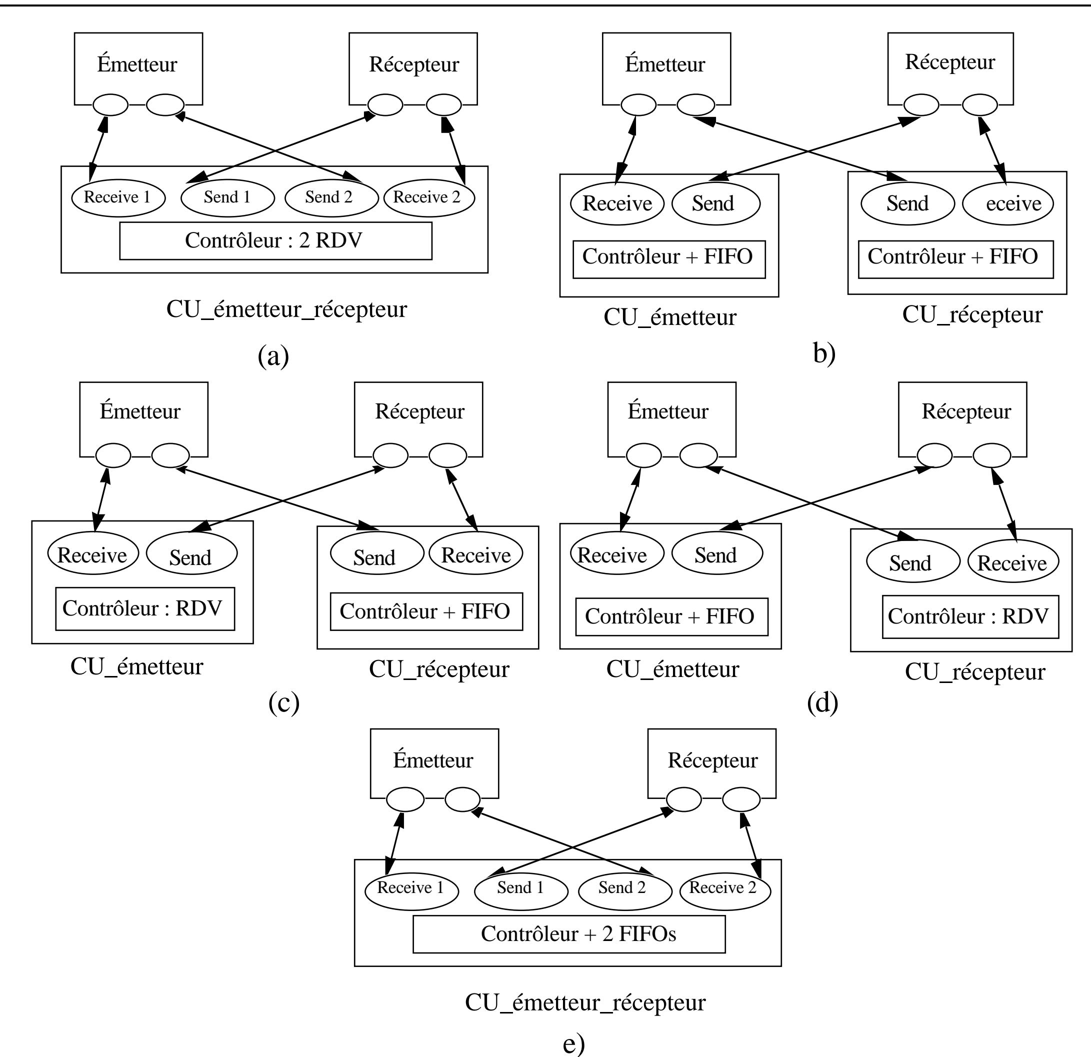


Figure 6.14. *Alternatives d'allocation par l'algorithme ALLOC.*

Le Coût_Total obtenu pour chaque alternative est calculé en utilisant la fonction coût de l'algorithme ALLOC. Cet algorithme choisit l'alternative avec le plus faible coût d'allocation. Par conséquent la solution (a) sera retenue. Les deux canaux logiques sont alors réalisés sur le même support physique qui est un bus bi-directionnel et sans capacité de stockage de messages.

La phase de synthèse d'interfaces transpose une interface générique pré-définie sur l'émetteur et le récepteur. Cette interface est fixée en fonction du débit de transfert des données. La synthèse d'interfaces réalise aussi une expansion en ligne des appels de procédures (méthodes) de communication. Le résultat est présenté dans la figure 6.15.

Émetteur
Interface
Récepteur
Interface
Bus
Données
Lignes de contrôle

Figure 6.15. *Synthèse d'interfaces du résultat d'allocation par l'algorithme ALLOC.*

6.5.3. Comparaison des algorithmes d'allocation

Dans cette section, les deux algorithmes d'allocation SELECT et ALLOC seront comparés dans le cadre de cet exemple.

Il faut tout d'abord préciser que l'algorithme SELECT est une version simplifiée de l'algorithme ALLOC, même si les fonctions coût utilisées sont différentes. En effet, ces deux algorithmes trouvent la même solution si les alternatives de réalisation des deux canaux logiques par un seul canal physique sont écartées. La bibliothèque qui est utilisée par l'algorithme ALLOC comporte deux canaux supplémentaires (appelés CU 1 et CU 3) par rapport à celle utilisée par l'algorithme SELECT. Le canal “CU 1” de la bibliothèque utilisée par SELECT est appelé “CU 0” dans la bibliothèque utilisée par ALLOC. Les deux canaux communs aux deux bibliothèques présentent les mêmes caractéristiques sur le type de protocole, le débit maximum, et le coût intrinsèque. Enfin, dans la liste de contraintes qui est considérée pour chaque canal logique, les deux algorithmes utilisent le débit moyen mais en plus l'algorithme SELECT utilise le débit minimum alors que l'algorithme ALLOC utilise le débit de pointe.

6.6. Conclusion

Dans ce chapitre l'activité de synthèse de la communication a été présentée comme un problème d'allocation. Afin de prendre en considération tous les aspects de synthèse de la communication, à la fois la sélection de protocoles et la synthèse d'interfaces ont été combinées. Ceci a permis de développer, dans le cadre de cette thèse, une approche globale de la synthèse de la communication. Pour l'étape de sélection de protocoles de communication, deux algorithmes d'allocation de composants de communication, à partir d'une bibliothèque prévue à cet effet, ont été présentés. Le premier algorithme permet de sélectionner pour chaque canal logique une unité canal physique dédiée. Le deuxième algorithme essaye d'allouer une unité canal physique pour plusieurs canaux logiques. Ce multiplexage de canaux logiques se base sur un arbre de décision. Il sert à réduire le nombre d'instances de canaux physiques et donc le coût de réalisation en augmentant le taux d'utilisation des canaux physiques.

Dans ce chapitre, l'étape de synthèse d'interfaces, qui vient juste après la sélection des protocoles de communication, a été également présentée. Rappelons que la synthèse d'interfaces sert à adapter l'interface de chaque unité communicante pour que les communications prévues à travers les différents protocoles puissent avoir lieu. Cette étape de synthèse d'interfaces est réalisée automatiquement en appliquant la primitive Map sur une description en Solar. La phase suivante consiste à générer du code C ou VHDL pour chaque composant. Cette génération est automatique puisque le comportement et l'interface de chaque composant sont clairement définis. Plus particulièrement, les interfaces sont déjà fixées et les protocoles de communication sont bien établis.

Enfin, les algorithmes de sélection de protocoles et l'étape de synthèse d'interfaces ont été illustrés à travers une application au problème classique du producteur/consommateur.

D'autres travaux peuvent venir compléter ceux qui ont été réalisés au cours de cette thèse. Les travaux futurs peuvent inclure ce qui suit :

• Une bonne caractérisation des unités canal qui font partie d'une bibliothèque d'éléments de communication.

• La recherche d'une bonne fonction coût pouvant s'appliquer à la sélection de protocoles complexes, par exemple pour choisir entre un Ethernet et un FDDI.

5.1. Introduction

Le découpage sert à distribuer les fonctionnalités d'une spécification système sur un ensemble de processeurs. Étant donné un graphe de processus (ou de tâches), le but est de transposer les différents processus sur un graphe de processeurs (voir figure 5.1).

Figure 5.1. *Le découpage au niveau système.*

Il s'agit de prendre les décisions qui permettent de répondre aux questions suivantes :

- Combien de processeurs faut-il pour exécuter une description donnée ?
- Dans quelle technologie (logicielle ou matérielle) doit être réalisé chaque processeur ?
- Sur quels processeurs doit s'exécuter chaque fonction d'une spécification ?

Il est à noter qu'une fonction (après son découpage) peut être affectée à plusieurs processeurs et que plusieurs fonctions peuvent être affectées à un seul processeur. Par exemple, dans la figure 5.1, la fonction F3 est affectée aux processeurs P2 et P3. De même les fonctions F2, une partie de F3 et une partie de F4 sont affectées au processeur P2. Dans la figure 5.1, C1 et C2 désignent des unités de communication, par exemple des arbitres de bus ou bien des micro-contrôleurs.

L'approche de synthèse au niveau système qui est présentée dans la figure 5.2, comporte deux phases principales qui sont le découpage et la résolution des protocoles de communication (synthèse de communication). Le premier point fera l'objet de ce chapitre; le second point sera détaillé dans le chapitre suivant (chapitre 6). La première phase (découpage) découpe une spécification en un ensemble de partitions qui seront transposées sur une architecture cible. La spécification à découper représente un ensemble de processus qui peuvent être hiérarchiques et parallèles. Ces processus communiquent entre eux, soit via des variables partagées, soit par passage de messages. La stratégie de découpage suivie est interactive en raison de la complexité des systèmes à découper. Le problème du découpage est un problème NP-complet, il est donc difficile de réaliser une approche automatique. La deuxième phase (synthèse de communication) permet de définir des protocoles de communication entre les partitions. Elle permet aussi d'adapter les interfaces des différentes partitions afin de communiquer à travers les protocoles de communication préalablement choisis.

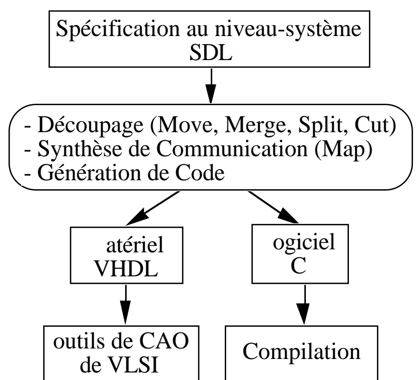


Figure 5.2. *Approche de synthèse.*

Actuellement, il existe trois cultures différentes des chercheurs qui travaillent sur le découpage :

1) Les premiers sont de culture logicielle [ErHe93]. Ils commencent donc à partir d'une spécification entièrement logicielle et cherchent à faire de la migration de code vers le matériel. Les parties critiques d'un système sont identifiées puis affectées à une réalisation matérielle.

2) Les deuxièmes sont de culture matérielle [GuDM93]. Ils partent d'une spécification initiale entièrement en matériel. Par la suite, les parties non critiques sont identifiées afin de les affecter à une réalisation logicielle, ce qui permet de réduire le coût de réalisation.

3) Les troisièmes sont de culture système (concepteurs de systèmes). Ils ne se limitent pas à un type particulier de spécification en entrée. Dans ce type d'approche, les différentes parties d'une spécification sont affectées à une réalisation (logicielle ou matérielle) qui satisfait les contraintes de conception (e.g. temps de réponse) [GaVa95].

L'affectation d'un composant vers une réalisation logicielle ou matérielle se base sur une fonction coût à minimiser. L'approche présentée dans la suite de ce chapitre utilise une architecture généralisée multiprocesseur. Les méthodes de découpage proposées dans [ErHe93, GuDM93] supposent une architecture composée d'un seul processeur, un seul circuit intégré (ASIC), une mémoire, et un bus. D'autres méthodes [SrBr91, ThAd93] prennent en considération une architecture composée d'un seul processeur avec plusieurs circuits intégrés (ASICs).

Le découpage qui sera présenté dans ce chapitre est orienté vers des systèmes de contrôle complexes (e.g. un système de télécommunication par satellite). Par opposition, il existe des systèmes dominés par le flot de données tels que les applications de traitement de signal [VaVa93]. Le résultat d'un découpage est un ensemble de partitions interconnectées qui communiquent à travers des interfaces bien définies. Un tel découpage peut résulter en un ensemble de sous-systèmes distribués s'exécutant en parallèle sur différentes machines. Chaque sous-système peut être réalisé dans une technologie matérielle ou logicielle.

L'approche de découpage adoptée est interactive. Ce découpage suit une méthode semi-automatique; il s'agit de combiner à la fois la conception manuelle et la conception automatique. La décision de ne pas développer une approche automatique a été prise en raison de la difficulté de réaliser des estimations fiables indépendantes du domaine d'application. Une approche automatique doit se baser impérativement sur des estimations précises or certains critères comme la surface d'un circuit ou la fiabilité d'une réalisation sont difficiles à estimer au niveau système. L'approche qui est proposée suppose que le concepteur commence la synthèse à partir d'une spécification initiale des fonctions d'un système et ayant en tête une solution d'architecture. L'environnement offre au concepteur un ensemble de primitives de transformation d'une spécification, en suivant un schéma de raffinement incrémental, en un modèle distribué qui concorde avec le modèle d'architecture. Toutes les transformations de raffinement sont réalisées de manière automatique. Ceci permet une très grande accélération du procédé de conception. D'un autre côté, toutes les décisions de raffinement sont prises par le concepteur qui peut utiliser son savoir faire et son expérience pour converger vers une solution efficace.

Dans l'approche qui est proposée, le découpage est réalisé à l'aide de quatre primitives de base qui sont appelées : *Move*, *Merge*, *Split*, et *Cut*. Le modèle de description a été décrit dans le chapitre 3. Il s'agit d'une extension au modèle couramment utilisé des machines d'états finis (MEF). Ce modèle a été choisi parce qu'il s'adapte bien à la représentation des systèmes de contrôle complexes. En effet, les extensions au modèle MEF permettent de représenter la hiérarchie et les actions parallèles.

Une méthode d'utilisation du système interactif de découpage appelé PARTIF (PARTItioning of extended Finite state machines) sera présentée dans la section 2. L'accent sera mis tout particulièrement sur le compte rendu du système PARTIF. Ce résultat sert à évaluer la qualité du découpage et permet aussi de comparer différentes réalisations d'une spécification. Les primitives utilisées pour réaliser le découpage seront détaillées dans la section 3. La réalisation de l'outil PARTIF sera décrite dans la section suivante. Enfin, les résultats du découpage d'un exemple d'application seront présentés juste avant la conclusion.

5.2. L'outil PARTIF

Cette section définit le modèle de découpage utilisé dans le système PARTIF et décrit la méthode permettant de réaliser ce découpage. Le système PARTIF permet au concepteur de modifier le parallélisme et la hiérarchie d'une spécification afin de trouver un compromis surface/vitesse.

Définition 1: Étant donnée une spécification comportementale et au niveau système qui suit le modèle des MEFs étendues, le découpage consiste à partitionner cette description en plusieurs sous-systèmes communicants, ayant un contrôle distribué.

Comme il a été dit précédemment, l'un des problèmes clef du découpage au niveau système est de trouver une bonne fonction coût qui sert à estimer une réalisation. En effet, il est difficile d'estimer la surface du circuit et le temps d'exécution d'une partie ou de toute la conception à ce niveau élevé d'abstraction. L'idée qui est suivie dans cette thèse est de fournir au concepteur un environnement de type boîte à outils avec un ensemble de primitives de transformation et de découpage de descriptions. En même temps, un rapport des résultats permet de donner des indications sur les incidences des choix réalisés par le concepteur. Ceci permet d'aider le concepteur à explorer différentes alternatives de découpage. Ensuite, le concepteur peut choisir l'alternative qui correspond le mieux à ses contraintes (e.g. rapidité du circuit).

Exemple 1: Soit une machine hiérarchique ABC, la figure 5.3 montre trois partitionnements possibles de cette machine. Dans la figure 5.3(a) qui est tirée du modèle de Drusinsky [DrHa89], il existe un contrôle distribué entre les niveaux de la hiérarchie. Par exemple, le processus ABC contient tout le contrôle qui lui permet de séquencer A et BC. Ces deux processus sont mis à feu (selon la terminologie utilisée dans les réseaux de Petri) quand il est nécessaire. Lorsque l'un d'eux a terminé, il redonne le contrôle à ABC qui va alors décider le prochain état. De manière similaire, le processus BC contrôle la sous-hiérarchie contenant B et C. Dans cette réalisation chaque processus aura ses propres ressources, sans possibilité de partage entre processus. En plus, cette solution entraîne un surcoût dû à la communication hiérarchique.

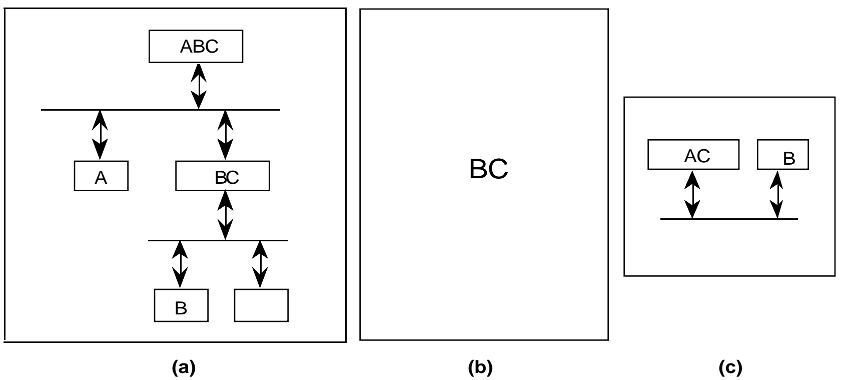


Figure 5.3. *Partitionnements possibles :*
(a) contrôle distribué, (b) contrôle centralisé, (c) compromis.

La figure 5.3(b) présente une solution où la hiérarchie a été éliminée par une fusion des états, donc le contrôle est centralisé dans une seule machine (MEF). Cette approche de mise à plat a pour conséquence une explosion combinatoire du nombre d'états, ce qui la rend inefficace du point de vue des performances pour les grands circuits. La figure 5.3(c) présente un compromis entre les deux précédentes dans le cas où on a des machines parallèles. En fusionnant deux processus (A et C) dans une seule machine (MEF AC), on réduit le surcoût causé par le contrôle distribué, tout en évitant l'explosion combinatoire du nombre d'états.

Soient les symboles U, + et ∗ qui désignent respectivement l'union, la somme et le produit. La table 5.1 présente un tableau comparatif entre la fusion de deux machines séquentielles et celle de deux machines parallèles.

	Opérateurs	Etats	E / S
MEF Séq	∪	+	∪
MEF //	+	*	∪

Table 5.1. *Résultats de fusion de MEFs.*

La fusion de machines parallèles peut générer une explosion combinatoire du nombre d'états. Il s'agit du produit d'automates. Par exemple, la fusion de deux machines ayant chacune n états, permet d'obtenir une machine ayant n^2 états. Cependant, il existe des méthodes pour réduire ce nombre d'états, en éliminant les états redondants [Wolf90b].

L'exemple suivant explique l'intérêt du découpage réalisé dans l'exemple 1.

Exemple 2: Considérons le cas où on voudrait trouver la meilleure combinaison des processus de la figure 5.3 pour réduire le nombre d'états et le nombre d'additionneurs et de soustracteurs. On suppose que la fonction coût admet uniquement deux paramètres qui sont : le nombre d'opérateurs et le nombre d'états. L'estimation du coût de chaque processus est donnée dans la table 5.2. Par exemple, l'état B contient huit états et nécessite deux additionneurs.

MEF	Etats	Opérateurs
A	4	+ -
B	8	+ +
C	10	+ -

Table 5.2. *Estimation du coût de chaque processus de la figure 5.3.*

La table 5.3 donne les résultats obtenus en essayant toutes les combinaisons de fusion des processus A, B et C. D'après cette fonction coût simplifiée, la meilleure solution de découpage est obtenue en fusionnant les processus A et C (ligne 3 de la table 5.3). En effet, dans ce cas on a le nombre minimum d'états (4+8+10=22 seulement) et on utilise le nombre minimum d'opérateurs (4 seulement). On a besoin d'un additionneur et d'un soustracteur pour les processus A et C; ceci est dû au fait qu'il est possible de partager les ressources entre ces processus qui sont exclusifs. De même on a besoin de deux autres additionneurs pour le processus B.

		Etats	# Opérateurs
1.	Pas de Fusion	22	6 (+-,++,+-)
2.	Fusion A, B	22	5 (++-,+-)
3.	Fusion A, C	22	4 (+-,++)
4.	Fusion B, C	84	6 (+++-,+-)
5.	Fusion A, B, C	84	4 (+++-)

Table 5.3. *Résultats de découpage du système de la figure 5.3.*

5.2.1. Le principe

Le découpage comportemental repose sur des opérations de fusion et de découpage de MEFs. La figure 5.4 décrit le schéma de fonctionnement du système PARTIF. La spécification en entrée de PARTIF est une MEF étendue où coexistent des machines parallèles et des machines séquentielles. La sortie de PARTIF est une réalisation distribuée dans laquelle chaque machine est séquentielle. Ces différentes machines (partitions) seront traitées par la suite par les outils de synthèse appropriés. L'entrée et la sortie de PARTIF sont des descriptions en Solar. Après le découpage, les parties à réaliser en logiciel seront traduite en langage C, et les parties à réaliser en matériel seront traduite en VHDL. Les autres entrées au système PARTIF sont des contraintes de conception et une bibliothèque de modèles de communication. Un rapport d'estimation du coût du découpage est fournit en sortie après chaque opération de découpage. Les contraintes de conception, la bibliothèque de modèles de communication ainsi que le rapport d'estimation donné en sortie de PARTIF seront détaillés dans la suite de cette section.

Figure 5.4. *Le système PARTIF.*

5.2.1.1. Les contraintes de conception

Les contraintes considérées dans le système PARTIF représentent l'ensemble de règles régissant les conditions d'applicabilité des primitives de découpage. Le concepteur peut imposer des contraintes telles que le nombre maximum d'états à permettre dans une partition, ou bien le nombre maximum d'entrées/sorties dans l'interface d'une partition. Toute violation de ces contraintes est signalée par PARTIF. Par exemple, si le nombre d'états dans une partition dépasse la limite fixée, ou si le nombre d'entrées/sorties entre deux partitions dépasse le seuil fixé, le système informe alors le concepteur pour lui suggérer de réaliser des opérations de découpage ou de fusion. Une autre contrainte possible concerne le nombre maximum de partitions à réaliser. En effet, on considère une architecture cible distribuée et chaque partition représente un processeur (matériel ou logiciel). Par conséquent, si on a idée de l'architecture sur laquelle on veut transposer la description découpée, il faut que le nombre de partitions obtenues soit inférieur ou égal au nombre de processeurs de l'architecture cible. Supposons qu'on dispose d'une architecture qui comporte deux processeurs standards et une carte pouvant contenir un processeur matériel (ASIC ou FPGA); dans ce cas il ne faut pas que le découpage génère plus que trois partitions (deux à réaliser en logiciel et une en matériel par un ASIC ou un FPGA).

5.2.1.2. La bibliothèque de modèles de communication

Cette bibliothèque renferme un ensemble de protocoles de communication ainsi que leurs descriptions. L'utilité de ces éléments de communication (canaux de Solar) vient du fait que le découpage peut générer de la communication. En effet, si on découpe une MEF en plusieurs partitions qui se partagent une donnée (variable), on aura besoin d'insérer un protocole qui permet de gérer les conflits d'accès à la ressource partagée.

5.2.1.3. Estimation de coût

L'un des problèmes majeur du découpage au niveau système consiste à estimer le coût d'une solution donnée. En effet, les critères peuvent être, par exemple, l'efficacité, la fiabilité, la maintenabilité, la portabilité, ou la facilité d'utilisation. Certains de ces critères peuvent comporter une liste d'autres critères. Par exemple, l'efficacité peut inclure la rapidité, le coût monétaire, la consommation d'énergie, la surface ou le volume. Pour chaque critère, une valeur métrique doit être associée de même qu'un facteur qui sert à pondérer les critères. La valeur de chaque facteur dépend de la technologie utilisée et du domaine d'application. Par exemple, la fiabilité peut être le facteur majeur pour une application du domaine spatial ou en général un système concerné par la sécurité de la vie humaine. D'autre part, pour d'autres systèmes tels que les applications multimédia qui sont portables, le coût et la consommation d'énergie sont les plus importants facteurs. Il est donc difficile de définir une fonction d'estimation qui soit réaliste même pour un domaine d'application spécifique. PARTIF fournit une évaluation statistique du coût de chaque partition générée. Ce rapport est donné sous forme de table contenant, pour chaque partition les caractéristiques suivantes :

1 - Les entrées/sorties,
2 - Les variables partagées entre les partitions,
3 - La surface (pour une réalisation en matériel),
4 - Les opérateurs,
5 - Les variables locales,
6 - Le nombre d'états (pour une réalisation en logiciel),
7 - Le temps d'exécution (en nombre de cycles).

Un tel rapport va constituer la clef pour le réglage et l'optimisation du découpage réalisé. Il sert aussi à comparer différentes alternatives de découpage.

L'analyse et la modification d'un système nécessitent d'examiner deux types de paramètres : (A) les états de la machine et (B) les connexions entre ces machines [WoTa91]. Pour le système PARTIF, les caractéristiques 1 et 2 précédemment mentionnées concernent l'interface (interconnexions) d'une partition (à réaliser en logiciel ou en matériel). Les autres caractéristiques (3, 4, et 5) sont liées à la surface du circuit correspondant à une partition à réaliser en matériel. Les interconnexions génèrent un surcoût dû à l'utilisation d'unités d'entrée/sortie et d'unités de communication. Pour une réalisation matérielle d'une partition, les opérateurs utilisés vont engendrer l'utilisation d'unités fonctionnelles (e.g. unité arithmétique et logique) et les variables locales impliquent l'utilisation de registres ou de mémoires. La caractéristique 7 (temps d'exécution) concerne la rapidité d'exécution d'une partition sans compter le temps de communication qui peut être plus ou moins lent.

5.2.2. La méthode de découpage

Partant d'une spécification au niveau système, décrivant des MEFs étendues, cette approche consiste à permettre au concepteur de construire ses partitions. Ce dernier effectue alors, une distribution des machines dans des blocs différents. Chaque bloc (partition) correspond à un processeur dans l'architecture cible. Rappelons que le découpage génère des sous-systèmes communicants. Ces derniers doivent réaliser la spécification de départ, en prenant en même temps des considérations de surface et de vitesse. On suppose que le concepteur a en tête une architecture cible qui peut être multi-processeurs. Les différents processeurs sont soit logiciels (e.g. microprocesseur standard), soit matériels (e.g. ASIC ou FPGA). Cette architecture cible peut être déjà existante (fixe), ou bien elle reste à définir.

La méthode de découpage utilisée dans le système PARTIF est présentée dans ce qui suit. Cette méthode se base sur un découpage interactif [BIOB95]. Le découpage est réalisé à l'aide de quatre primitives de base : *Move*, *Merge*, *Split*, et *Cut*. La primitive *Move* sert à déplacer des machines à travers la hiérarchie d'une machine d'états finis étendue. La primitive *Merge* fusionne des machines séquentielles pour produire une seule machine. La primitive *Split* découpe une machine séquentielle et produit plusieurs machines en parallèle. La primitive *Cut* transforme une machine parallèle en un ensemble de machines interconnectées ou communicantes. La figure 5.5 résume la méthode de découpage du système PARTIF.

Figure 5.5. *Méthode inhérente à PARTIF.*

Une fois que le concepteur a choisi une primitive, une analyse de faisabilité est effectuée par le système. Cette analyse consiste à tester la faisabilité du découpage compte tenu des contraintes fixées par le concepteur, telles que le nombre maximum d'états dans une partition, le nombre maximum d'interconnexions dans l'interface d'une partition, et le nombre de partitions. L'analyse de faisabilité applique la primitive choisie tout en maintenant la description initiale. Ensuite, une évaluation est effectuée pour voir si les contraintes imposées par le concepteur sont satisfaites.

PARTIF utilise une évaluation qui estime le bénéfice de fusion ou de découpage des processus. Les critères à considérer sont les sept caractéristiques mentionnées dans la section précédente. Par exemple, si une variable (ou un tableau) va être partagée par deux processus, ceci va rendre ces derniers de bons candidats à être fusionnés. De manière similaire, si des processus peuvent partager les mêmes ressources, ceci va rendre favorable leur fusion. En général, pour aller plus vite il faudrait augmenter le parallélisme en utilisant plus de processeurs. Pour diminuer la surface de la réalisation, il faudrait diminuer le parallélisme, donc séquentialiser les fonctions du système dans un nombre minimum de processeurs. Les deux cas extrêmes sont d'une part une solution avec un

maximum de processeurs qui revient plus cher (en coût et en surface) mais qui est plus rapide; d'autre part une solution monoprocesseur qui revient moins cher mais qui est plus lente car toute l'application s'exécute en séquentiel. Par ailleurs, la distribution du calcul entre plusieurs processeurs peut entraîner des surcoûts dus à la communication entre ces processeurs. Cette communication peut augmenter le coût de réalisation et diminuer la vitesse d'exécution ce qui peut réduire l'intérêt d'une telle solution parallèle (multiprocesseurs).

Après l'analyse de faisabilité, si le découpage est accepté, on applique la primitive choisie. Ensuite, selon la qualité du découpage réalisé (en se basant sur le coût estimé), le concepteur va décider soit de modifier le découpage (ou bien le refaire) pour trouver une meilleure découpe à moindre coût, soit de l'accepter vu les bonnes performances qu'il induit. Le concepteur peut aussi annuler l'application de la dernière primitive. Cette approche itérative d'application des primitives s'arrête lorsque toutes les contraintes sont satisfaites.

5.2.3. L'affectation logiciel/matériel

Le nombre de partitions générées par le découpage représente le nombre de processeurs de l'architecture cible. La nouvelle version en cours de préparation [MaBI95] devra permettre l'exécution de plusieurs partitions sur un seul processeur. Les partitions qui font partie du chemin critique de l'application et demandent un temps de réponse court seront réalisées en matériel. Les autres partitions seront réalisées en logiciel afin de réduire le coût monétaire de réalisation. Généralement, dans un système on a des parties qui ne peuvent être réalisées qu'en matériel. On peut citer comme exemple les parties d'interface. D'autres parties du système ne peuvent être réalisées qu'en logiciel. On peut citer à titre d'exemple les parties dont la fonction doit évoluer, donc il est envisageable de les modifier. Un algorithme de compression ou de codage de données peut être réalisé en logiciel pour avoir plus de souplesse au cas où on veut améliorer cet algorithme ou le remplacer par un autre. Finalement, d'autres parties du système peuvent être réalisées indifféremment en logiciel ou en matériel. Le concepteur va alors essayer de trouver un compromis surface/vitesse d'exécution pour ces dernières parties. Il peut alors essayer différentes alternatives où il affecte chacune de ces parties une fois à une réalisation logicielle et une fois à une réalisation matérielle. En se basant sur les résultats des estimations, il pourra décider la technologie de réalisation.

5.3. Les primitives de découpage

Dans cette partie, les primitives de découpage utilisées par le système PARTIF seront présentées. Un découpage peut être réalisé à l'aide des quatre primitives : *Move*, *Merge*, *Split*, et *Cut* . Pour chacune des primitives, on présentera le principe suivi d'un exemple portant sur son application. L'algorithme de chaque primitive sera présenté dans l'annexe C.

5.3.1. La primitive Move

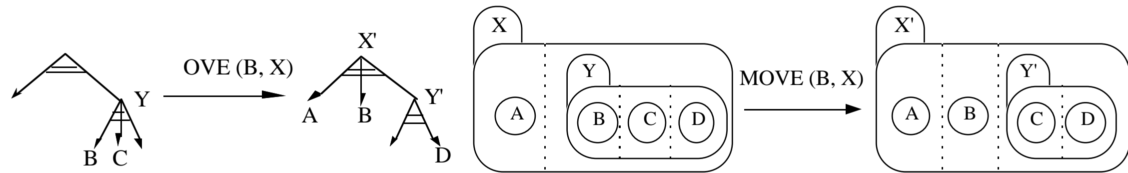
Principe: Cette primitive sert à déplacer une machine (état ou table d'états) à travers la hiérarchie d'une MEF étendue. Elle peut être ainsi utilisée pour la migration de code d'une réalisation logicielle à une réalisation matérielle. L'application de cette primitive est généralement une étape transitoire avant l'application de l'une des autres primitives *Merge*, *Split* ou *Cut*. On peut citer l'exemple où on veut fusionner deux machines de niveaux hiérarchiques différents, dans ce cas on applique la primitive *Move* avant d'appliquer la primitive *Merge*. Cette primitive *Merge* ne peut fusionner que des machines de même niveau dans une hiérarchie.

Dans ce qui suit on définit le chemin entre deux machines de niveaux hiérarchiques différents.

Définition 2: Un chemin entre deux machines prises dans une représentation hiérarchique, est la liste des machines (noeuds intermédiaires) constituant le plus court chemin entre ces deux machines. Ce chemin est dit séquentiel (respectivement parallèle), si toutes ses machines sont séquentielles (respectivement parallèles). Les bornes ne sont pas incluses dans le chemin.

Notation: *Move* (P, Q), place la machine P dans la machine Q. Ce placement, n'est possible que dans les deux cas suivants : (1) Le chemin de P à Q est séquentiel. (2) le chemin de P à Q est parallèle. On suppose que P n'est pas un état de Q. L'algorithme de la primitive *Move* est présenté en annexe C.2.

Exemple 3: Soit l'exemple suivant dans lequel le chemin entre B et X est parallèle (voir figure 5.6).



(a) Représentation hiérarchique (b) Représentation par un StateChart

Figure 5.6. *Exemple d'application de la primitive Move.*

5.3.2. La primitive Merge

Principe: Cette primitive a pour objectif la fusion de deux machines séquentielles pour produire une seule machine. La fusion de deux machines permet le partage de ressources. En fait, certains registres utilisés pour les variables locales de chaque machine peuvent être partagés. Il en est de même pour les unités fonctionnelles (Additionneur, Multiplieur, etc.) utilisées pour exécuter les différentes opérations [BIOJ94]. Dans tous les cas, la fusion peut engendrer une meilleure utilisation des ressources matérielles, ce qui veut dire une réduction de la surface au niveau du circuit résultant.

La primitive *Merge* a deux paramètres qui sont, soit deux tables d'états (*StateTables*), soit une table d'états (*StateTable)* et un état (*State*). Le résultat de la fusion est toujours une table d'états.

L'application de cette primitive sur deux machines A et B (notée par *Merge*(A, B)) donne une nouvelle machine AB.

Exemple 4: Soit une machine séquentielle ABCDE constituée de deux tables d'états (*StateTables*) AB et CD et d'un état (*State*) E (figure 5.7(a)). L'application de la primitive *Merge* sur AB et CD donne une nouvelle machine ABCDE' qui ne contient plus que deux machines ABCD et E (figure 5.7(b)). L'application de nouveau de la primitive *Merge* sur ces deux machines, produit une seule machine ABCDE" (figure 5.7(c)).

Figure 5.7. *Fusion de machines.*

Dans ce cas l'application de cette primitive a réduit la hauteur de l'arbre représentant la MEF étendue. Le même résultat peut être obtenu en appliquant successivement *Merge*(CD, E), puis *Merge*(AB, CDE). L'algorithme de la primitive *Merge* est présenté en annexe C.3.

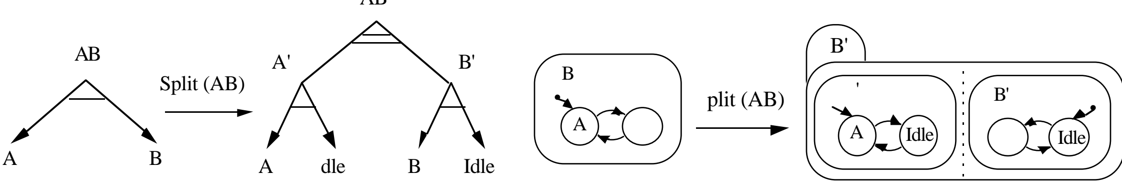
5.3.3. La primitive Split

Principe: Cette primitive opère sur une machine séquentielle. Elle a pour objet de transformer cette machine en une machine parallèle. La parallélisation est rendue possible par l'ajout de signaux de contrôle.

Dans ce qui suit, un modèle abstrait de MEFs est utilisé. Ce modèle permet de manipuler hiérarchiquement des MEFs indépendamment de leurs états. Il introduit pour chaque machine un état “repos” et un signal de contrôle. Une MEF est dite au repos si aucun de ses états n'est actif. Dans le cas d'une MEF hiérarchique, l'état de repos implique que toutes les MEFs de la machine hiérarchique sont à leur état de repos nommé par “idle”. Une MEF hiérarchique est dite active lorsque l'une de ses composantes est active.

Il est important d'insister sur l'intérêt de ce modèle, étant donné qu'il permet de représenter le contrôle de processus, tel qu'arrêter un processus et commencer un autre processus. D'autre part, la valeur des signaux de contrôle donne l'état global du système.

Exemple 5: Soit une machine séquentielle AB (*StateTable*) contenant deux machines A et B. *Split*(AB) a pour effet de transformer la machine AB en une machine parallèle AB' contenant deux états A' et B' (voir figure 5.8). La machine A' (respectivement B') contient deux processus séquentiels qui sont A (respectivement B) et *Idle* (voir figure 5.9). Dans ce cas, deux nouveaux signaux de contrôle qui sont ctrl_A et ctrl_B ont été générés automatiquement.



(a) Représentation hiérarchique (b) Représentation par un StateChart

Figure 5.8. *Découpage d'une machine AB.*

Lorsque AB' est activée (figure 5.8), les deux machines A' et B' deviennent actives. Cependant, l'une d'entre elles va être dans son état actif (A ou B) et l'autre dans son état *Idle*. Par exemple, si la machine A' est dans l'état actif (A), la machine B' est dans l'état de repos (*Idle*) et on a ctrl_A qui est égale à un et ctrl_B qui est égale à zéro. Lorsque ctrl_A passe de un à zéro, la machine A' passe dans l'état (inactif) *Idle* et le contrôle est donné à la machine B' qui passe de son état *Idle* à l'état actif B.

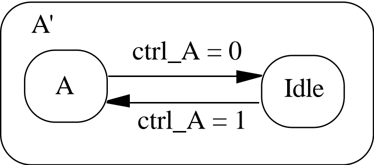


Figure 5.9. *La machine A'.*

Chaque fois qu'il résulte, après évaluation d'une machine, que les contraintes imposées sur la surface et/ou sur le nombre d'interconnexions ne sont pas satisfaites, on a recours à la primitive *Split*. L'intérêt de ce découpage réside dans le fait que les machines séparées vont être dans des partitions différentes. Dans le cas d'une machine parallèle, un état de repos “idle” est rajouté à chacun de ses composants. En effet, pour sortir d'une machine parallèle, il faudrait sortir de chacun de ses composants, ce qui

revient à le mettre dans un état de repos. L'exemple suivant sert à montrer le résultat de l'application de la primitive *Split* sur une machine contenant des états parallèles.

Exemple 6: Soit une machine séquentielle ABC constituée de deux machines AB et C. La machine AB est parallèle, alors que la machine C est séquentielle (voir figure 5.10).

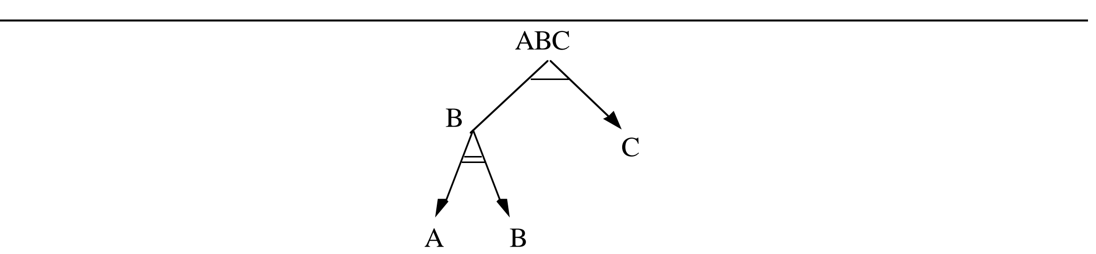
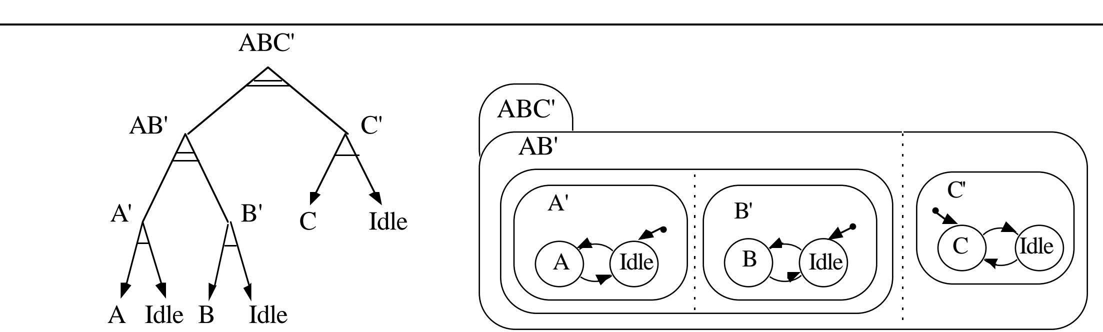


Figure 5.10. *La machine ABC à découper.*

L'application de la primitive *Split*, sur cette machine ABC, a pour effet de la transformer en une machine parallèle ABC' (voir figure 5.11).

Supposons que l'état d'entrée par défaut de la machine ABC est l'état C, alors l'entrée par défaut pour A' ainsi que B' est l'état inactif Idle, alors que l'entrée par défaut de la machine C' sera l'état actif C. Il est important de préciser que la machine ABC' traduit exactement le comportement de la machine ABC de départ. L'algorithme de la primitive *Split* est présenté en annexe C.4.



(a) Représentation hiérarchique (b) Représentation par un StateChart

Figure 5.11. *La machine ABC après le découpage.*

5.3.4. La primitive Cut

Principe: Cette primitive a pour objet de transformer une machine parallèle en un ensemble de machines (unités de conception) interconnectées. Dans le cas où les

machines de départ utilisent des données partagées, les machines résultantes, après l'application de *Cut*, seront communicantes via des canaux. On associe un canal à chaque donnée partagée. Il s'agit d'un canal abstrait pouvant accepter plusieurs types de données. Le protocole d'accès à ce canal dépend du type d'accès à la donnée partagée.

Une donnée peut être dans l'un des cas suivants :

• En lecture exclusive et en écriture exclusive (EREW),
• Partagée en lecture avec écriture exclusive (CREW),
• Partagée en lecture et partagée en écriture (CRCW).

Dans le cas où la bibliothèque de canaux contient un canal pour chaque protocole (EREW, CREW, CRCW) alors à chaque utilisation d'une donnée partagée, on fera appel au canal correspondant pour pouvoir utiliser la donnée. Le choix du canal est guidé par l'utilisateur, en fonction de la donnée partagée (cas EREW, CREW, ou CRCW). Ce canal suit le modèle de canal vu dans le chapitre 3. Cependant, un canal qui gère l'accès à une donnée partagée offre deux services seulement, à travers les méthodes *READ* et *WRITE*. Si la donnée est à accéder en lecture (respectivement écriture), la machine qui l'utilise fait appel à la méthode *READ* (respectivement *WRITE*) du canal, à l'aide du constructeur *CUCALL* de Solar [JeOB94]. Cette communication, modélisée en Solar [OBBI93], se fait par appel de procédures à distance (RPC). Les canaux correspondants aux cas EREW et CREW différent uniquement par leurs contrôleurs (voir exemple de canal Solar pour le cas EREW en annexe A.2). Ainsi la primitive *Cut* permet de résoudre le problème des données partagées par des processus en parallèle (problème de l'entrelacement). Pour le cas de CRCW, c'est le contrôleur (fonction de résolution) du canal (correspondant à la donnée partagée) qui va gérer le cas de deux écritures en parallèles sur une même donnée. Il est à noter que la primitive *Cut* n'est applicable que sur le premier niveau de la hiérarchie, donc la MEF étendue relative à une seule unité de conception.

Exemple 7: Soit la machine AB (*State*) qui est parallèle de la figure 5.12. Dans cet exemple, on suppose que les machines A' et B' partagent une donnée commune (variable “VAR”). Après l'application de la primitive *Cut* sur AB, on génère deux blocs (unités de conception) A" et B" qui sont en parallèle et qui communiquent via un canal CTRL_DATA (voir figure 5.13). L'accès à cette donnée “VAR” se fera désormais via le canal CTRL_DATA. C'est ce canal qui contient la déclaration ainsi que le protocole d'accès à la donnée partagée.

Figure 5.12. *Machine AB* avant l'application du *Cut.*

Le canal CTRL_DATA contrôle l'accès à la donnée commune, en autorisant à chaque fois un ou plusieurs processus (selon le type d'accès à la donnée) pour lire ou écrire dans la variable “VAR”. La primitive *Cut* définit donc les interfaces entre les sous-systèmes (e.g. interface avec le signal de remise à zéro : *Reset*) ainsi que les protocoles de communication (figure 5.13). Après le découpage, chaque sous-système ne communiquera avec les autres qu'au moyen de ses entrées/sorties (canaux et signaux). L'algorithme de la primitive *Cut* est présenté en annexe C.5.

Figure 5.13. *Machine AB après l'application de Cut(AB).*

5.4. Réalisation

L'outil PARTIF a été programmé en langage C++ avec environ 12000 lignes de code. Cet outil a été développé sur une station de travail Sun (Sparc 20). Deux versions sont actuellement disponibles; la première permet l'application des primitives à travers des commandes sous le système d'exploitation UNIX. La deuxième version est graphique et permet d'utiliser l'outil à travers des menus déroulants [BIMa96]. Dans cette deuxième version l'application des primitives s'effectue simplement en sélectionnant avec la souris

une des primitives, ensuite l'une des MEFs étendues de la description. La figure 5.14 montre le menu de base de l'outil PARTIF dans sa version graphique.

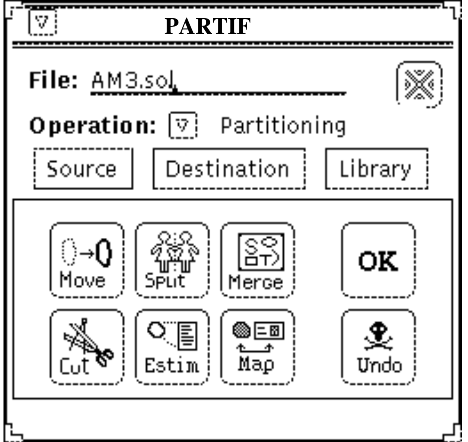


Figure 5.14. *Menu graphique de l'outil PARTIF.*

L'utilisateur a la possibilité de visualiser la description graphique ou textuelle avant d'appliquer une primitive. Cette description source est visible dans une fenêtre appelée “Source”. L'application d'une primitive entraîne la création d'une nouvelle description en Solar. Une fois qu'une primitive a été choisie, le résultat apparaît dans une fenêtre appelée “Destination”. Afin de valider l'opération, l'utilisateur peut confirmer en appuyant sur le bouton “OK” dans le menu. Ainsi, la description qui apparaissait dans la fenêtre Destination devient à son tour dans la fenêtre Source, et l'utilisateur peut alors poursuivre de nouveau l'application des primitives. A tout moment, l'utilisateur peut annuler l'application de la dernière primitive (même si elle a été confirmée) en appuyant sur le bouton “Undo”. Ceci permet de revenir sur des choix effectués au préalable et d'explorer d'autres alternatives de transformation et de découpage. On peut même revenir à la description initiale en appuyant plusieurs fois successivement sur le bouton “Undo”.

Dans la figure 5.14, le bouton appelé “Estim” qui apparaît dans le menu sert à générer un tableau des résultats d'estimations. Ces résultats sont destinés à aider l'utilisateur dans sa décision de poursuivre ou non l'application d'une séquence de primitives. L'utilisateur peut appliquer une séquence de primitives et voir le résultat du découpage à travers la description générée et visualiser le tableau des résultats d'estimations. Par la suite, il peut appliquer l'opération d'annulation (“Undo”) un certain nombre de fois, afin de repartir avec un autre point de départ (la description initiale ou bien une description sur laquelle des primitives ont été appliquées). Cet outil permet aussi de visualiser l'historique des opérations réalisées sur la description initiale.

Les autres possibilités de cet outil graphique sont la génération automatique de codes C et VHDL. A partir de la description en Solar d'une application, il est possible de générer des parties en langage C et d'autres parties en langage VHDL (comportemental ou structurel). On obtient une description mixte C/VHDL qui peut servir à la simulation (co-simulation C/VHDL) et qui sert à la synthèse sur une plate-forme d'architecture composée de microprocesseurs, d'ASICs, et de FPGAs. La description en Solar et qui est le résultat d'un découpage constitue l'entrée à l'étape de synthèse de communication. Le résultat de cette étape peut être traduit automatiquement en une description purement logicielle (programmes en C) ou bien une description purement matérielle (en langage VHDL). Ces deux descriptions constituent les deux cas extrêmes de réalisation. Elles peuvent aussi servir de référence pour analyser les performances d'une réalisation mixte C/VHDL. L'annexe D.1 présente un exemple d'un système émetteur/récepteur décrit en SDL, ensuite cet exemple a été traduit en Solar. La représentation graphique des descriptions Solar relatives aux étapes de synthèse et de transformation sont montrés. L'émetteur a été affecté à une réalisation logicielle et le récepteur ainsi que les autres parties du système ont été affectés à une réalisation en matériel. Le reste de cet annexe D.1 montre les descriptions en C et en VHDL qui sont générées automatiquement.

5.5. Exemple d'application

Pour illustrer la méthode de découpage par PARTIF, l'exemple d'un répondeur téléphonique sera présenté. Cette application comporte un lecteur de cassettes pour enregistrer les messages (Lecteur1), un composant numérique qui permet de lire une annonce préenregistrée (Lecteur2), un compteur de temps (Compteur), et un système de contrôle (CTRL) assurant les différentes fonctions du répondeur (voir figure 5.15). On va s'intéresser à ce dernier système. Le but est de découper la description du système de contrôle en deux partitions afin de réaliser un ensemble parmi ses fonctions en logiciel et le reste des fonctions en matériel. Le logiciel sera exécutable par un microprocesseur standard, en revanche, le matériel sera réalisé par un circuit spécialisé (ASIC) ou un circuit programmable tel qu'un FPGA.

Figure 5.15. *Structure du répondeur de téléphone.*

Dans la figure 5.15, qui est une copie d'écran, la convention de notation utilisée est la suivante :

Les ports d'E/S externes d'un système sont représentés en haut de sa structure (e.g. Ring, Voice, etc.); le reste des signaux (internes) sont représentés en bas (e.g. N_9, N_10, etc.).

Les contraintes de conception qui sont à considérer sont les suivantes :

1- Nombre total de partitions égal à deux, et

2- Nombre total d'interconnexions pour chaque partition égal à vingt.

Le système de contrôle comprend une MEF étendues appelée *behaviour* composée elle-même par deux autres MEFs étendues (*Wait_For_A_Call*, et *OffHook*). L'état d'entrée de la machine *behaviour* est *Wait_For_A_Call*. En cas d'appel, le système compte trois sonneries, et si personne ne décroche le combiné alors l'état suivant est *OffHook*. L'annonce préenregistrée est alors activée, et le correspondant peut poster un message après le bip sonore ou bien interroger le répondeur à distance. Le système de contrôle revient à son état initial (*Wait_For_A_Call*) dès que le correspondant raccroche le combiné. La description en Solar de ce répondeur téléphonique est présentée en annexe A.1.

La figure 5.16(a) montre une copie d'écran de la hiérarchie de la machine *behaviour*. Les niveaux de la hiérarchie sont représentés par des boîtes imbriquées et qui sont alignées verticalement (resp. horizontalement) dans le cas où les MEFs correspondantes sont séquentielles (resp. parallèles). Par exemple, les machines *Wait_For_A_Call* et *OffHook* sont en séquence donc leurs représentations sous forme de boîtes sont alignées verticalement.

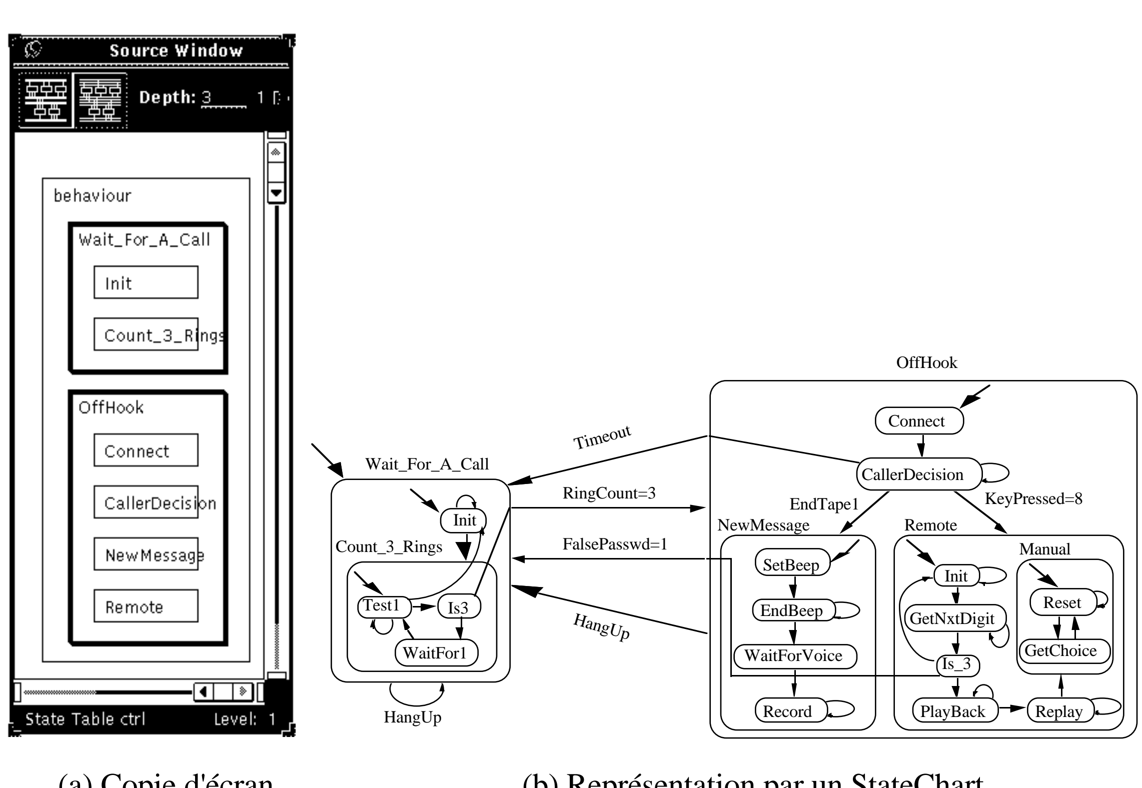


Figure 5.16. *Hiérarchie de la machine behaviour du contrôleur.*

Comme il a été dit précédemment, le découpage avec PARTIF suit une méthode interactive où le concepteur applique successivement les primitives. Le choix de la séquence de primitives est à la charge du concepteur. Dans cet exemple, on s'est fixé comme objectif la réalisation des fonctions d'interrogation à distance du répondeur (MEF *Remote*) dans une technologie logicielle. Les autres fonctions seront réalisées en matériel. Le reste de cette section commente une session (stratégie) de découpage qui inclut la séquence suivante de primitives :

- Move (*Remote*, *behaviour*),
- Merge (*Wait_For_A_Call*, *OffHook*),
- Split (*behaviour*),
- Cut (*behaviour'*),

Afin de séparer la MEF *Remote* du reste de la description, il faut fusionner les MEFs restantes dans une même MEF étendue. Ainsi, on applique la primitive *Move* (voir figure 5.17(a)) juste avant l'application de la primitive *Merge*. Une fois que la primitive *Merge* est appliquée, la MEF étendue *behaviour* se compose d'un côté de la machine *Remote*, et de l'autre côté du reste de la spécification du contrôleur téléphonique (voir figure 5.17(b)). On peut alors appliquer la primitive *Split* pour rendre parallèles ces deux machines (voir figure 5.17(c)) ensuite la primitive *Cut* pour les séparer dans deux partitions différentes (voir figure 5.17(d)). Cette primitive Cut a inséré dans la description un nouveau canal qui sert à contrôler l'accès à quatre variables partagées. En fait, les quatre variables partagées ont été organisées sous forme d'un tableau d'entiers de taille quatre. Ceci a permis de rajouter dans la description un seul canal au lieu de quatre pour la gestion des données partagées. Ces ressources partagées sont accessibles uniquement en mode EREW. Le résultat est alors composé de deux partitions qui seront réalisées respectivement en logiciel et en matériel ainsi qu'un ensemble de quatre ressources partagées qui seront transposées sur une mémoire. Une brève aperçue de la description, en Solar, du résultat de la primitive *Split*, et qui sert en entrée à la primitive *Cut*, est présentée dans ce qui suit.

```
(DESIGNUNIT ctrl
  (VIEW Behaviour (VIEWTYPE "communicatingprocesses")
    (INTERFACE
      (PORT Ring (DIRECTION IN ) (BIT )) ... )
    (CONTENTS
      (VARIABLE MsgCount (INTEGER 0 )) ...
     (STATETABLE behaviour
       (STATELIST behaviour_s )
       (STATE behaviour_s
         (PARACTION
           (STATETABLE Wait_OffH_f_s
            (ENTRYSTATE Wait_OffH_f )
            (GLOBALACTION
              (IF (= HangUp 1 ) (THEN (ASSIGN RecordTape2 0 ) ... )))
            (STATETABLE Wait_OffH_f
              (STATELIST Connect CallerDecision NewMessage Init Count_3_Rings ) ... )
            (STATE idle
             (IF (= ctrl_Wait_OffH_f 1 ) (THEN (NEXTSTATE Wait_OffH_f )))
              (NEXTSTATE idle )))
           (STATETABLE Remote_s
            (ENTRYSTATE idle )
            (GLOBALACTION
              (IF (= HangUp 1 ) (THEN (ASSIGN RecordTape2 0 ) ... )))
            (STATETABLE Remote
              (STATELIST init GetNxtDigit Is_3 PlayBack Replay Manual ) ... )
            (STATE idle
              (IF (= ctrl_Remote 1 ) (THEN (NEXTSTATE Remote )))
              (NEXTSTATE idle )))))))))
```

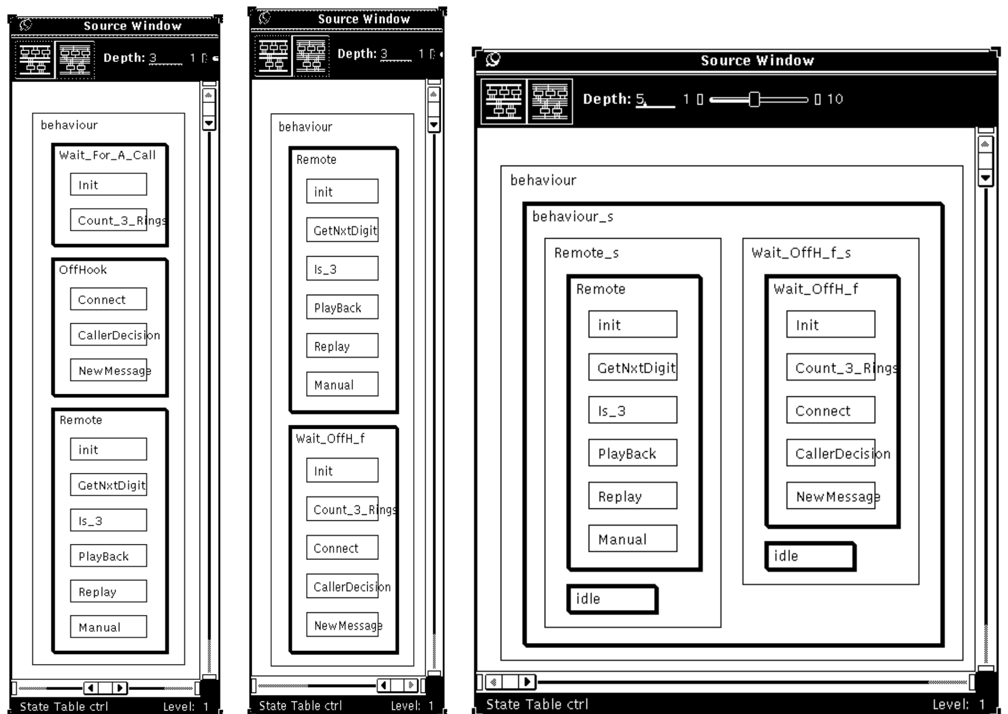
Aperçue de la description en Solar avant l'application du *Cut*.

La résultat de l'application de la primitive *Cut* est donné brièvement par l'extrait de la description, en Solar, suivante :

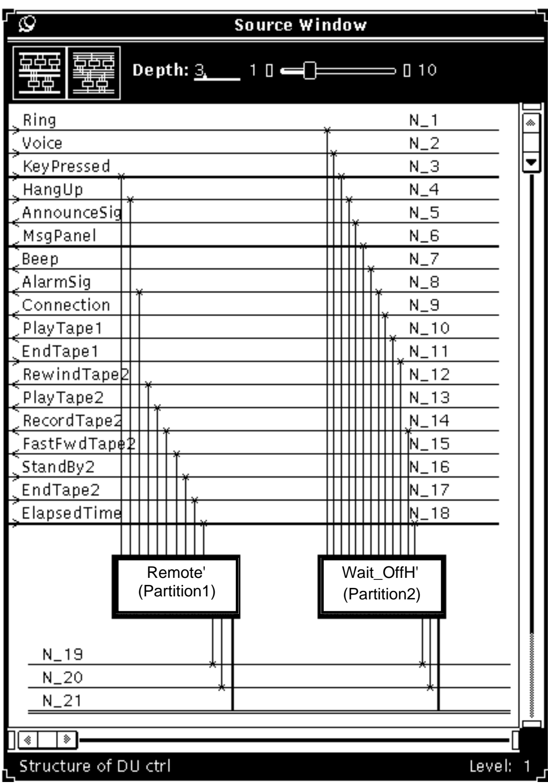
```
(DESIGNUNIT ctrl
  (VIEW Structure (VIEWTYPE "Structure")
    (INTERFACE
      (PORT Ring (DIRECTION IN ) (BIT )) ... )
    (CONTENTS
      (INSTANCE Wait_OffH'
        (VIEWREF Wait_OffH_f_s Wait_OffH_f_s )
         (PORTINSTANCE Ring ) ...
         (ACCESSINSTANCE CTRL_DATA#1 ))
      (INSTANCE Remote'
         (VIEWREF Remote_s Remote_s )
         (PORTINSTANCE Ring ) ...
         (ACCESSINSTANCE CTRL_DATA#1 ))
      (NET N_1
        (JOINED
          (PORTREF Ring )
          (PORTREF Ring (INSTANCEREF Remote_s ))
          (PORTREF Ring (INSTANCEREF Wait_OffH_f_s )))) ...
      (NET N_21
        (JOINED
          (ACCESSREF CTRL_DATA#1 (INSTANCEREF Remote_s ))
          (ACCESSREF CTRL_DATA#1 (INSTANCEREF Wait_OffH_f_s )))))))
 (DESIGNUNIT Remote_s
  (VIEW Remote_s
   (VIEWTYPE "communicatingprocesses")
    (INTERFACE
      (PORT Ring (DIRECTION IN ) (BIT )) ...
      (ACCESS CTRL_DATA#1 (VIEWREF communicatingsystems CTRL_DATA#1 )))
    (CONTENTS ... )))
 (DESIGNUNIT Wait_OffH_f_s
  (VIEW Wait_OffH_f_s
   (VIEWTYPE "communicatingprocesses")
    (INTERFACE
      (PORT Ring (DIRECTION IN ) (BIT )) ...
      (ACCESS CTRL_DATA#1 (VIEWREF communicatingsystems CTRL_DATA#1 )))
    (CONTENTS ... )))
 (CHANNELUNIT CTRL_DATA#1
  (VIEW communicatingsystems
    (INTERFACE
      (PORT data (DIRECTION INOUT ) (INTEGER )) ...
      (METHOD READ ... ))
      (METHOD WRITE ... ))
    (CONTENTS ... )))
```

Aperçue de la description en Solar après l'application du *Cut*.

La figure 5.17 montre le résultat de chaque étape de découpage. Dans la figure 5.17(d), N_19 et N_20 représentent les signaux de contrôle rajoutés par la primitive *Split* et N_21 représente le canal de communication *CTRL_DATA#1* qui gère l'accès aux ressources partagées.



(a) Résultat de Move (b) Résultat de Merge (c) Résultat de Split



(d) Résultat de Cut

Figure 5.17. *Découpage du système de contrôle du répondeur téléphonique.*

Le rapport du coût des deux partitions générées en sortie de PARTIF est présenté dans la table 5.4. Dans cette table, “Total Interconnexions” correspond à la somme des nombres de ports d'entrée et de sortie ainsi que ceux correspondant aux variables partagées (accès à des mémoires ou des registres). Un port d'E/S qui sert à la fois en entrée et en sortie est comptabilisé une seule fois. Ce rapport donne une estimation des performances de chaque partition résultat de ce découpage. Par exemple, il indique l'estimation de la surface et du temps (en nombre de micro-cycles) des partitions générées. La surface estimée ne tient en compte que d'une évaluation simple de la surface des opérateurs utilisés.

Coût	Partition: Remote'	Partition: Wait_OffH'
# Entrées		7
# Sorties		9
Variables Partagées		4
Total Interconnections	16	19
Surface (mm^2)	1.53	0.74
Opérateurs (+, mod)	2	1
# Variables Locales		2
Temps d'exécution (cycles)	66	34

Table 5.4. *Performances des partitions générées à la suite du découpage.*

5.6. Conclusion

L'objet de ce chapitre a été de présenter une méthode de découpage interactif et qui traite des spécifications au niveau algorithmique. L'architecture du système PARTIF, ainsi que des primitives de découpage ont été décrits. Ces primitives permettent au concepteur de découper interactivement la spécification du système à traiter. Une fois que le découpage est effectué, le système PARTIF génère en sortie un rapport d'estimation qui permet de juger la qualité du découpage. Le système PARTIF permet au concepteur d'analyser différentes alternatives de réalisation, en tenant compte des contraintes matérielles à une étape avancée du processus de synthèse.

Les points forts de l'approche qui est proposée sont d'une part l'interactivité, et d'autre part l'utilisation d'éléments de bibliothèque. L'interactivité est un élément

essentiel qui permet de justifier les choix réalisés par le concepteur. En effet, ce dernier peut dresser l'historique de la séquence réalisée d'opérations de découpage et les documenter, ce qui permet, par exemple, d'enrichir l'expérience des futurs utilisateurs qui peuvent être en charge de la maintenance d'une conception. D'autre part, cette approche est basée sur l'utilisation d'éléments de bibliothèque qui sont des canaux de communication e.g. réseau pouvant avoir différents degrés de complexité. Cette approche facilite et favorise la réutilisation d'éléments existants. Ces éléments peuvent être soit des composants conçus en utilisant l'outil PARTIF puis testés et validés, soit des éléments provenant d'autres outils de conception; par exemple, des composants standards de communication qui sont commercialisés.

Néanmoins, l'approche de découpage qui est proposée nécessite des développements ultérieurs. Il s'agit de développer un outil d'estimations plus précises qui permet de guider le concepteur même non expérimenté. Il faut aussi raffiner le découpage. Dans la version actuelle de PARTIF, chaque unité résultat d'un découpage fonctionnel correspond à un processeur. Il serait intéressant de permettre l'exécution de plusieurs unités sur le même processeur (logiciel ou matériel). Ceci permet de réduire le coût de communication et une meilleure utilisation des processeurs. De même, l'affectation logiciel/matériel qui est faite manuellement maintenant, peut être améliorée par le développement d'un outil qui se base sur une description simplifiée de la configuration de l'architecture cible. Ainsi, cette architecture cible ne sera pas simplement dans l'esprit du concepteur mais un paramètre pouvant intervenir dans les choix réalisés par l'utilisateur. La configuration de l'architecture cible peut comporter des paramètres sur le type des processeurs logiciels (e.g. Intel Pentium ou Motorolla 68040 avec leurs caractéristiques) et processeurs matériels utilisés (e.g. FPGA Xilinx de la famille 4000).

4.1. Introduction

Ce chapitre présente une approche pour la conception conjointe logiciel/matériel. La figure 4.1 ci-dessous décrit les principales étapes qui conduisent d'une spécification de système vers un modèle qui puisse être utilisé par des outils de conception pour sa réalisation matérielle et logicielle [BIJe95].

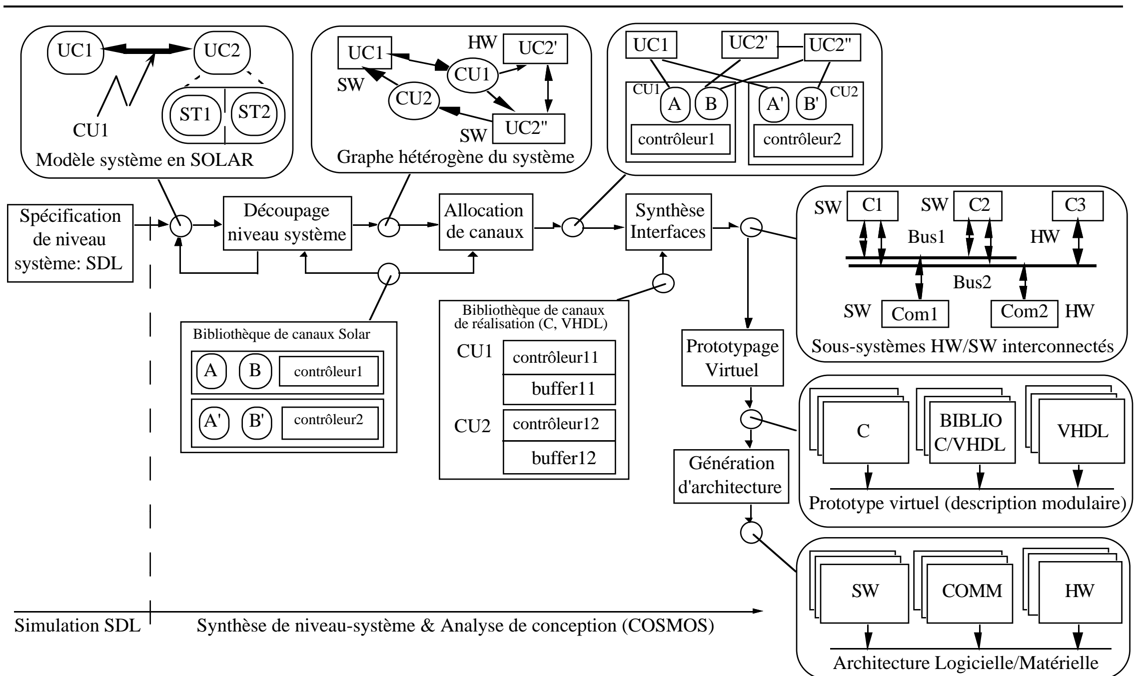


Figure 4.1. *Approche de synthèse dans COSMOS [BIsm94].*

La première étape est celle de saisie d'une spécification en langage SDL. Cette spécification est alors simulée afin de vérifier sa correspondance vis à vis du cahier des charges. De même, cette spécification peut être validée de manière exhaustive afin de s'assurer de la cohérence de son comportement et de détecter des situations non prévues dans la spécification de départ. Une fois que différents scénarios d'exécution sont simulés, l'étape suivante est la traduction de cette spécification en un modèle équivalent en Solar. Ensuite, les étapes de synthèse et de transformation peuvent commencer [BIA94a]. La première étape de cette synthèse est le découpage logiciel/matériel; la seconde étape est la synthèse de communication qui comporte deux phases qui sont la sélection (choix et allocation) de protocoles de communication et la synthèse d'interfaces. L'étape suivante est celle du prototypage virtuel [VaCh95] qui permet de générer du code C et du code VHDL. Une simulation mixte C/VHDL [VaCh95] de ce code est alors réalisée afin de valider les étapes précédentes de synthèse et de s'assurer de la validité du

comportement des descriptions obtenue. Enfin, l'étape finale de ce processus de raffinement et de synthèse est celle de la génération d'architecture. Il s'agit de transposer le comportement distribué dans divers blocs communicants sur une architecture physique existante ou bien qu'il reste à générer. Chacune de ces étapes sera brièvement présentée dans les sections qui suivent. Les étapes de découpage et de synthèse de la communication seront détaillées dans les chapitres 5 et 6.

4.2. Saisie de spécifications

Cette section définit l'étape de saisie de spécifications afin d'obtenir un modèle exécutable sur lequel, des vérifications peuvent être réalisées, et les algorithmes de synthèse et de transformation peuvent être appliqués.

Cette opération consiste à transcrire une description littérale d'un système en un modèle fonctionnel compréhensible par un grand nombre d'utilisateurs et sur lequel des vérifications [Clar89] de conformité, par simulation, peuvent s'effectuer. De nombreux langages de spécification [Dzer90] ont été définis pour répondre à des applications ciblées. Chacun d'eux possède des avantages et des inconvénients pour la modélisation de systèmes [CaCo94]. Le langage SDL, qui répond le mieux aux besoins des applications riches en communications et orientées vers le contrôle, sera adopté.

Actuellement, aucun langage ne décrit tous les concepts du niveau-système cités dans le chapitre 3. Par conséquent, une application complexe (du monde réel) ne peut être décrite simplement par un seul langage. Souvent, on a besoin de décrire les parties (sous-systèmes) de communication par un des langages orientés vers la description de protocoles (SDL [SaTi87], ESTELLE [ISO87], LOTOS [Loto89, BoBr87]), les parties qui doivent réagir à temps réel par les langages de ce domaine (StateCharts [Hare90, DrHa89], SpecCharts [VaNa91], ESTEREL [BeCo84], ARGOS [Mara90, Mara89]), et les parties parallèles par un langage de programmation parallèle (CSP [Hoar78], CSML [ClLo91]). Une telle approche est adoptée dans le cas des systèmes avioniques [RoHa95]. Comme indiqué dans la figure 4.2 une forme intermédiaire [JeOB94, Dutt90] est utile pour la représentation des systèmes dont la réalisation peut comporter à la fois des parties logicielles et des parties matérielles (systèmes mixtes logiciel/matériel) [JeOB93]. Ensuite une représentation de cette forme intermédiaire pourra être transférée dans un langage de description de matériel tel que VHDL (pour les parties à réaliser en matériel) et un langage de programmation de logiciel tel que C (pour les parties à réaliser

en logiciel). L'annexe D.1 donne une illustration d'un exemple décrit en SDL puis traduit en C et en VHDL après des opérations de découpage.

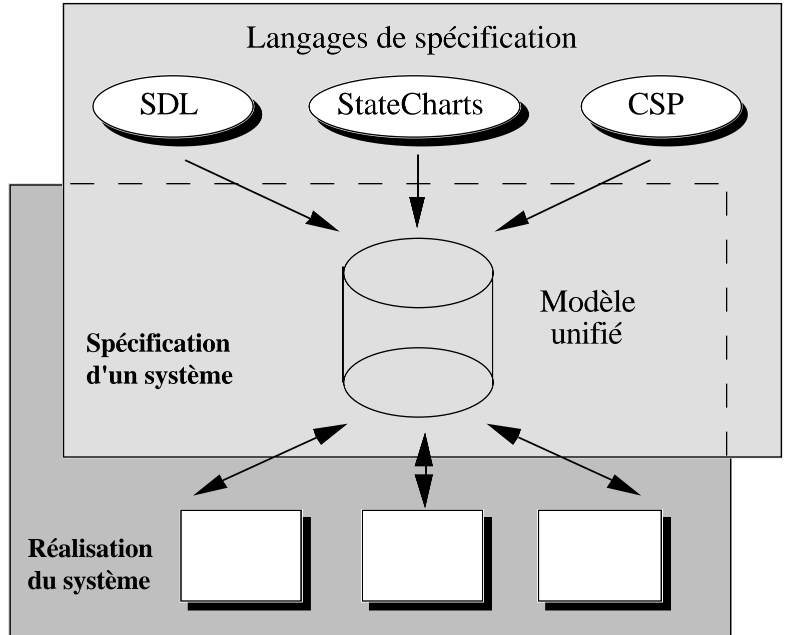


Figure 4.2. *Saisie de spécifications.*

Le choix du langage qui a été retenu dans COSMOS (langage SDL) est justifié dans le chapitre 2 (§ 2.4.2) alors que le modèle unifié, Solar, qui est adopté est décrit dans le chapitre 3 de cette thèse. La première opération, après la spécification, consiste à convertir le modèle de communication d'un système écrit en SDL, et vérifié par simulation, en un modèle faisant intervenir les éléments suivants de Solar :

- Unité de Conception (UC) qui contient des MEF étendues.
- Unité Canal qui assure la communication et qui gère les messages entre les UC par des files d'attentes.

SDL et Solar utilisent le modèle de machine d'états finis (MEF) étendue. En SDL, les MEFs sont décrites par des processus, alors que dans Solar elles sont décrites par des tables d'états. La communication en utilisant SDL est basée sur le transfert de messages moyennant un routage explicite (en utilisant les concepts de canaux et de routes de SDL). Un processus émetteur n'est pas bloqué, lors de l'émission d'un message, puisque tout message est stocké dans la file d'attente du processus récepteur. Chaque processus possède une file d'attente implicite de messages (conformément à la sémantique de SDL). La traduction de SDL à Solar ainsi que la correspondance entre les concepts de SDL et ceux de Solar sont détaillées dans [Romd92]. Mis à part les concepts de communication

qui nécessitent une réorganisation de la description en mettant à plat les canaux et les routes de SDL, la traduction du comportement, décrit en SDL, en Solar est directe. Cette transformation est résumée dans la figure 4.3. La figure 4.3(a) présente une description SDL qui comporte deux blocs B1 et B2 communicant à travers un canal de SDL. Le bloc B1 contient deux processus communicants (P11 et P12). Le bloc B2 contient un seul processus (P13). Le modèle Solar équivalent est composé des trois processus de départ (décrits dans des UC) qui communiquent à travers trois canaux de communication (canal_P11, canal_P12, et canal_P13). Comme indiqué dans la figure 4.3(b), cette communication s'effectue par envoi de messages en appelant les services (méthodes) *send* et *receive* respectifs à chaque canal.

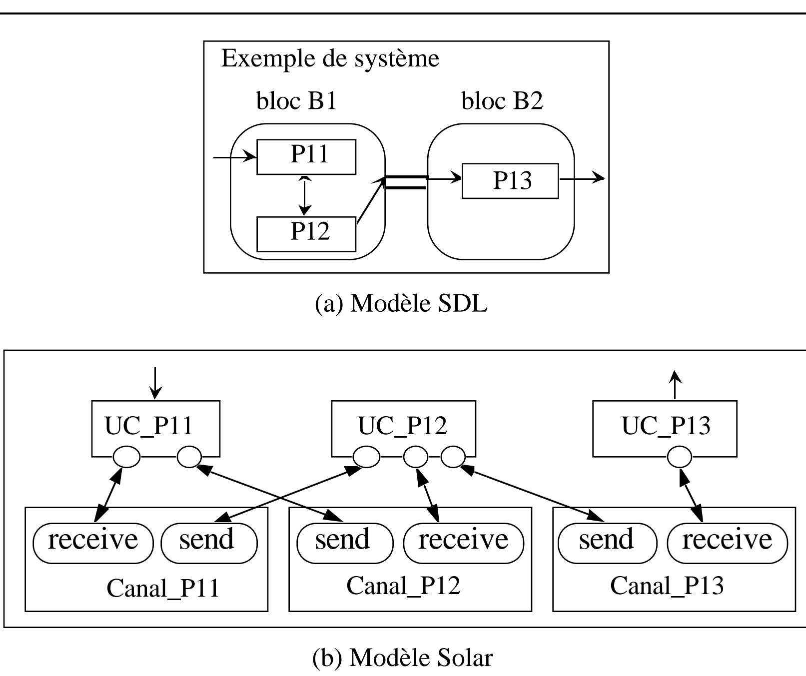


Figure 4.3. *Conversion du modèle SDL en un modèle Solar.*

La communication hiérarchique dans SDL, qui fait appel à des canaux (entre des blocs) et des routes (entre des processus d'un même bloc), a été mise à plat dans le modèle Solar. Par exemple, la communication entre les deux processus P12 et P13 fait appel à un canal ainsi que deux routes (SignalRoutes) dans SDL. Dans le modèle Solar, cette communication ne fait intervenir que les canaux (canal_P12, et canal_P13) relatifs à chacun des processus. En effet, un canal de communication est associé à chaque processus afin de gérer sa file d'attente de messages. Les processus émetteurs font appel au service *send* de ce canal, et le processus récepteur est le seul à pouvoir faire appel au service *receive* du même canal. L'annexe D.1 donne un exemple décrit en SDL et une représentation graphique du système équivalent en Solar.

4.3. Découpage au niveau-système

Le problème du découpage, pour la conception mixte matériel/logiciel [Wolf94, WoDu94], est similaire à celui de l'ordonnancement/allocation de tâches sur différents processeurs d'une machine multiprocesseurs. Les algorithmes d'ordonnancement regroupent ces tâches (processus) en classes de manière à ce que chaque classe puisse être exécutée séquentiellement sur un des processeurs. De la même manière, le but du découpage est de constituer (regrouper) des ensembles de tâches en partitions qui seront réalisées par des processeurs indépendants.

4.3.1. Définition du problème de découpage

Le problème de découpage peut être formulé de plusieurs façons. Dans ce qui suit, une formulation très simplifiée sera présentée.

L'entrée au découpage est une description fonctionnelle d'un système. Cette description peut être soit mono-flot (algorithme séquentiel) soit multi-flots composée d'un ensemble de processus communicants.

La sortie du découpage est un ensemble de partitions où chaque partition est à réaliser soit en logiciel soit en matériel. Les différentes partitions vont contenir chacune une ou plusieurs fonctions provenant de la spécification initiale. Les partitions affectées à une réalisation logicielle nécessitent chacune un processeur pour leur exécution. Les partitions à implanter en matériel peuvent nécessiter des composants d'émulation tels que les FPGAs ou bien elles vont permettre de générer des circuits intégrés (ASICs). Le nombre de partitions générées représente le nombre de processeurs de l'architecture cible. Le concepteur doit essayer de trouver un compromis surface/vitesse d'exécution pour chaque partition. Il peut alors essayer différentes alternatives où il affecte chacune de ces partitions une fois à une réalisation logicielle et une fois à une réalisation matérielle. En se basant sur les résultats des estimations, il pourra décider la technologie de réalisation. Généralement, les partitions, qui font partie du chemin critique de l'application et demandent un temps de réponse court, sont à réaliser en matériel. Les autres partitions sont à réaliser en logiciel afin de réduire le coût monétaire de réalisation.

- **Définition d'une partition**

Soit la spécification d'un système comportant un ensemble $\mathcal{F}$ de fonctions qui sont à découper. Une partition $\mathcal{P}$ est formée d'un sous-ensemble de fonctions de $\mathcal{F}$. Cette partition est à réaliser soit en logiciel soit en matériel.

• Définition d'une alternative de découpage

Une alternative $\mathfrak{A}$ de découpage pour un ensemble $\mathfrak{F}$ de fonctions est un triplet composé d'un ensemble de partitions $\mathfrak{P}_M$ à réaliser en matériel, d'un ensemble de partitions $\mathfrak{P}_L$ à réaliser en logiciel, et d'un ensemble d'unités de communication $\mathfrak{C}$ (voir équation 1).

$$\mathfrak{A}(\mathfrak{F}) = \{ \mathfrak{P}_L, \mathfrak{P}_M, \mathfrak{C} \} \qquad (1)$$

Les unités de communication sont réalisable en logiciel ou en matériel. Elles servent à connecter des partitions à réaliser entièrement en logiciel ou entièrement en matériel. Elles peuvent également servir à connecter d'une part des partitions à réaliser en logiciel et d'autre part des partitions à réaliser en matériel.

Ainsi, un spectre d'alternatives est possible pour la réalisation d'un ensemble $\mathfrak{F}$ de fonctions. Les classes possibles d'alternatives de découpage sont exprimées dans le reste de cette section. L'utilisation d'une réalisation purement logicielle, purement matérielle, en unités matérielles communicantes, en unités logicielles communicantes, ou bien en unités logicielles et matérielles communicantes sont exprimés par les équations respectives (2), (3), (4), (5) et (6) ci-dessous :

$$\mathfrak{A}_1(\mathfrak{F}) = \{ \mathfrak{P}_L, \emptyset, \emptyset \} \qquad (2)$$

$$\mathfrak{A}_2(\mathfrak{F}) = \{ \emptyset, \mathfrak{P}_M, \emptyset \} \qquad (3)$$

$$\mathfrak{A}_3(\mathfrak{F}) = \{ \emptyset, \mathfrak{P}_M, \mathfrak{C} \} \qquad (4)$$

$$\mathfrak{A}_4(\mathfrak{F}) = \{ \mathfrak{P}_L, \emptyset, \mathfrak{C} \} \qquad (5)$$

$$\mathfrak{A}_5(\mathfrak{F}) = \{ \mathfrak{P}_L, \mathfrak{P}_M, \mathfrak{C} \} \qquad (6)$$

Les deux premiers cas représentent les cas extrêmes entre une réalisation monoprocesseur entièrement logicielle et une réalisation entièrement matérielle. Les équations (4) et (5) représentent un modèle distribué homogène en logiciel et en matériel respectivement.

Dans le projet COSMOS [BIA94b], chaque partition va correspondre soit à un programme C à compiler pour un processeur standard, soit à une description VHDL qui va permettre de générer un circuit intégré. Chaque description VHDL, relative à une partition, peut aussi servir à programmer un FPGA. Le découpage logiciel/matériel d'un système est un problème NP-complet. L'approche suivie dans ce projet est interactive en raison de la complexité de ce problème.

Le découpage d'un système peut être considéré comme étant un problème d'optimisation de performances de ce système selon une certaine fonction coût. La variable principale de cette optimisation est la granularité des partitions. Le découpage d'un système est terminé quand une découpe optimale est obtenue.

Tout le problème du découpage revient à un bon choix de la granularité des partitions. Le but est de minimiser (1) les interconnexions et (2) la surface nécessaire à la réalisation des parties matérielles et à augmenter (3) la vitesse d'exécution. Ces trois critères sont généralement interdépendants; il s'agit alors de faire un compromis entre les trois. L'amélioration de l'un des trois critères entraîne dans la plupart des cas une dégradation des autres.

Prenons le cas simple où on voudrait réaliser différentes fonctions de calcul qui seront appelées par plusieurs processus. On suppose que ces fonctions sont indépendantes et nécessitent plusieurs ressources physiques tels que les unités arithmétiques et logiques, les mémoires et les registres. On peut se fixer comme objectif (prioritaire) la réduction de la surface de la partition qui va les contenir. Dans le cas d'une réalisation en matériel, on va essayer d'utiliser les même ressources physiques pour toutes les fonctions. Ce partage de ressources va permettre d'économiser des composants matériels donc de la surface sur silicium. Cependant, dans le cas où différents processus font appel en même temps à différentes fonctions, ceci va affecter le temps de réponse du système total. Ainsi, au lieu d'utiliser des ressources distinctes pour chaque fonction, on a optimisé le nombre de ressources et gagné en surface, mais on a généré une réalisation plus lente donc moins performante. Une estimation précise des critères précédemment mentionnés s'avère nécessaire afin d'arriver à un compromis entre les trois.

4.3.2. La méthode interactive dans COSMOS

Le découpage, dans le projet COSMOS [BIA94c], permet d'affecter chaque unité de conception (UC) et chaque unité canal à un type de réalisation (voir figure 4.4). Comme il a été dit dans la section précédente, le choix s'est porté sur une approche interactive et non automatique en raison de la complexité de ce problème. L'outil de découpage, appelé PARTIF, possède des opérations permettant de regrouper, réorganiser, découper, ou distribuer les UC, et de les faire communiquer à travers des canaux. Afin de favoriser la ré-utilisation de composants de communication, ces canaux sont répertoriés dans une bibliothèque. Chacun des UC et des canaux sera affecté à une technologie de réalisation allant d'une réalisation complètement matérielle, réalisée avec

des instructions micro-codées, jusqu'à une réalisation logicielle s'exécutant sur un système d'exploitation connu.

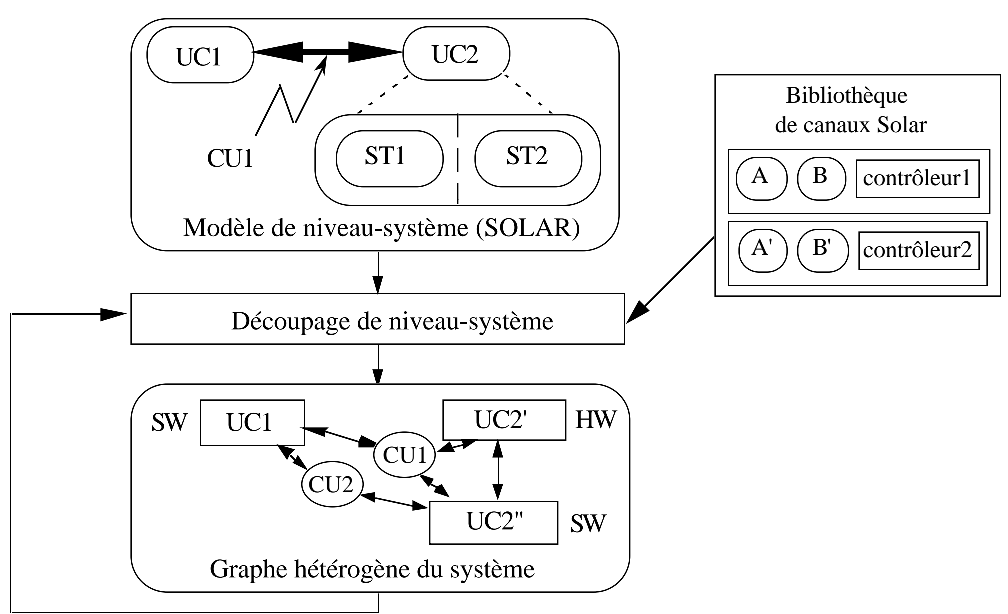


Figure 4.4. *Étape de découpage dans COSMOS.*

Les critères de choix des opérations (regrouper, réorganiser, découper, ou distribuer) à appliquer de manière interactive sont laissés à la discrétion du concepteur. Son choix sera dicté en fonction des performances, du coût de réalisation, de la fréquence d'utilisation, de la reprogrammation du logiciel, de la réutilisation du matériel, des familles de technologies retenues, etc.

4.3.3. Les primitives

L'idée est de fournir au concepteur un environnement de type boîte à outils avec un ensemble de primitives de transformation et de découpage de descriptions. Dans l'approche qui est proposée dans cette thèse, les opérations de découpage sont réalisées à l'aide de ces primitives [BenI92]. Ces primitives permettent d'exécuter des opérations de base sur une spécification, comme le changement du niveau d'une machine d'états finis dans la hiérarchie d'un système. Les transformations plus complexes seront réalisées en combinant plusieurs primitives de base. Le découpage est réalisé à l'aide de quatre primitives de base qui sont appelées par : *Move*, *Merge*, *Split*, et *Cut*. Ces primitives réalisent des transformations de MEFs. La primitive *Move* sert à déplacer des machines à travers la hiérarchie d'une machine d'états finis étendue. La primitive *Merge* fusionne des machines séquentielles pour produire une seule machine. La primitive *Split* découpe une

machine séquentielle et produit plusieurs machines en parallèle. Une méthode de découpage de machines séquentielles est présentée dans [Józw90, JóKo91]. La primitive *Cut* transforme une machine parallèle en un ensemble de machines interconnectées. Cette approche itérative d'application des primitives s'arrête lorsque toutes les contraintes sont satisfaites. L'outil de découpage, PARTIF, sera détaillé dans le chapitre 5.

4.4. La synthèse de communication

Cette section définit l'activité de synthèse de la communication. L'entrée pour cette synthèse est un ensemble de processeurs abstraits qui représentent les différentes partitions résultat d'un découpage. Ces processeurs coopèrent entre eux à travers un réseau conceptuel de communication. La sortie de la synthèse de communication est un ensemble de processeurs interconnectés. L'objet de la synthèse de communication est de transformer un système composé d'un ensemble de processeurs communicants en un système composé d'un ensemble de processeurs interconnectés. Ces derniers processeurs communiquent simplement via des signaux et peuvent partager le contrôle de la communication.

Dans cette thèse, l'approche proposée pour la synthèse de communication est réalisée en deux étapes qui sont la sélection de protocoles et la synthèse d'interfaces. L'intérêt de séparer cette activité en deux étapes est de pouvoir raffiner la communication. L'idée est de retarder autant que possible les choix concernant la réalisation physique de la communication. Dans cette approche chaque étape fait appel à une bibliothèque de communication. L'étape de synthèse de protocoles fait appel à une bibliothèque qui comporte un ensemble d'unités de communication. Chaque unité à un type déterminé de protocole et représente l'abstraction d'un composant physique de communication. Pour chaque composant de cette bibliothèque, il peut exister plusieurs réalisations possibles. Ces réalisations peuvent différer, par exemple, par la capacité de stockage des files d'attente respectives qu'elles peuvent comporter. La synthèse d'interfaces fait appel à une bibliothèque de réalisations de la communication.

4.4.1. La synthèse de protocoles

Cette opération consiste à sélectionner parmi les canaux (CU) précédemment prévus dans une bibliothèque celui en mesure d'exécuter la communication entre les UCs concernées (voir figure 4.5). Cette communication peut se réaliser, soit sous forme de protocoles comportant des fonctions de résolution des conflits d'accès (contrôleurs), soit

sous forme de signaux échangés directement entre les UCs [OBPa92]. Tous ces éléments de communication sont catalogués dans la bibliothèque. Les critères de sélection vont dépendre d'une part :

- Du type de communication retenu (parallèle, série),
- Des performances demandées,
- Du protocole de communication à réaliser (synchrone, asynchrone, etc.)

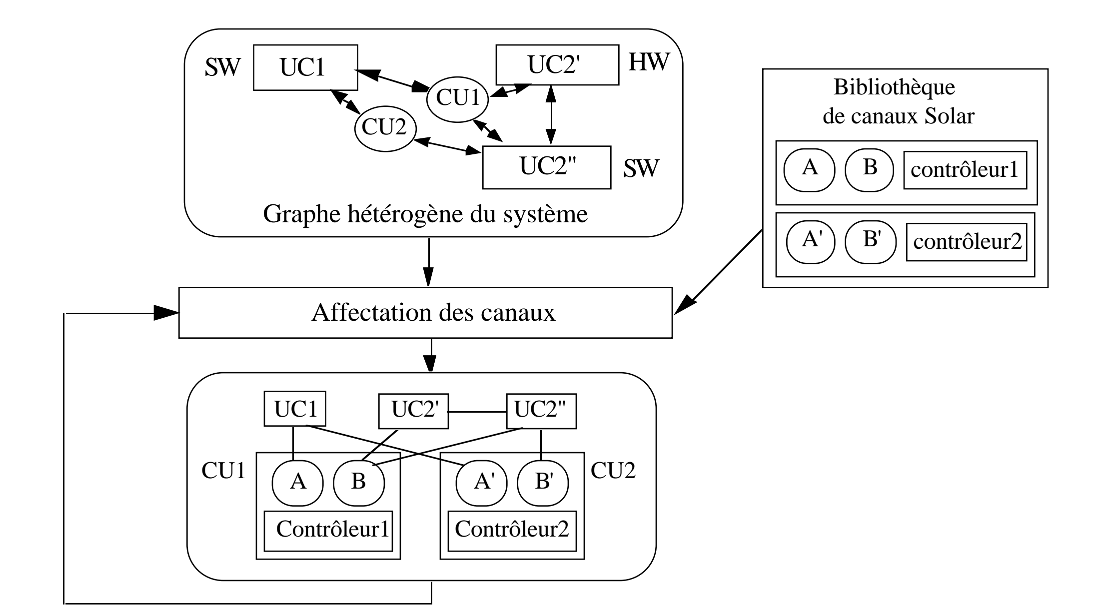


Figure 4.5. *Étape d'affectation des canaux.*

Un canal possède généralement plusieurs méthodes (services) de communication, celle-ci sont indiquées dans le graphe résultat d'affectation des canaux.

4.4.2. La synthèse d'interfaces

Les canaux étant connus, l'étape suivante a pour objectif de distribuer les éléments nécessaires à la réalisation de ces canaux à l'ensemble du système. Ainsi, les services qui réalisent la communication seront attachés aux UCs. Les contrôleurs de communication quant à eux vont apparaître explicitement lors de cette opération (voir figure 4.6). La méthode consiste à examiner dans les différents UCs tous les appels aux services de communication et de les remplacer par des appels à des procédures locales réalisant ces services. Bien entendu, l'interface des UCs subit aussi des modifications pour l'adapter aux services retenus. La réalisation (logiciel, matériel) des procédures dépend de la réalisation des UCs. De même la réalisation (logiciel, matériel) des contrôleurs de

communication et des bus doit répondre aux impératifs de communication demandés par les canaux et les UCs. Dans cette étape, la primitive appelée *Map* transforme des sous-systèmes communicants en sous-systèmes interconnectés en réalisant les modifications d'interfaces citées plus haut.

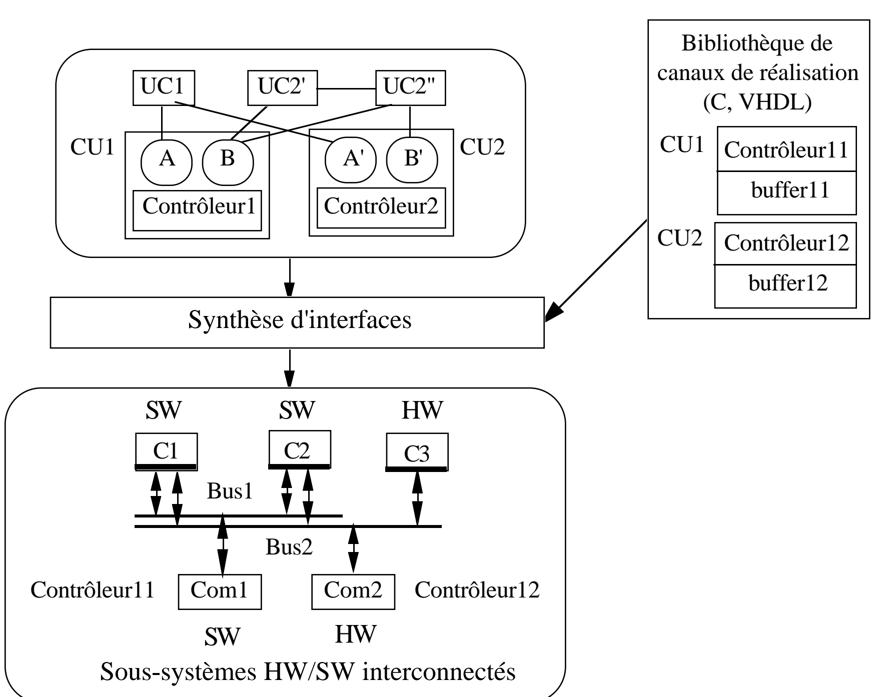


Figure 4.6. *Étape de synthèse des interfaces.*

4.5. Le prototypage virtuel

Le prototypage virtuel est une étape qui génère du code exécutable pour chaque processeur abstrait modifié lors de l'étape précédente de synthèse de la communication. Les descriptions produites sont à la fois simulables et peuvent servir à la synthèse. Dans le projet COSMOS [VaCh95], cette opération consiste à générer, à partir de Solar, le code exécutable de chacun des processeurs abstraits en fonction de leur réalisation :

- Code C pour une réalisation logicielle,
- Code VHDL (comportemental ou structurel) pour une réalisation en matériel (ASIC ou FPGA).

Chaque sous-système (résultat de l'étape de synthèse de communication) est traduit séparément. La sortie du prototypage virtuel est une architecture hétérogène représentée par du code C (processeurs virtuels logiciels) et du code VHDL (processeurs virtuels matériels). Les codes correspondants aux processus de communication, que ce soient les procédures d'appel locales ou les contrôleurs de communication, sont extraits de la bibliothèque de réalisation de la communication (voir figure 4.7). Bien entendu, ils correspondent au mode de réalisation logiciel ou matériel qui leur est affecté. Ils sont donc décrits soit en code C, soit en VHDL.

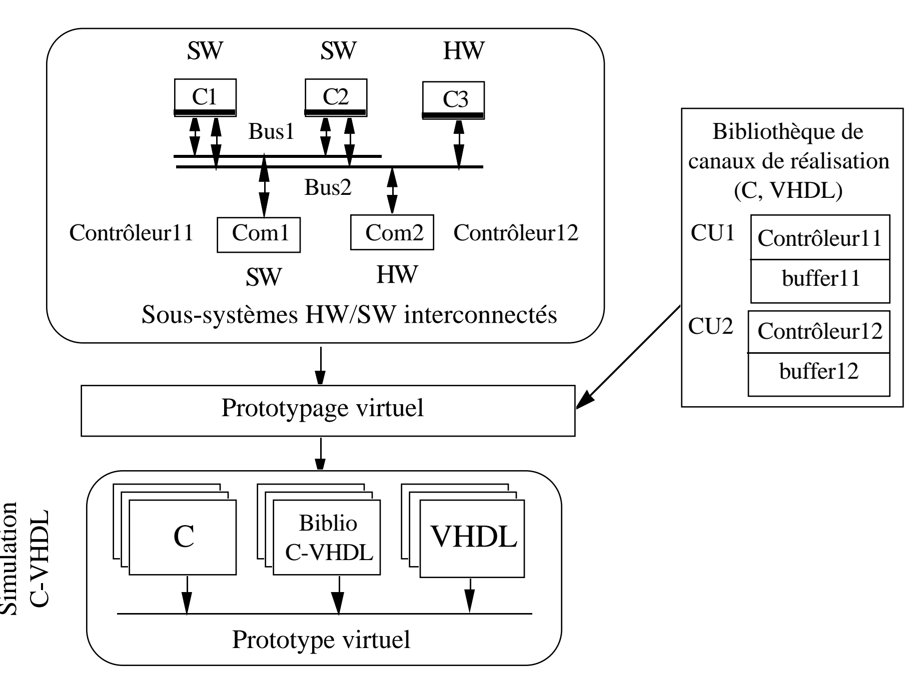


Figure 4.7. *Étape de génération d'un prototype virtuel.*

Le prototype qui est constitué de codes C et VHDL peut être exécuté à partir des mêmes stimuli utilisés pour la vérification de la spécification initiale décrite en SDL. C'est un moyen de vérifier, par simulation [Rows94, LeRa93, BeSi92], que les étapes de synthèse de niveau système n'ont pas provoqué de dérives du comportement initial. Par ailleurs, ces descriptions exécutables (en C ou en VHDL) permettent d'estimer avec plus de précisions certaines caractéristiques de performances d'un système.

4.5.1. La génération de code VHDL comportemental

Dans cette section la génération de VHDL comportemental, à partir d'une description en Solar, sera présentée [VaCh95]. Ce travail est cité pour la complétude de

l'exposé. En premier lieu, la correspondance entre les constructeurs de Solar et ceux de VHDL sera donnée. Le tableau 4.1 résume cette correspondance.

Constructeur Solar	Constructeur VHDL
DesignUnit	Entity
View	Architecture
Interface	Port
Port	Port
StateTable	Process
State (S)	When (S)
Alt (Condition)	If (Condition) Then
SateList	Type State_Type is ()
Instance	Component
ViewRef	Architecture
PortInstance	Port
Net	Port map
Joined	Port map
InstanceRef	Instanciation (X : Instance)
PortRef	Port map

Table 4.1. *Correspondance entre Solar et VHDL comportemental.*

Pour plus de détails concernant cette traduction, un exemple de traduction d'une table d'états Solar (StateTable) en un processus VHDL sera proposé. Il faut préciser au passage que la description Solar à traduire en VHDL doit impérativement contenir une seule table d'états composée d'états simples non hiérarchiques. Cette limitation se justifie par le fait que la description VHDL doit contenir un seul processus pour être acceptable par les outils de synthèse comportementale. La figure 4.8 présente le procédé de traduction d'une table d'états en un processus VHDL.

En Solar, une unité de conception structurelle est composée d'une interface (ensemble de ports), d'instances d'autres unités de conception (structurelles ou comportementales) et d'un réseau d'interconnexion. Les instances permettent l'utilisation d'unités de conception déjà existantes en décrivant une partie ou la totalité de leurs interfaces. Ceci facilite la réutilisation de composants existants et permet d'éviter de redéfinir une unité de conception déjà disponible. D'autre part, plusieurs instances d'une même unité de conception peuvent être utilisées dans un même système. Le réseau d'interconnexion relie les instances des unités de conception.

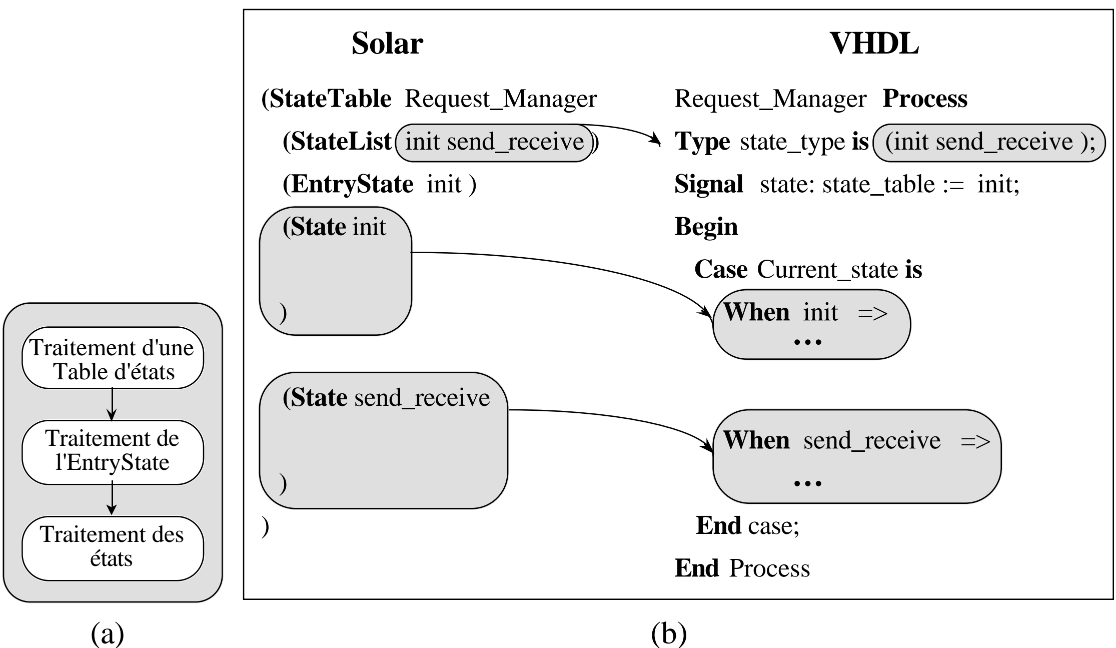


Figure 4.8. *Procédé de traduction d'une table d'états :*
(a) Flot de traitement, (b) Exemple de traduction d'une table d'états.

Comme le montre la figure 4.9, l'interface d'une unité de conception est traduite directement en une entité renfermant l'ensemble des ports de l'interface. Les instances d'unités de conception sont traduites en composants (component) VHDL (voir figure 4.9). La traduction du réseau d'interconnexion en VHDL nécessite l'introduction d'instructions de connexion tels que les “port map”.

Chaque fois que le mot clef *Interface* est rencontré, le traducteur, appelé s2cv [VaCh95], génère la déclaration des ports en VHDL à l'intérieur d'une entité. Le mot clef *Contents* qui spécifie la description du contenu d'une unité de conception permet de générer une *Architecture* contenant la fonctionnalité du circuit correspondant. Les noms de l'entité et de l'architecture sont ceux de l'unité de conception (*DesignUnit*) et de la vue (*View*) respectivement (voir figure 4.9).

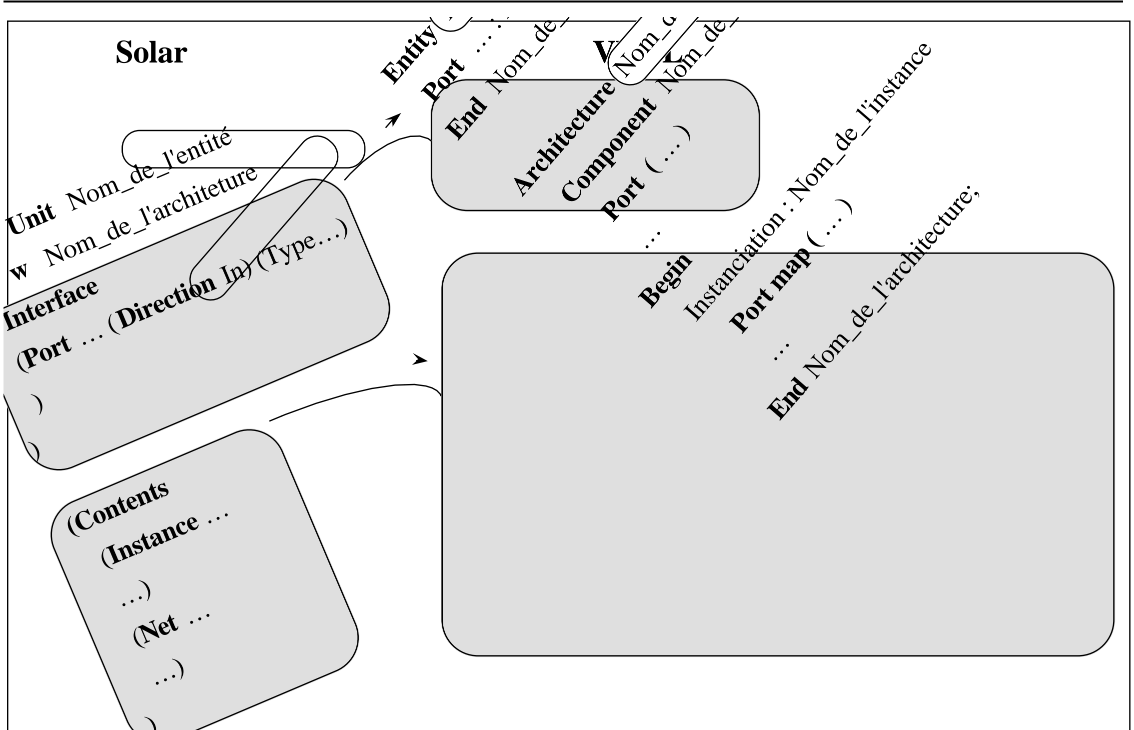


Figure 4.9. *Procédé de traduction de l'unité de conception structurelle.*

4.5.2. La génération de code C

Dans cette section, la génération de code C à partir d'une description en Solar sera brièvement survolée. Le point important concernant cette traduction est le style de description C obtenue. En fait, comme le montre la figure 4.10, vu qu'une description à traduire suit le modèle de machine d'états, le code C qui est généré est une instruction *Case*. Par exemple une machine d'états, en Solar, formée de quatre états (S1, S2, S3, et S4) sera traduite en une instruction *Case* où chaque cas correspond à l'un des états.

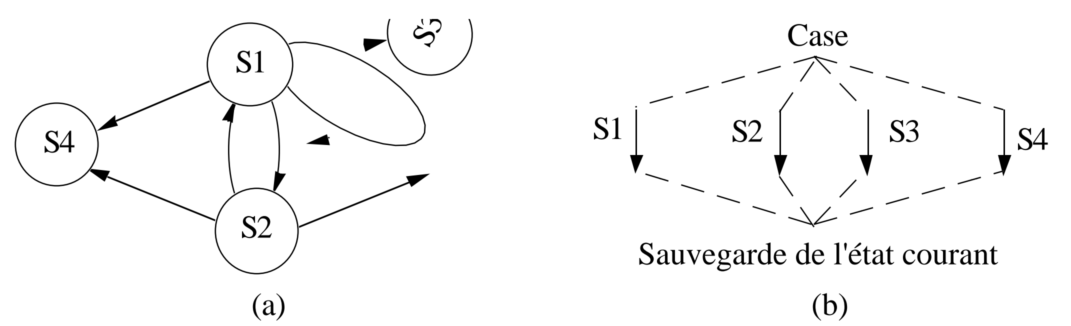


Figure 4.10. *Style de description d'un programme C généré :*
(a) Modèle de la machine d'états décrite en Solar, (b) Modèle du programme C généré.

4.5.3. La co-simulation matérielle/logicielle C/VHDL

Cette section introduit une brève description de la co-simulation C/VHDL [VaCh95] dans le projet COSMOS. Dans cette approche, le simulateur de l'outil “Synopsys” est utilisé. Plus encore, la simulation C/VHDL [Fros93] fait appel à des fonctions définies dans “Synopsys” [Syno93] pour interfacer les programmes en C avec les descriptions en VHDL. La figure 4.11 montre un exemple d'un Émetteur/Récepteur ainsi que l'environnement de co-simulation. Dans cet exemple, le code de l'émetteur et celui du récepteur ont été généré en C. Le canal de communication a été généré en VHDL. Cet environnement permet en même temps du débogage des programmes en C avec la simulation des parties en VHDL. Ceci peut servir à détecter des fautes de synchronisation qui peuvent apparaître sur les chronogrammes de simulation. L'annexe D.1 donne la description en SDL de cet exemple. Après une illustration des étapes de transformation, cet annexe montre une alternative de découpage avec une affectation de l'émetteur à une réalisation logicielle et une affectation du reste du système à une réalisation matérielle. Les descriptions générée en C et celles générées en VHDL sont alors données.

Figure 4.11. *Co-simulation de l'application Émetteur/Récepteur basée sur IPC.*

4.6. La génération d'architecture

La génération d'architecture [ChJe95] produit une architecture de réalisation (ou d'émulation) de la spécification initiale (voir figure 4.12). Cette transposition architecturale s'effectue en utilisant des compilateurs standards pour les parties à réaliser en logiciel, et des outils de synthèse (comportementale [GaDu92, JePO93] ou logique [ThLW90]) pour les parties à réaliser en matériel. Les compilateurs transforment les programmes C en code assembleur et les outils de synthèse transforment les descriptions VHDL en circuits spécifiques (ASIC) [MFK90] ou bien en processeurs matériels virtuels (émulateurs e.g. FPGA). Cette architecture générée va contenir des parties logicielles (paire logiciel/processeur), des parties matérielles, et des parties qui assurent la communication.

Figure 4.12. *Génération d'architecture.*

Actuellement, dans le projet COSMOS, les FPGAs sont utilisés pour réaliser les parties matérielles. Les avantages de ces circuits programmables sont le temps rapide d'implantation et la possibilité de re-programmation (limitée ou infinie dans certains cas). En revanche, les inconvénients des FPGAs, lorsqu'ils sont comparés aux circuits intégrés, sont le temps d'exécution moins rapide et la densité qui est plus faible.

Les FPGAs sont très pratiques pour le prototypage quand une émulation rapide d'une spécification est nécessaire. Ils peuvent aussi servir s'il est envisageable que la spécification des fonctions à réaliser puisse changer.

Aujourd'hui, la popularité des FPGAs peut s'expliquer par leur facilité d'utilisation. Ils fournissent des circuits logiques et numériques avec une complexité moyenne (actuellement des milliers de transistors) et peuvent être utilisés pour des applications spécifiques et dans un seul circuit intégré. Les FPGAs peuvent servir dans trois créneaux :

• Le prototypage rapide : Un FPGA permet de transférer une nouvelle application sur silicium en quelques heures alors que la réalisation des circuits intégrés (ASICs) nécessite des semaines voire des mois. Ceci est intéressant pour l'évaluation de plusieurs options d'architecture de réalisation pour une application donnée.

• L'émulation de matériel : Ceci est particulièrement efficace pour des applications pour lesquelles la simulation logique [HaMe93, LoDo93, FlJe93] s'avère inutilisable à cause d'événements survenant à temps réel. Un FPGA peut servir comme circuit de prototypage pas cher dans la mesure où le temps de conception est une tâche critique.

• L'accélération de fonctions : Un FPGA peut fournir une aide précieuse pour le développement d'algorithmes parallèles en supportant la reconfiguration [VeMa94]. Autrement dit, le même matériel peut se comporter différemment à différents moments dans le but de satisfaire au mieux les besoins de communication entre plusieurs processeurs parallèles.

4.7. Conception orientée vers la maintenance

Cette section donne un bref aperçu de la conception logiciel/matériel appliquée à la maintenance. En fait, les exigences opérationnelles des systèmes et les impacts économiques étroitement dépendants des arrêts d'exploitation nécessitent des études de plus en plus approfondies des fonctions, mais aussi de la bonne exploitation du système que l'on appelle “maintenabilité”. Ces aspects nouveaux doivent en particulier englober les préoccupations concernant la surveillance du système et garantir l'intégrité des données et des fonctions.

Une étude a été réalisée, dans le cadre de cette thèse, afin de proposer une méthodologie de conception de systèmes électroniques complexes qui permet de prendre en compte dès les phases de spécification, les impératifs liés à la maintenabilité. L'étude montre comment faire évoluer les machines d'états fonctionnelles afin de prendre en compte la maintenabilité et comment sont traités les différents canaux de communication

et de maintenance. Elle montre également comment agissent les réalisations en terme de matériel et de logiciel sur les fonctions et comment sont traités les incidents.

Les processus de maintenabilité doivent cohabiter harmonieusement avec les fonctions du système. Ils ont pour objet de détecter, diagnostiquer et corriger les anomalies fonctionnelles. Pour qu'ils soient efficaces, les fonctionnements erronés doivent être analysés au plus près du défaut. C'est la raison pour laquelle ces processus sont répartis à travers l'ensemble du système. Ils peuvent se matérialiser différemment, en fonction des objectifs à atteindre (coûts, performances, disponibilité, etc.). Les processus de maintenance sont activés localement ou à distance, instantanément ou en différé, autonomes ou sous le contrôle d'un opérateur et peuvent entraîner un arrêt temporaire ou définitif des processus fonctionnels.

Figure 4.13. *Bloc télécommande d'un satellite.*

L'exemple suivant illustre cette approche qui privilégie la maintenance. Il s'agit d'une fonction télécommande d'un satellite appelé MILSTAR. La figure 4.13 décrit une vue globale de cette application. Le système en question comporte deux cartes qui reçoivent les commandes à partir de la terre. La première carte assure le fonctionnement nominal alors que la deuxième est utilisée en cas de pannes. Les deux cartes comportent les mêmes services de réception de messages, de synchronisation de trames, d'identification de messages, et de télécommande des relais du satellite. La deuxième carte peut être utilisée dans sa totalité, pour remplacer la première en cas de défaillances graves, ou en partie, par exemple en utilisant quelques uns de ses services. Les deux cartes sont reliées par un canal de maintenance. Celui-ci peut gérer deux types de pannes;

les premières sont dues à des messages erronés (par exemple non destinés au satellite); les deuxièmes sont dues soit à des pannes matérielles de composants soit à des pannes logicielles. Ce canal de maintenance est traité, dans l'approche qui est proposée, de manière analogue aux autres canaux de communication. Néanmoins, un canal de maintenance peut comporter des processus qui permettent de gérer des pannes ayant différents niveaux de gravité. Un canal de maintenance est aussi utilisé pour répertorier toutes les pannes survenues afin de pouvoir informer les opérateurs au sol. La description en SDL du système de la figure 4.13 est donnée dans l'annexe D.2.

Cette recherche [OuJe95] a permis de jeter les bases d'une méthodologie pour modéliser la maintenance en vue des phases de synthèse adaptées à des systèmes composés de processus communicants. On a voulu aussi montrer que la maintenabilité pouvait être traitée en même temps que les aspects fonctionnels. Ceci permet d'aboutir à une réalisation du système intégrant aussi bien les fonctions de surveillance que celles relatives à la mise au point et au test des prototypes, de production, et de dépannage des systèmes.

Lorsque l'on connaît les enjeux économiques des systèmes électroniques, dans un environnement quotidien, et ce que représentent leurs coûts de test et de maintenance, on se rend compte de la nécessité de traiter correctement tous ces aspects.

4.8. Conclusion

Dans ce chapitre les étapes d'une approche de conception conjointe logiciel/matériel ont été présentées. Cette approche commence avec une spécification au niveau système et aboutit à une réalisation. Les différentes étapes et les modèles qui constituent l'approche proposée ont été décrits. Actuellement, le point de départ est une spécification en SDL qui est traduite en Solar une fois qu'elle est simulée dans l'environnement (atelier logiciel) de SDL. Toutes les étapes de synthèse sont réalisées sur des modèles intermédiaires en Solar. Ensuite, les parties logicielles sont traduites en C et les parties matérielles sont traduites en VHDL. Cette traduction est réalisée automatiquement par un outil développé pour générer du code C et du code VHDL comportemental ou structurel.

Le point clef de cette approche est l'utilisation d'un modèle général pour la communication. La séparation entre les descriptions de communication et celles de calcul permet la réutilisation de modèles et de composants de communication existants. En effet,

chaque unité de communication peut être spécifiée à différents niveaux de détails allant du niveau fonctionnel comme une boîte noire reliant des composants jusqu'au niveau de réalisation physique. Les avantages d'avoir une approche de conception basée sur des bibliothèques de canaux décrits à différents niveaux d'abstraction sont :

1. L'amélioration de la réutilisation de conceptions existantes, en favorisant la réutilisation d'éléments de bibliothèque aux dépens d'une nouvelle conception. Ainsi, un sous-système peut être conçu puis réutilisé comme composant dans un autre système plus grand. Néanmoins, un composant de bibliothèque peut provenir d'outils externes.

2. Une bibliothèque de canaux offre au concepteur une multitude de choix de communications. Celui-ci peut alors choisir les protocoles de communication appropriés à son application. Par exemple, il peut utiliser dans la même application des protocoles synchrones et asynchrones qui peuvent se baser sur des échanges de messages bloquants ou non bloquants.

3. Les différents modules d'un système peuvent être mis à jour plus facilement. Cette flexibilité rend plus aisée la maintenance de systèmes complexes.

4. Etant donné qu'il n'existe pas de contraintes imposées sur les modèles de communication, cette approche de conception conjointe logiciel/matériel peut être appliquée à un grand nombre d'applications.

3.1. Introduction

Dans ce chapitre, le format intermédiaire, Solar [JeOB92, OBri93], qui est un modèle unifié pour la représentation du logiciel et la représentation du matériel, sera présenté. Ce format permet d'accommoder les aspects d'une sémantique logicielle et ceux d'une sémantique matérielle. Il sert aussi à tous les outils de synthèse au niveau système qui ont pour objet de synthétiser et de concevoir des systèmes mixtes logiciels/matériels. Ce chapitre ne donne qu'un aperçu très bref sur Solar. Plus de détails se trouvent dans [OBri93].

Ce chapitre est composé de six sections. La section 2 expose les concepts de base de Solar. Les sections 3, 4, et 5 présentent respectivement les formalismes de la table d'états, de l'unité de conception, et du canal de communication. Finalement, la section 6 propose la conclusion de ce chapitre.

3.2. Le format Solar : Les concepts de base

La principale motivation pour le développement de Solar a été d'accommoder dans une même représentation les principaux concepts du niveau système (hiérarchie, parallélisme, communication, et synchronisation). Comme il est montré dans la figure 3.1, le but est d'accommoder plusieurs langages tels que SDL [SaTi87], ESTELLE [ISO87], LOTOS [Loto89, BoBr87, BrSc87, NaBa86], StateCharts [Hare90, DrHa89], ESTEREL [BeCo84], ARGOS [Mara90], CSP [Hoar78], OCCAM [Jone87], CSML [ClLo91]. Solar a été conçu pour représenter à la fois les concepts manipulés par les langages de spécification de systèmes et ceux manipulés par les langages de description de matériel et de logiciel. C'est une représentation qui facilite le passage entre ces deux types de description. D'autre part il permet de décrire aussi bien la structure et le comportement d'un système. Par ailleurs, ce format supporte trois niveaux d'abstraction qui sont : le niveau système, le niveau comportemental, et le niveau transfert de registres (RTL). Ce dernier est un sous-ensemble du niveau comportemental qui est lui même un sous-ensemble du niveau système. Une représentation en Solar peut être compilée de deux manières différentes :

- La première produit une spécification de logiciel utilisable par un atelier logiciel (simulation, vérification, génération de programme, etc.)

- La deuxième permet de générer une description matérielle utilisable par des outils de CAO de VLSI (simulation, vérification, synthèse, etc.).

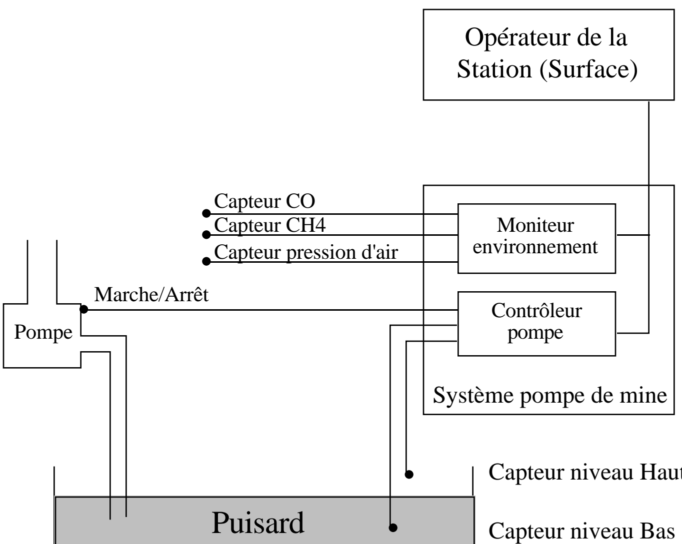
Cette représentation est donc connectée à deux types d'environnement : les environnements de spécification et de développement logiciel et les environnements de conception de circuits intégrés.

Atelier logiciel SDL, ESTELLE, LOTOS, ARGOS, StateCharts, C ...
Modèle Système SOLAR
Atelier de CAO DE VLSI VHDL, Verilog ...
Synthèse au niveau-système

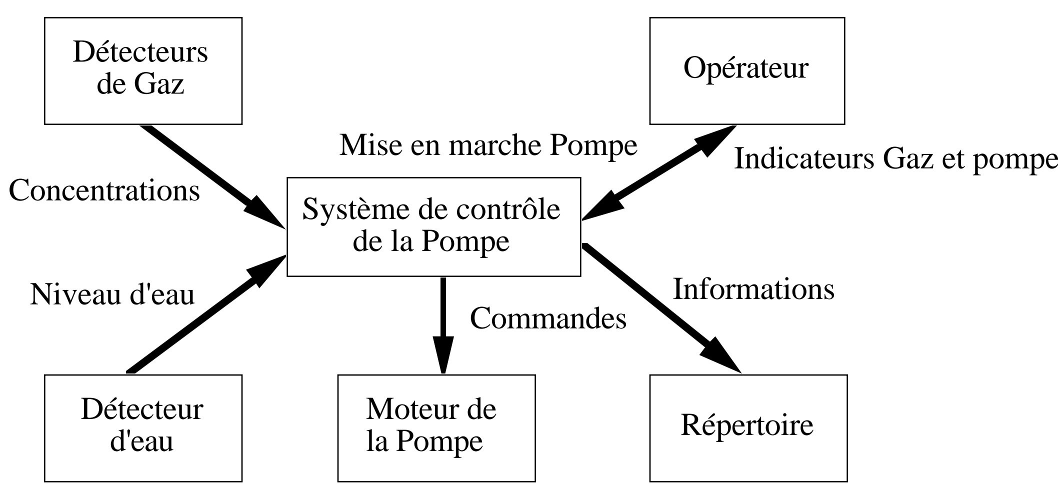
Figure 3.1. *Environnement de Solar.*

Le but est d'utiliser Solar comme forme intermédiaire durant le procédé de synthèse et de conception de systèmes mixtes logiciels/matériels. Cette synthèse est réalisée par des raffinements successifs de descriptions émanant de langages de spécification du niveau système ou de langages de description de matériel. La première étape de conception consiste à traduire le langage de spécification en Solar. Les étapes de synthèse sont effectuées sur une représentation en Solar. Ensuite, cette représentation en Solar pourra être transférée dans un langage de description de matériel tel que VHDL (pour les parties à réaliser en matériel) et un langage de programmation de logiciel tel que C (pour les parties à réaliser en logiciel).

L'exemple de la figure 3.2(a) montre une application temps réel. Il s'agit d'un système de contrôle d'une pompe à eau dans une mine [MoEv95] dont l'objet est de prévenir les inondations dans un puits de mine. L'eau se concentre au fond d'un puits de la mine et quand le niveau d'eau atteint un certain seuil (détecté par le capteur de niveau haut) la pompe à eau doit être actionnée. De manière similaire, lorsque le niveau d'eau a été suffisamment réduit (détecté par le capteur de niveau bas) la pompe doit être arrêtée. Cette pompe à eau peut être aussi actionnée (mise en marche ou arrêtée) par un opérateur humain qui se trouve dans une station de contrôle hors de la mine. N'importe quel opérateur peut actionner la pompe quand le niveau d'eau est entre les niveaux minimum et maximum tolérés. Cependant, un seul opérateur désigné, appelé superviseur, est en mesure de contrôler la pompe à eau quel que soit le niveau d'eau.



(a) Représentation schématique



(b) Représentation au niveau système par des MEFs communicantes

Figure 3.2. *Système de contrôle d'une pompe minière.*

Pour des raisons de sécurité, il existe des capteurs de concentrations du taux de méthane (CH4) et d'oxyde de carbone (CO), et de la pression de l'air dans la mine. Une indication doit être donnée pour n'importe quelle valeur critique qui exigerait alors l'évacuation de la mine. De plus, pour éviter les risques d'explosion, la pompe à eau ne doit pas être actionnée lorsque l'atmosphère contient trop de méthane. Finalement, toutes les valeurs des capteurs et l'état de la pompe à eau (marche ou arrêt) doivent être répertoriés périodiquement, pour permettre une analyse éventuelle.

Malgré la simplicité de cette application, elle comporte néanmoins des parties logicielle (e.g. le contrôleur de la pompe), des parties matérielles (e.g. les capteurs de gaz), et des parties mécaniques (la pompe à eau). Certaines parties du système sont déjà existantes (e.g. détecteurs, à usage général, de méthane et d'oxyde de carbone), et le concepteur aura simplement à les réutiliser. En revanche, il aura à concevoir les autres parties tel que le contrôleur de la pompe à eau qui doit fonctionner en tenant compte des contraintes, préalablement mentionnées, de pression de l'air et de concentrations de gaz. Toutes les parties de ce système doivent opérer ensemble, c'est pourquoi le concepteur doit modéliser toutes les parties même celles qui existent déjà. Pour chaque partie qu'il reste à concevoir, le concepteur doit la spécifier, la valider (en vérifiant sa fonctionnalité), puis la synthétiser. Comme il est proposé dans la figure 3.2(b), cette application peut être modélisée, en Solar, par un ensemble de MEFs parallèles qui interagissent entre elles à travers des canaux de communication. Chacune des MEFs peut être elle même hiérarchique et contenir la spécification de son comportement.

Les caractéristiques du modèle de MEFs étendues, qui est adopté par Solar, sont les suivantes :

- L'organisation hiérarchique des états dans une machine. Un état dans cette machine peut être lui même une nouvelle machine.
- La possibilité d'organiser une machine comme un ensemble de plusieurs machines en parallèle.
- La facilité de contrôle des processus, tel que recommencer l'exécution d'une machine à états finis ou arrêter une machine à états finis.
- Des mécanismes de manipulation des signaux asynchrones d'interruption, par exemple, pour la réinitialisation ou le changement de contexte.
- Des protocoles efficaces pour la communication entre sous-systèmes.
- L'existence de transitions globales entre MEFs, tel que sortir de l'automate A et entrer dans l'automate B à l'état B_i.

L'exemple simple suivant illustre ce modèle de machine d'états finis étendue. Il a pour intérêt de montrer, dans une application, les notions de hiérarchie, de parallélisme, et de réaction en présence des exceptions (e.g. face aux signaux asynchrones d'interruption).

Exemple 1 : Soit une machine ABC qui est constituée de deux processus séquentiels A et BC. Le processus BC est lui même constitué de deux serveurs B et C, qui opèrent en parallèle (comme indiqué par une ligne en pointillés dans la figure 3.3(a) et par deux traits

au nœud BC de la figure 3.3(b)). La flèche située au coin supérieur gauche du processus A indique que A est l'état d'entrée par défaut de la machine ABC. Les flèches entre A et BC indiquent des transitions (transfert de contrôle) entre ces états.

Le signal *Reset* s'applique à tous les processus. Lorsqu'il est émis, il force le système à l'état d'entrée par défaut (A). Cette machine est représentée dans la figure 3.3(a) par un StateCharts [Hare90]. La figure 3.3(b) représente la hiérarchie de la définition de la machine ABC sous forme d'un arbre et/ou [Mara90]. Un nœud "ou" de l'arbre représente une machine séquentielle, alors qu'un nœud "et" représente une machine parallèle.

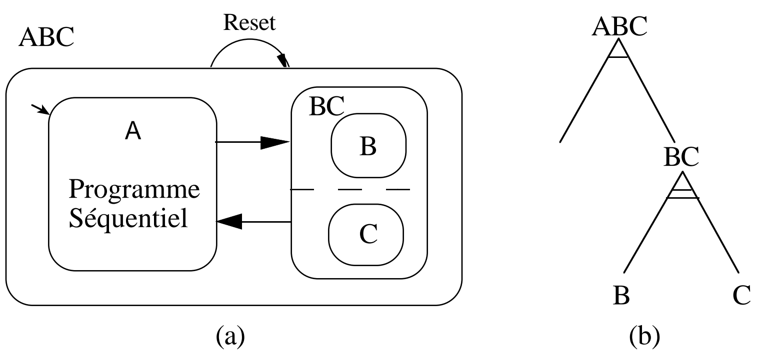


Figure 3.3. *(a) Représentation de ABC par un StateCharts*
(b) Représentation hiérarchique de ABC.

Chaque nœud de la figure 3.3(b) représente un état de la machine hiérarchique. Un état peut être l'un des cas suivants :

1. Un état feuille qui représente un processus (A, B, ou C).
2. Un ensemble d'états séquentiels (ABC).
3. Un ensemble d'états parallèles (BC).

Ces deux formalismes StateCharts et arbre et/ou seront utilisés dans le reste de ce chapitre pour illustrer les exemples. Ils sont bien détaillés dans [Tier88, Hare90] et [Mara90].

Les trois paragraphes suivants présentent les principaux concepts de Solar qui sont la table d'états, l'unité de conception et le canal de communication. L'annexe A.1 donne la description en Solar d'une application de répondeur téléphonique simplifié. Plus de détails sur Solar se trouvent dans [OBri93].

3.3. La table d'états

Le constructeur de base, dans Solar, pour la description du comportement est la table d'états (*StateTable*), qui représente une MEF étendue. Une table d'états peut être constituée d'une combinaison illimitée d'états (*States*) et de tables d'états (*StateTables*). Cette combinaison peut être séquentielle, parallèle, ou les deux en même temps. La figure 3.4 donne les attributs d'une table d'états.

StateTable

- Nom
- Actions globales (exceptions)
- État suivant par défaut
- État de réinitialisation
- État d'entrée par défaut
- Liste des variables
- Liste des noms des machines de la table d'états
- Liste des états et des tables d'états

Figure 3.4. *Attributs d'une table d'états.*

Une action globale représente une exception (action à réaliser quel que soit l'état de la machine). L'état suivant par défaut est utilisé dans le cas où l'état suivant n'est pas spécifié de façon explicite. L'état de réinitialisation indique l'état suivant à exécuter en cas de réinitialisation de la machine.

Exemple 2 : Dans ce qui suit, un exemple d'une table d'états appelée *Request* (voir figure 3.5) est donné. La machine *Request* est constituée de deux machines (états) séquentielles *init* et *send-recv*. La machine *send-recv* est elle même constituée de deux machines parallèles *send* (état) et *recv* (table d'états). Cette dernière machine (*recv*) est composée de deux états séquentiels *wait_req* et *buf_full*. L'état d'entrée par défaut de la machine *Request* est l'état *init*, alors que pour la machine *recv* (table d'états), l'état d'entrée par défaut est *wait_req*. La figure 3.5 donne la représentation hiérarchique de la machine *Request* ainsi que le StateCharts correspondant.

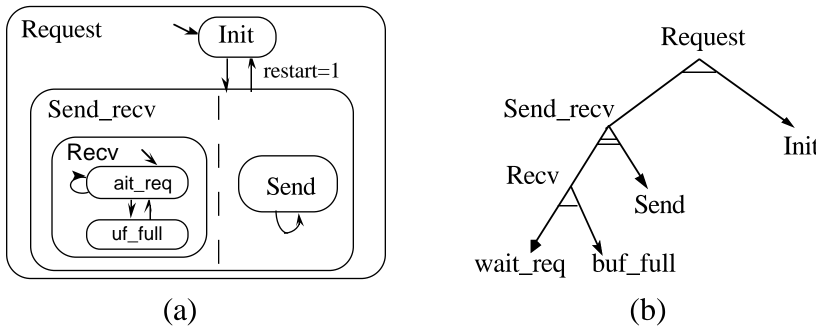


Figure 3.5. *Machine Request : (a) StateCharts de Request, (b) Représentation hiérarchique et parallèle de la table d'état : Request.*

Voyons maintenant la définition d'une transition dans le modèle de MEFs étendues.

Définition 1 : Une transition entre deux états (A et B) consiste à sortir du premier état (A) et à entrer dans le deuxième (B), indépendamment de leurs niveaux. En d'autres termes, les transitions peuvent franchir les niveaux hiérarchiques. Ces transitions inter-niveaux (selon la terminologie adoptée dans [Mara90]) sont appelées des transitions globales.

Ceci constitue un aspect très important pour les systèmes complexes. Les transitions sont permises entre deux MEFs, indépendamment de leur position respective dans la hiérarchie e.g. il est possible d'avoir dans la machine de la figure 3.5 une transition de l'état *wait_req* à l'état *init*. Une transition est décrite par le constructeur "nextstate" de Solar. Voyons maintenant la définition d'une exception dans le modèle de MEFs étendues.

Définition 2 : Une exception est une action à réaliser quel que soit l'état courant d'une machine à états finis. Cette exception est généralement provoquée soit par un événement soit par un signal asynchrone (e.g. interruption de réinitialisation ou d'arrêt).

3.4. L'unité de conception

Le constructeur unité de conception (UC) de Solar permet la structuration de la description d'un système sous forme d'un ensemble de sous-systèmes interagissants. Chaque sous-système interagit avec le monde extérieur à travers des frontières bien définies.

La figure 3.6 décrit une représentation au niveau-système contenant quatre unités de conception (UC0, UC1, UC2, et UC3). Les unités de conception peuvent être comportementales ou structurelles. Chaque unité de conception comportementale peut contenir une ou plusieurs tables d'états qui décrivent son comportement interne e.g. UC3 dans la figure 3.6. Les unités de conception structurelles comportent des instances d'autres unités de conception (comportementales ou structurelles) ainsi que les liaisons qui connectent ces unités. Par conséquent, une unité de conception structurelle peut être elle même hiérarchique (UC0 de la figure 3.6 contient d'autres unités de conception). La structure est décrite en Solar de la même manière qu'une structure (Netlist) dans EDIF [EDIF88]. La communication entre des unités de conception est réalisée de deux façons différentes. La première, à travers le concept classique de réseau dans lequel un signal transmet des données dans une ou deux directions. La deuxième, à travers des canaux de communication qui permettent au concepteur de spécifier des protocoles avec différents degrés de complexité. Les canaux de communication sont aussi considérés comme des liens (Net) de EDIF.

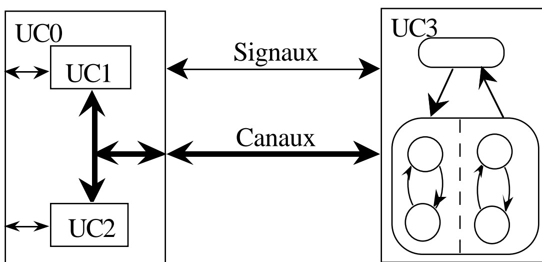


Figure 3.6. *Unités de conception.*

3.5. Le canal de communication

La communication entre sous-systèmes (unités de conception) est réalisée à l'aide de l'unité canal de Solar. Le modèle du canal regroupe les principes des moniteurs [Hoar74] et ceux de passage de messages. Il est connu sous le nom du modèle d'appel de procédures à distance (Remote Procedure Call ou RPC [Andr91, BiNe84]). Ce modèle offre une flexibilité pour représenter la plupart des systèmes de communication tout en ayant une sémantique claire. Solar se différencie des autres représentations systèmes par sa capacité d'accommoder les protocoles de communication [AnCa93], le passage de messages, les canaux lui permettant de modéliser la plupart des schémas de communication. Un canal dans Solar permet la communication entre n'importe quel nombre de processus. L'accès au canal Solar s'effectue uniquement en invoquant un ou

plusieurs de ses services (appelés méthodes dans la terminologie de Solar). Ces services représentent l'unique partie visible d'un canal.

Un canal en Solar renferme un contrôleur, et un ensemble de services (méthodes) et de ports qui constituent l'interface du canal (figure 3.7). Il est considéré comme une ressource partagée, son accès étant contrôlé par les services. Les processus distants peuvent accéder aux services et c'est au contrôleur (fonction de résolution) de gérer les conflits d'accès à ces services. Un canal est invoqué par un simple appel. La figure 3.7 représente une vue conceptuelle d'un canal.

Dans cet exemple (figure 3.7) *n* unités de conception communiquent à travers le même canal. Ce dernier offre *m* services (e.g. Émission, Réception qui peuvent être bloquantes ou non bloquantes) et comporte une fonction de résolution (contrôleur) ainsi qu'un ensemble de ports physique qui servent à synchroniser les paramètres des services avec le contrôleur du canal. L'annexe A.2 propose un exemple de canal décrit en Solar.

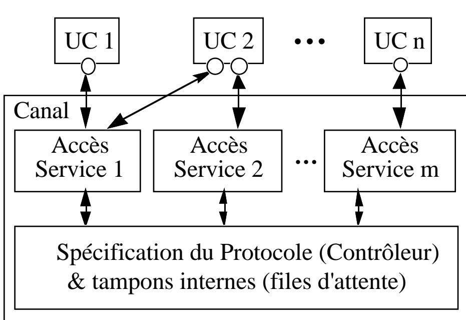


Figure 3.7. *Structure d'un canal Solar qui offre m services.*

Avec ce modèle non seulement il est possible de décrire différents types de protocoles, mais aussi la communication est séparée du reste de la conception (comportement). Ceci permet d'appliquer les algorithmes de synthèse (e.g. découpage) de manière distincte soit sur le comportement, soit sur les canaux de communication. Cette distinction permet au concepteur (ou au processus de synthèse) de sélectionner le canal approprié à son application dans une bibliothèque de canaux [OBBI93]. La description de la bibliothèque contient le comportement (type de protocole) de tous les canaux ainsi que leurs débits et la largeur de leurs interfaces.

Pour illustrer le modèle de communication de Solar, un exemple qui détaille les concepts de l'utilisation d'un canal Solar (voir figure 3.8) sera présenté.

La figure 3.8(a) montre une vue conceptuelle d'un canal qui fait le lien entre deux processeurs appelées *Host* et *Server*. Un exemple d'un tel système peut être un ordinateur personnel (PC) connecté à une imprimante. Quand le PC (*Host*) est prêt à envoyer un fichier à l'imprimante (*Server*) il transmet, à travers un canal, une requête et ensuite tout ou une partie du fichier à imprimer. Lorsque l'imprimante est prête, elle lit la requête puis exécute la commande d'impression. Les figures 3.8(b) et 3.8(c) donnent un aperçu de la description Solar. Cette description (figure 3.8) est indépendante du protocole de communication. La seule information nécessaire se résume à la possibilité d'exécuter deux services (méthodes) appelés *send* et *receive*. Comme il sera vu dans le chapitre 4, la phase d'allocation des canaux est en charge de sélectionner les canaux Solar pour pouvoir exécuter les différents services. La figure 3.8(b) donne un bref aperçu de la description en Solar de ce système. La première UC, *Host-Server*, est une vue structurelle du système. Les autres UCs appelées *Host* et *Server* sont instanciées et une liste de liens les connectent. Cette liste contient le nom du canal qui relie les deux UCs. Les autres UCs sont comportementales et représentent la fonctionnalité de l'émetteur et celle du récepteur. La figure 3.8(c) présente un extrait de la description en Solar d'un canal de communication. Cette description se compose d'une interface et d'un contrôleur (*Contents*). L'interface décrit les ports et les méthodes (services) décrivent le protocole qui utilise ces ports. Dans ce cas le canal est accessible par les services *Send* (émission) et *Receive* (réception). Cette description peut correspondre à l'abstraction d'une ou de plusieurs unités de communication existantes. Ce modèle permet de cacher les détails de réalisation d'un canal.

Durant les étapes avancées du processus de synthèse, l'unique partie visible d'un canal est l'ensemble des méthodes (services) exécutables. La réalisation du canal n'est nécessaire que pendant la phase de synthèse d'interfaces. Dans le chapitre 4, il sera expliqué comment cette étape remplace les canaux abstraits par une réalisation. Le résultat d'une telle étape est un ensemble de modules interconnectés. Les services de communication sont expansés dans les processus appelant afin de contrôler l'échange de données avec le contrôleur de communication. Autrement dit, chaque processus qui fait appel à un service va avoir une copie de ce service qui devient une procédure locale. Comme le montre la figure 3.8(d), les différents services seront enlevés du canal et il ne restera alors que le contrôleur de la communication.

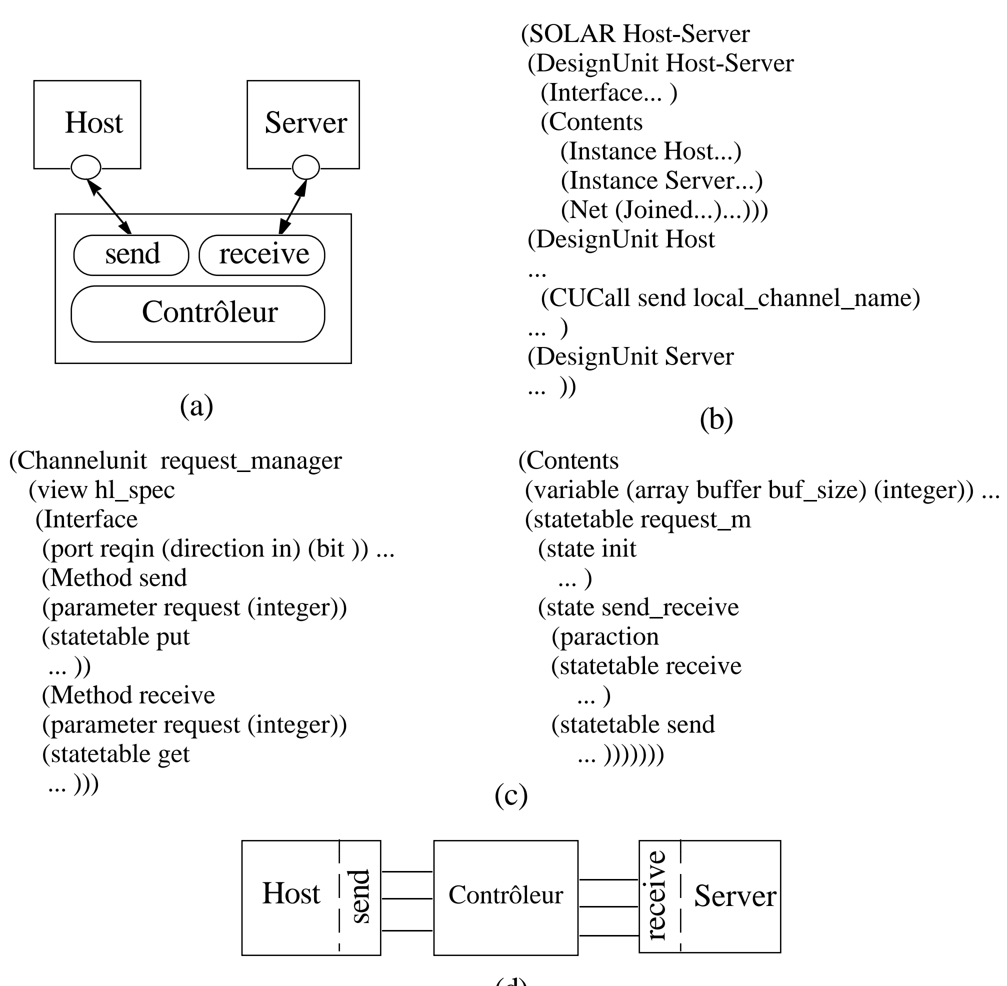


Figure 3.8. Niveaux d'abstraction d'un canal connectant deux processeurs : (a) vue conceptuelle, (b) extrait de la spécification en Solar du système, (c) description en Solar d'un canal, (d) réalisation d'une unité canal physique.

La figure 3.9 présente l'organisation d'une description Solar avec des unités de conception qui renferment des tables d'état (MEFs hiérarchiques et parallèles notées par ST) et qui communiquent à travers un seul canal Solar. Il est à noter que les unités de conception peuvent communiquer à travers plusieurs canaux Solar en même temps. Par exemple, il est possible d'imaginer une communication synchrone entre UC1 et UC2 à travers un canal et une communication asynchrone entre UC2 et UC3 à travers un autre canal. Les différentes unités peuvent donc être connectées selon différentes topologies.

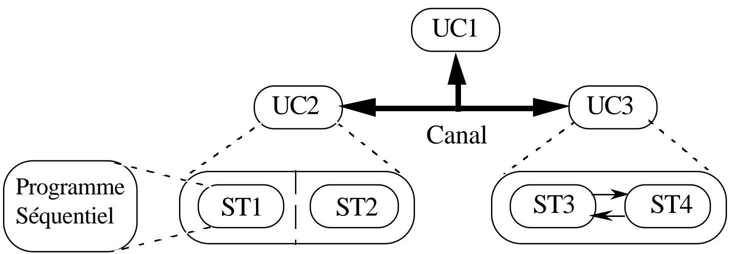


Figure 3.9. *Organisation de la structure d'une description en Solar.*

La section qui suit définit de manière informelle le modèle d'exécution de Solar.

Modèle d'exécution

Une description en Solar représente un système distribué où le comportement (les différentes fonctions) est réparti dans différentes unités de conception. Ces unités de conception s'exécutent indépendamment les unes des autres sauf au moment où elles ont besoin de se synchroniser. Cette synchronisation est possible soit à travers l'utilisation de signaux (ports) soit à travers l'utilisation de canaux de communication qui relient les différentes unités. Par exemple, une unité émettrice et une unité réceptrice qui s'exécutent en parallèle peuvent avoir besoin de se synchroniser dans le cas où elles communiquent à travers un canal de communication ayant un protocole synchrone.

Solar est composé d'un modèle de représentation des données et d'un langage textuel. Les données peuvent être représentées textuellement ou graphiquement et sont manipulées à travers une structure de données interne et une interface procédurale [JeOB94, OBri93]. La structure de données Solar a été programmée avec le langage C++. L'annexe A donne la description en Solar d'une application de répondeur téléphonique simplifié ainsi qu'un exemple de canal de communication.

3.6. Conclusion

Dans ce chapitre le modèle de représentation, Solar, a été présenté. Ce modèle est utilisé pour accommoder plusieurs langages de description provenant soit d'un atelier logiciel soit d'un atelier matériel. Ce modèle, basé sur une extension au modèle des machines d'états finis, sert non pas comme langage de spécification à la disposition des

concepteurs mais plutôt comme format intermédiaire sur lequel il est possible d'appliquer aisément les algorithmes de synthèse. Le format Solar, qui est situé entre la spécification et la réalisation, a été défini afin de s'affranchir des évolutions inévitables qui peuvent survenir sur les langages de spécification. Le principal avantage de ce format unifié est qu'il permet de développer des outils orientés réalisation (découpage, synthèse, raffinement, etc.) sans se soucier des évolutions des langages de spécification. Le format Solar est basé à la fois sur des concepts du domaine logiciel et des concepts du domaine matériel. Cette approche est donc souple et surtout évolutive par sa capacité d'accommoder de nouveaux langages qui peuvent apparaître sur le marché.

Peu de travaux ont été entrepris pour la définition d'une sémantique formelle pour Solar. Le modèle d'exécution de Solar est défini de manière qui reste informelle comme c'est le cas pour plusieurs langages de programmation; certains éléments de ces langages sont définis simplement par les outils qui les utilisent. L'absence d'une telle sémantique formelle empêche le développement d'outils de vérifications tels que la vérification de propriétés de vivacité ou l'équivalence entre deux modèles représentés différemment.

Il faut cependant noter que Solar a été conçu pour accommoder plusieurs types de langages qui peuvent se baser sur des modèles qui n'ont pas de sémantique formelle tels que les langages C ou VHDL. L'utilisation d'un modèle formel comme forme intermédiaire aurait probablement restreint l'utilisation de ce type de langages.

2.1. Introduction

Le but de ce chapitre est de présenter l'état de l'art des systèmes de conception conjointe logicielle/matérielle [Wolf93] et de la modélisation à travers les modèles les plus connus et les langages de spécification au niveau système. Il s'agit des systèmes les plus avancés dans différents centres de recherches. Les approches concernées de conception diffèrent par la spécification donnée en entrée, les domaines d'application, la méthode de synthèse, et l'architecture cible qui est considérée. Les descriptions qui sont données en entrée peuvent être soit dans un langage de haut niveau de description de matériel (VHDL, Verilog, HardwareC [KuD88], etc.), soit dans un langage de description de niveau système (SDL, StateCharts, CSP, SpecCharts, ADDL, C, etc.). Les types d'applications considérées sont très dépendants de la puissance d'expression de ces langages de spécification.

La synthèse au niveau système peut être résumée en deux activités principales qui sont le découpage logiciel/matériel et la synthèse de communication (protocoles et interfaces) [BIJe94]. Le découpage de la spécification d'un système génère un ensemble de partitions (puces, composants matériels, modules logiciels, blocs de communication) qui seront transposées sur une architecture cible. Ce découpage peut être automatique, interactif, ou manuel. Dans le cas où une spécification en entrée est transposée sur un graphe de flux de contrôle et de données, le système de découpage correspondant est automatique dans la plupart des cas [Vahi91, VaGa92]. Cependant, à cause d'une complexité supplémentaire, les systèmes basés sur un modèle de processus communicants sont soit interactifs soit manuels. La synthèse de communication permet de raffiner les interfaces des sous-systèmes communicants [WaBo93]. Par exemple, elle permet la définition des protocoles de communication ou bien les interfaces d'E/S entre plusieurs partitions.

Les approches qui seront présentées se basent sur l'un des modèles de communication suivants :

1. Modèle de communication fixe (e.g. point à point),
2. Communication à travers une mémoire partagée,
3. Communication avec un protocole qui peut avoir différents degrés de complexité.

La plupart de ces approches réalisent un découpage sur un graphe de flux de contrôle et de donnée, et la communication est limitée par la simplicité des protocoles considérés.

Pour ce qui concerne l'architecture cible utilisée pour réaliser une conception, elle devra être flexible, extensible, et devra maximiser la ré-utilisation de composants (matériels ou logiciels) existants. Globalement, deux types d'architectures peuvent être pris en compte :

1. Architecture basée sur un processeur et sur du matériel (ASICs, FPGAs),
2. Architecture distribuée et flexible, par exemple une architecture multi-puces ou bien une configuration multiprocesseurs avec une communication par passage de messages ou à mémoire partagée. Cette communication peut être véhiculée à travers un bus partagé ou bien à travers un réseau.

La section 2.2 survole les systèmes de conception conjointe logicielle/matérielle les plus connus qui sont en cours de développement dans des centres internationaux de recherches. La section suivante (§ 2.3) introduit les modèles utilisés pour la conception système. Le modèle des machines à états finis, ainsi qu'une extension possible à ce modèle seront détaillés. D'autre part, le modèle des réseaux de petri, la logique temporelle, et le modèle, *SART*, d'analyse structurée pour les systèmes temps réel seront introduits. Finalement, la section 2.4 donne les caractéristiques de base d'un langage de spécification, puis propose une étude comparative de ces langages.

2.2. Systèmes de conception conjointe logiciel/matériel : État de l'art

Pour maîtriser la complexité grandissante des systèmes et pour répondre aux critères de performances attendus, il est important d'aborder la conception de tels systèmes en terme de conception conjointe de matériel/logiciel [BoBu93, Keut94]. Cette conception agit à la fois au niveau d'abstraction le plus élevé utilisé dans la synthèse, soit le niveau système, et au niveau fonctionnel, dit souvent comportemental. La figure 2.1 présente une approche typique de synthèse. Elle part d'un langage de spécification de niveau système, puis elle réalise une répartition des fonctions entre logiciel et matériel, et ensuite elle synthétise la communication entre les blocs résultants pour qu'ils puissent communiquer. L'étape suivante est la génération de code qui est suivie de la compilation pour les parties logicielles et de la synthèse comportementale pour les parties matérielles.

La conception mixte fournit au concepteur des méthodes et outils pour explorer plusieurs possibilités de découpage du système et d'évaluer les performances des partitions [VoBI94]. L'analyse des compromis d'implémentation en matériel ou en logiciel est très utile pour permettre une réévaluation de l'architecture du système qui

s'accommode aux mieux aux contraintes diverses (coûts de développement, contraintes temps réel, fiabilité, etc.) [Fish94, Rich94].

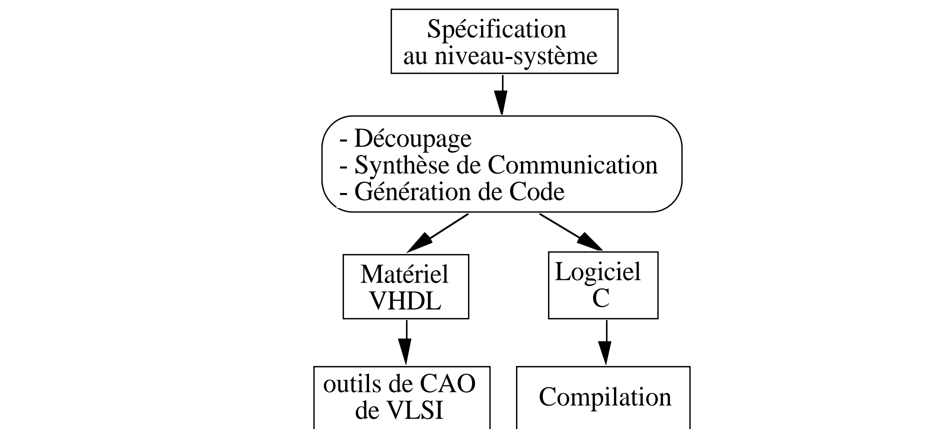


Figure 2.1. *Approche typique de synthèse.*

Par ailleurs, la combinaison, dans une même représentation, des ensembles matériels avec des ensembles logiciels nécessite un modèle ou un langage unifiant la sémantique associée à ces ensembles. Certaines méthodes de conception mixte utilisent des langages de spécification de niveau système qui font une abstraction sur la façon dont une fonction sera implantée [Goer92, Mali93]. D'autres font usage à la fois d'un langage logiciel et d'un langage matériel et essaient de les intégrer dans un même environnement [GlKr93, PuKr92].

L'un des problèmes inhérents à la conception mixte matériel/logiciel et au quel il faut souvent faire face consiste à modéliser l'interaction entre plusieurs ensembles matériels et logiciels [HePa94].

Lors du développement d'un nouveau système, le point de départ est toujours un cahier des charges qui correspond à une formulation des besoins. Les informations fournies concernent l'application dans son ensemble et expriment les objectifs souhaités. Le cycle de développement d'un système logiciel ou matériel largement accepté [Abri88, BoSt93] est présenté dans la figure 2.2.

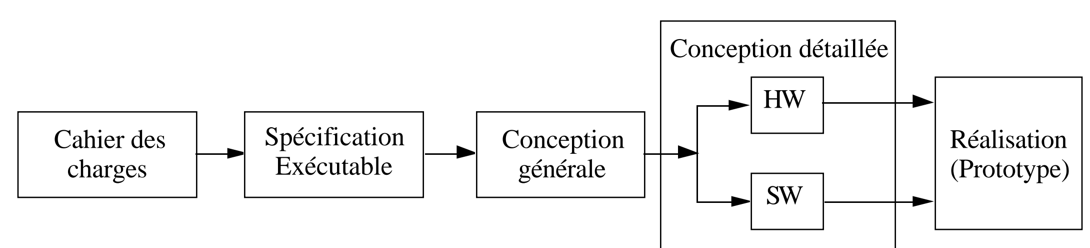


Figure 2.2. *Étapes du cycle de développement d'un système.*

• La phase de spécification

Elle correspond à la description du système à concevoir selon une vue purement externe. Les spécifications incluent toutes les contraintes auxquelles doit satisfaire ce système. Si les spécifications sont trop détaillées, elles sont alors confondues avec la conception.

• La phase de conception générale

Elle exprime la structure du système sur le plan fonctionnel. L'organisation interne et le comportement de chacune des fonctions sont explicités.

• La phase de conception détaillée

Elle a pour objectif de trouver des schémas d'implantation pour le logiciel et les structures du matériel qui sont nécessaires.

• La phase de réalisation

Elle décrit la solution finale en termes de logiciel et de matériel e.g. un ensemble de cartes électroniques. Elle conduit à un système opérationnel.

Une approche de niveau système est souvent invoquée naturellement en tant que condition nécessaire à la maîtrise de la complexité [GlVe91, GoBo94]. Dans le contexte des systèmes électroniques complexes, ceci se traduit par une approche de développement simultané du matériel et du logiciel [BIJe95]. Parmi les bénéfices d'une telle approche, on peut citer :

• Un compromis réaliste entre matériel et logiciel qui répond au mieux aux performances souhaitées.
• L'utilisation des possibilités de la technologie actuelle en ce qui concerne les répartitions matériel/logiciel.

2.2.1. Taxonomie des outils de conception logiciel/matériel

Cette section classifie les outils les plus connus pour la conception conjointe logiciel/matériel. La liste des outils n'est pas exhaustive. Ces outils sont développés soit dans des universités soit dans des centres de recherches et développements appartenant à des compagnies industrielles. Le tableau 2.1 donne une comparaison entre certains outils de conception conjointe logiciel/matériel.

Tableau 2.1. *Tableau comparatif des outils de conception logiciel/matériel.*

Outils	Spécification Système	Type d'application	Approche de conception	Architecture cible
SpecSyn Université Irvine (USA)	SpecCharts	Systèmes de contrôle et de communication	(1) Découpage matériel/logiciel, (2) Raffinement, et (3) Implantation	Machine multi-processeurs avec des ASICs
Ptolemy Université Berkeley (USA)	Schémas de blocs interconnectés (multi-langages)	Traitement du signal et systèmes communicants	(1) Découpage (2) Synthèse de matériel, logiciel, et d'interfaces, (3) Simulation hétérogène	Architecture paramétrable (parallèle ou mono-processeur)
VULCAN Université Stanford (USA)	HardwareC (extension à C)	Systèmes temps-réel	Migration du matériel au logiciel en utilisant des estimations	CPU + ASIC avec un bus et une mémoire
COSYMA Université Braunsch-weig (Allemagne)	C^x (extension à C)	Systèmes complexes	Migration du logiciel au matériel en utilisant la simulation et le profilage	CPU + ASIC avec un bus et une mémoire
CODES Siemens (Allemagne)	SDL, ou StateCharts	Systèmes de communication	(1) Découpage matériel/logiciel, (2) Conception des composants, et (3) Prototypage	Machine multi-processeurs + FPGAs + ASICs
TOSCA Italtel (Italie)	SpeedChart	Systèmes dominés par le contrôle e.g. télécommuni-cation	(1) transformations (2) Découpage matériel/logiciel, et (3) Synthèse des partitions	Une puce avec un seul processeur et plusieurs co-processeurs
SynDex (INRIA)	SIGNAL	Traitement du signal	(1) Découpage (2) Ordonnance-ment et compilation sur architectures distribuées flexibles	Architecture distribuée, processeurs communicants
Université Carnegie Mellon (USA)	CSP	Accélération d'une fonction logicielle par un accélérateur matériel	(1) Découpage matériel/logiciel, (2) Synthèse des partitions, et (3) Prototypage	CPU + carte à FPGAs et ASICs
COWARE [DeBo95] (IMEC)	POPE	Traitement du signal et systèmes communicants	(1) Synthèse d'interfaces, (2) Raffinement de processeurs	Machine multi-processeurs avec des ASICs

Les critères de comparaison sont : (1) le langage de spécification qui est utilisé en entrée, (2) le type d'applications traitées, (3) les étapes de conception suivies, et (4) l'architecture cible qui est considérée. Les langages de spécifications peuvent être soit mono-flot (par exemple un programme en langage C), soit multi-flot (par exemple un système décrit en langage SDL et composé d'un ensemble de processus communicants). L'architecture cible peut être soit monoprocesseur, soit multiprocesseurs. Une architecture monoprocesseur [HePa92, ZaLi93] se compose généralement d'une CPU avec un ou plusieurs ASICs, et éventuellement d'un certain nombre de FPGAs. Dans ce cas les composants matériels jouent le rôle d'accélérateurs pour la partie logicielle. Les architectures multiprocesseurs sont composées de plusieurs processeurs logiciels et matériels. Pour ce qui concerne les étapes de synthèse, celles-ci dépendent du langage de spécification initial et de l'architecture cible. Généralement, les différentes approches réalisent un découpage logiciel/matériel qui est basé sur des estimations, ensuite les parties logicielles sont compilées et les parties matérielles sont développées. La phase finale est l'intégration des différentes parties sur une architecture prototype. Le résultat peut servir à l'émulation ou bien au prototypage. Il peut aussi permettre d'évaluer les performances d'un système afin de pouvoir refaire des itérations des étapes de découpage et de synthèse d'interfaces précédemment réalisées. Plus de détails, concernant ces outils ainsi que d'autres outils, seront proposés en annexe B. Un environnement, appelé SynDex [Quin94], pour la conception de systèmes temps réel distribués est présenté dans [LaSe91]. Cet environnement permet, généralement, de concevoir des applications de traitement de signal. Dans cette approche la spécification initiale est décrite avec le langage SIGNAL [Quin94]. L'architecture cible est multiprocesseur et composée de processeurs de calculs et de processeurs de communications. Dans cette architecture la mémoire est distribuée entre les processeurs. Un autre environnement ambitieux de conception conjointe logiciel/matériel est décrit dans [AuBe94]. Cet environnement intègre plusieurs formalismes et outils qui ont été développés par différentes équipes de recherches.

2.3. Modélisation : État de l'art

Cette section définit les modèles les plus couramment utilisés dans la modélisation de systèmes à un très haut niveau d'abstraction. Ces modèles sont indépendants des langages de spécifications. En fait, chaque langage est basé sur un ou une combinaison des quatres modèles de base : il s'agit du modèle des machines à états finis, des réseaux de Petri, de la logique temporelle, et de l'approche d'analyse structurée, *SART*, pour les systèmes temps réel. Chacun de ces modèles a fait engendrer un certain nombre de langages de spécifications. Ces langages seront présentés en section 2.4.

Le but est de limiter la représentation d'un système complexe à une description des grandes lignes des fonctions qu'il doit exécuter, et ceci par un mécanisme d'abstraction des détails. En effet, plus on avance dans la définition détaillée, plus le raisonnement humain atteint ses limites. Il est évident qu'une seule personne ne peut connaître les détails de fonctionnement de chacun des processus d'un système dans des domaines aussi variés que le logiciel, le matériel, et la mécanique par exemple.

On constate également que le problème se complique lorsque l'on doit décrire le comportement de ces fonctions tout en considérant le cas où des anomalies peuvent se produire. Ou bien on considère qu'un incident provoque l'arrêt de la fonction, ou bien on tolère des réactions non prévues du système, ou alors on consent à traiter correctement les incidents. Par conséquent, il est souvent nécessaire de prendre en compte lors de la modélisation d'un système, à la fois les fonctionnalités requises et les réactions en cas de pannes ou d'incidents. L'arrêt total d'un système peut se concevoir pour une application isolée, mais cette situation est inacceptable pour des systèmes complexes interconnectés, risquant de provoquer des arrêts permanents d'applications ou de missions (e.g. dans le cas du domaine aérospatial). C'est la raison pour laquelle un modèle fonctionnel initial est à définir. Ce modèle initial compréhensible par l'homme servira de base à toutes les définitions allant vers plus de détails.

Dans la suite de cette section, les modèles [Andr91, Holz91] les plus couramment utilisés seront présentés :

Machines à États Finis : MEF

Le modèle des Machines d'États Finis (MEFs) [Arno90] est très connu et bien maîtrisé. Ceci est dû au fait que les propriétés de terminaison, de séquentialité des actions et de déterminisme sont bien formulées [Hart66]. L'utilisation de machines à états finis pour la modélisation permet d'assimiler le modèle général d'un système à une fonction à trois paramètres : le contrôle, la fonction, et les données. Intuitivement, il est possible de dire que les fonctions transforment l'information dans le système, que les données représentent les entrées et sorties des fonctions et que le contrôle active les fonctions suivant la séquence voulue.

L'approche de la machine à états est particulièrement appropriée à la modélisation de systèmes où une entrée particulière (événement) déclenche une série d'actions précises. La figure 2.3 représente un modèle de machine à états simplifiée pour un four à micro-onde disposant d'un bouton pour la mise sous tension et le démarrage du système, et d'une minuterie pour contrôler la durée de cuisson [Somm89].

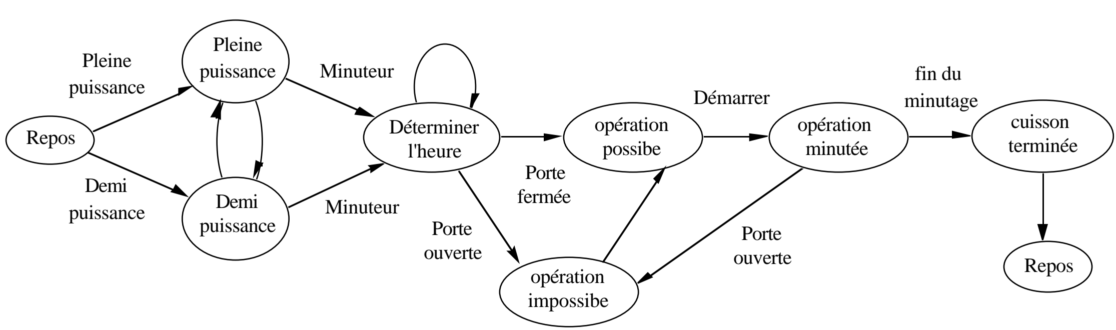


Figure 2.3. *Modèle de machine à états finis pour un four à micro-onde.*

Le problème, avec l'approche par machine à états, c'est que le nombre d'états possibles augmente rapidement. Même le modèle du four à micro-onde donne un schéma relativement complexe pour un système aussi simple. Il est donc nécessaire de trouver une alternative à cette approche lorsqu'on veut modéliser des systèmes beaucoup plus grands. Une autre façon de faire est d'étendre ce modèle afin de pouvoir représenter la hiérarchie ainsi que les actions parallèles. En effet, le modèle de MEF ne permet pas d'exprimer des activités parallèles dans une application distribuée de télécommunication par exemple. D'autres systèmes relevant du domaine des applications temps réels [DoPi93], où le temps est un paramètre critique, ne peuvent pas être séquentialisées et nécessitent donc une modélisation par un ensemble de MEFs parallèles qui interagissent entre-elles. Parmi ces systèmes, on peut citer les systèmes de contrôle pour une télécommunication par satellite. Ces systèmes peuvent être modélisés en organisant la conception dans un réseau hiérarchique de MEFs communicantes. La communication peut faire appel à des protocoles simples (e.g. communication entre un PC et une imprimante) ou même des protocoles complexes (e.g. communication entre des ordinateurs à travers un réseau Ethernet).

Pour prendre en considération les systèmes complexes, caractérisés par leur architecture parallèle, hiérarchique et distribuée, un modèle souhaitable est celui des MEFs étendues. Les extensions au modèle MEF peuvent porter sur la hiérarchie, le parallélisme et la communication entre MEFs. En effet, il a été montré dans [Wolf90a] que les systèmes complexes pouvaient être modélisés comme un réseau de MEFs communicantes. Peng, dans [Peng91], présente un bon formalisme des MEFs communicantes. Plusieurs langages de spécification système sont basés sur le modèle des MEFs. Parmi ceux-ci, figurent les langages SDL [SaTi87, SDL88], StateCharts [Hare90, DrHa89], et SpecCharts [NaVa91].

Les caractéristiques du modèle de MEFs étendues, qui est adopté dans cette thèse, sont les suivantes :

1- L'organisation hiérarchique des états dans une machine. Un état dans cette machine peut être lui même une nouvelle machine.

2- Une machine, prise dans ce modèle, peut être composée de plusieurs machines en parallèle.

3- La facilité de contrôle des processus, tel que recommencer l'exécution d'une MEF, ou arrêter une MEF.

4- Des mécanismes de manipulation des signaux asynchrones d'interruption.

5- Des protocoles efficaces pour la communication entre sous-systèmes.

6- Transitions globales entre MEFs, tel que sortir de l'automate A et entrer dans l'automate B à l'état B_i.

La modélisation par machine à états reste, néanmoins, une bonne manière de représenter un système indépendamment du langage de spécification. Plus de détails sur ce modèle de MEFs étendues seront donnés dans le chapitre 3.

Réseaux de Petri

Les réseaux de Petri [Bram83] sont des outils puissants qui permettent la représentation des systèmes à événement discrets. Cependant, ils ne permettent pas de décrire la structure de données utilisées et s'attachent plutôt aux aspects de contrôle du comportement d'un système. Ils évoluent par transitions qui correspondent à des événements en allant à des places correspondant aux activités et aux états d'attentes d'un système. L'état d'un système correspond alors à un marquage qui associe à toute place une valeur. A chaque événement, une transition est associée et elle est franchissable sous certaines pré-conditions qui dépendent du marquage initial des places d'entrée. Franchir une transition s'appelle opération de mise à feu dans la terminologie des réseaux de Petri.

Les réseaux de Petri s'appuient sur une théorie mathématique bien définie [Moal81, Atam94]. Ainsi, il est possible de déterminer a priori les caractéristiques et les propriétés du système modélisé, en particulier déterminer si le réseau est vivant (sans blocage).

Logique Temporelle

La logique temporelle [Ostr89] fournit des opérateurs qui permettent de formuler des assertions sur le comportement d'un système en terme d'états courants, et futurs. Elle permet d'étendre la logique des prédicats en qualifiant temporellement les formules de cette logique. Une spécification en logique temporelle est donc une suite de prédicats donnant les contraintes sur les états par lesquels passera le système.

La logique temporelle peut être utilisée comme langage de spécification selon certains auteurs [Pnue77]. Deux approches ont été proposées : la logique temporelle linéaire et la logique temporelle arborescente. Chacune d'elles propose ses propres définitions, et sa propre liste d'opérateurs qui caractérisent son pouvoir expressif. En logique linéaire, une exécution est modélisée par la suite des états successifs du système. Avec une logique arborescente, il est possible de spécifier des systèmes non déterministes. Pour prendre en compte le non-déterminisme, la logique arborescente permet à un état d'avoir plusieurs états successeurs possibles.

SART

SART (*Structured Analysis for Real-Time systems*) [Jaul90, HaPi88, WaMe85] est une extension de la méthode SA (Structured Analysis) développé par T. DeMacro [Lapl92]. Bien que SA soit basée sur une décomposition hiérarchique avec masquage d'informations à chaque niveau, elle ne permet pas de modéliser les aspects de contrôle et événementiels ainsi que les processus concurrents. SART comble ces insuffisances et permet de décrire des automates à états finis. Ceux-ci permettent de représenter la dynamique du système (son comportement) et de spécifier comment le système répond à des événements externes. La méthode consiste à définir tout d'abord un diagramme des flots de données, puis de décrire chaque élément du diagramme. Ce modèle a été repris par Calves [Calv93] pour la spécification de matériel.

2.4. Langages du niveau système : État de l'art

La spécification fait suite à une expression de besoins. C'est la phase initiale du développement d'un système. Elle consiste d'une part, à prendre en compte les exigences fonctionnelles et les contraintes de réalisation, et d'autre part précise le schéma directeur qui sera choisi pour la réalisation du système. Le fait de spécifier consiste à répondre à la question "que doit faire le système ?", ce qui est différent de la manière de concevoir qui répond à la question "comment doit-on le faire ?". Cette section est inspirée des études

menées par d'autres membres du groupe de synthèse au niveau système du laboratoire TIMA [Romd92].

2.4.1. Taxonomie des langages de spécification

Il existe actuellement d'excellents langages pour décrire d'une part le logiciel , et d'autre part le matériel. Par exemple, la communication entre processus peut être spécifiée à l'aide d'un langage tel que ESTELLE [ISO87], LOTOS (*LOgical Temporal Ordering Specification*) [Loto89, BoBr87, BrSc87, NaBa86], SDL (*Specification and Description Language*) [SaTi87, SDL88], ESTEREL [BeCo84], etc. Le cheminement des signaux à travers des portes logiques peut se décrire plus facilement à l'aide de langages de description de matériel tel que VHDL (*VHSIC Hardware Description Language*) [Vhdl88, WoM93], ou Verilog [ThMo91]. La question suivante peut être posée : Lequel de ces langages choisir en vue d'une intégration dans un environnement de spécification mixte matérielle ou logicielle ?

En fait, le choix d'un langage repose généralement sur trois facteurs principaux :

1- La puissance d'expression : Ce facteur concerne la facilité de spécifier les concepts systèmes. Il dépend donc du domaine des applications à considérer.

2- La puissance d'analyse : Ce facteur concerne la facilité de construire des outils autour du langage; il dépend du type de traitement envisagé. Dans le cas où l'on veut réaliser des outils de vérification formelle par exemple, il est important de choisir un langage formel. Par contre, s'il s'agit de réaliser des outils de documentation et d'archivage, il est important de choisir un langage lisible.

3- Le facteur commercial : Ce facteur concerne "l'utilité" du langage. Il dépend donc de l'environnement d'utilisation.

2.4.1.1. Puissance d'expression

Dans la suite de cette section, les critères qui permettent de décider le degré de difficulté ou de facilité pour modéliser un comportement donné seront détaillés.

• **Les aspects de concurrence et de communication**

La concurrence permet de spécifier dans un système des activités parallèles. Elle implique la communication et la synchronisation entre ces différentes activités. Le langage

de spécification doit fournir les moyens de spécifier le parallélisme [Lamp89]. Les communications synchrones ou asynchrones dépendent du problème traité et sont exprimées suivant deux modèles différents [Miln83]. Le modèle asynchrone [Mart90] est propre aux systèmes temps-réel, alors que les fonctions internes du système sont généralement des modèles synchrones [BeCo87]. Il est souhaitable qu'un langage de spécification supporte ces deux formes de représentation. Les langages ESTEREL, LUSTRE, SIGNAL [Halb93], et StateCharts [Hare87] sont synchrones. Par contre, les langages SDL, ESTELLE, et les réseaux de petri sont asynchrones. D'un autre côté, les langages CSP et LOTOS utilisent des mécanismes de synchronisation, en l'occurrence, par rendez-vous. Pour ce qui concerne la prise en compte des exceptions, certains langages tels que ESTEREL, SDL ou StateCharts permettent de spécifier des exceptions. Une spécification SDL se présente comme un ensemble de machines d'états interconnectées et qui communiquent entre-elles. Elle comporte la définition de la structure du système (interconnexion des machines) ainsi que le comportement (dynamique) de chacune d'elles. L'annexe D propose deux applications décrites en SDL. ESTELLE est un langage pour la spécification des protocoles de communication. Néanmoins, la communication avec ESTELLE présente des limitations pour spécifier le parallélisme puisqu'un processus père ne peut s'exécuter avec aucun de ses descendants.

• **Description des données**

L'aspect description des données ne doit pas se limiter à la description d'objets, il doit décrire aussi les opérations sur ces objets. Un point qui doit être également cité concerne l'intégrité, donc la protection et le contrôle d'accès à ces données. Un langage de spécification doit permettre la définition incrémentale des données par enrichissement ou dérivation [Brui87] ainsi que les mécanismes nécessaires pour rendre visibles ou non ces données. Une approche formelle est très importante pour décrire les objets et les opérations sur ces objets sans supposer une réalisation particulière. Les langages SDL et LOTOS permettent de décrire des types abstraits de donnée. Ainsi, il est possible de définir des types d'objets et les opérations possibles sur ces objets. D'autres langages, tels que ESTELLE et ESTEREL, supportent des données du même type que ceux du langage PASCAL. En revanche, les réseaux de petri ne permettent pas de décrire des données.

• **Contraintes du temps-réel**

La spécification d'un système doit prendre en compte l'environnement dans lequel le système évolue. Un système n'est jamais isolé, les contraintes et les interactions entre

ses éléments sont liées à son environnement physique, englobant, ou couplé. Les interfaces entre le système et son environnement doivent être modélisées [HaPn83]. Les contraintes temps réel permettent de se référencer à une échelle de temps, et d'imposer des contraintes sur les performances du système. Ces possibilités donnent les moyens pour spécifier un temps d'exécution, un temps de réaction, ou bien des délais pour l'exécution de certaines activités du système. Un langage de spécification doit supporter ces concepts. Le langage LOTOS ne permet pas de spécifier les contraintes temps réel. ESTELLE permet de décrire uniquement des délais. D'un autre côté, le langage SDL permet de spécifier des temporisateurs. D'autres langages tels que StateCharts ou ESTEREL permettent de déclarer des compteurs d'événements.

2.4.1.2. Puissance d'analyse

Cette section classifie les langages de spécification en fonction du degré de difficulté nécessaire à l'analyse et à la vérification d'un modèle.

• **Définition formelle : Syntaxe et sémantique**

La principale propriété d'un langage formel est d'avoir une syntaxe qui favorise son analyse et une sémantique non-ambigue ce qui rend unique son modèle d'interprétation. Cependant, si un langage est trop formel, il nécessitera un effort d'apprentissage. Par contre, un langage peu formel sera sujet à des interprétations diverses. Les langages Z [Spiv89] et B [BoSt93] possèdent une sémantique formelle. D'autre part, les langages SDL, LOTOS, ESTELLE, et les réseaux de petri possèdent un modèle d'interprétation formel. Par exemple, LOTOS est un langage de description formelle des protocoles et des systèmes distribués. Le langage ESTEREL a une syntaxe formelle.

• **Facilité d'analyse**

Cet aspect concerne l'évaluation des propriétés qui facilitent l'analyse d'un langage de spécification. Il est possible de citer parmi ces propriétés : la cohérence, la consistance, l'absence de blocage, l'équivalence, etc. Pour qu'un langage de spécification soit analysable, sa syntaxe doit être clairement définie. De plus, afin de vérifier et analyser un langage, il est souhaitable d'avoir une sémantique formelle ou un modèle d'interprétation formel.

2.4.1.3. Arguments commerciaux

Cette section classifie les langages de spécification, par des arguments commerciaux, en fonction de trois critères qui sont : (1) la clarté du modèle qui permet de faciliter la lisibilité et la mise en oeuvre, (2) la disponibilité d'outils autour des langages, et (3) les efforts de standardisation.

• **<u>Lisibilité / Mise en oeuvre</u>**

Les spécifications d'un système s'adressent à une population assez large d'utilisateurs, par conséquent elles doivent être compréhensibles, faciles à lire, même pour des non-initiés. Une forme graphique peut être très simple à lire alors qu'une forme textuelle s'adresse plutôt aux traitements par des outils automatiques. Il est souhaitable pour un langage de spécification de posséder ces deux formes de représentation.

• **<u>Disponibilité des outils</u>**

La disponibilité des outils est capitale pour l'utilisation d'un langage de spécification. Ces outils peuvent être multiples permettant : la vérification syntaxique, la vérification sémantique, la simulation, l'analyse dynamique du flot de contrôle et de données, la validation, la détection de blocage et de vivacité, et la vérification du processus de raffinement en effectuant des transformations [BeJe95]. Il existe des outils commercialisés pour les langages SDL, StateCharts, et VHDL. En revanche, les langages LOTOS, ESTELLE, ESTEREL, et B ne disposent que d'outils universitaires ou industriels très limités. L'environnement ESTEREL offre aux utilisateurs un outil de simulation. Cette simulation est facilitée par une interface graphique et interactive.

• **<u>Standardisation</u>**

Il est attendu d'un langage de spécification d'avoir une structure reconnue et sous le contrôle d'un organisme international. Son modèle d'interprétation doit être unique, seul l'organisme de standardisation est à même de le faire valoir. Parmi les standards ISO, il y a les langages ESTELLE, LOTOS, VDM (*Vienna Development Method*) [Jone90], et Z. D'autres langages tels que SDL et VHDL sont des standards CCITT et IEEE 1076 respectivement.

2.4.2. Comparaison des langages de spécification

Le tableau ci-dessous (tableau 2.2) montre les points forts et points faibles des différents langages de spécification analysés. La classification est donnée pour chaque

critère et varie de l'absence d'étoile (qui est éliminatoire dans ce cas) jusqu'a trois étoiles (meilleur choix pour le critère en question).

Tableau 2.2. *Comparaison des langages de spécification.*

	Lotos	SDL	Estelle	Esterel	VDM, Z et B	StateCharts	Réseau de Petri	SART	VHDL	C
Concurrence	***	***	**	* S	***	* S	***	**	***	
Description des données	***	**	*	*	**	*		***	*	*
Communication	**	***	*	*	***	*	*		*	
Temps-réel		**	*	***		***	***	*	**	
Analyse/ Syntaxe	*	**	*	***	***	***	***	*	***	***
Standard	** ISO	*** CCITT	** ISO	*	** ISO	*	*	*	*** IEEE	**
Lisibilité/ Sémantique	* T	*** T+G	* T	** T	*	*** G	*** G	** G	* T	** T
Disponiblité d'outils	*	**	*	*		**	*		***	***
Total	13	20	10	13	14	15	15	10	17	11

Notations :

S : Synchrone, T : Textuel, G : Graphique, ISO : Organisme de Standardisation Internationale, IEEE : Organisation de normalisation dans le domaine électronique, et CCITT : Organisme de normalisation des protocoles de communication. La ligne "Total" donne la somme des étoiles. Il ne s'agit que d'une indication sur les caractéristiques d'un langage.

LOTOS, qui est à la fois procédural et déclaratif, est un langage exécutable. Sa caractéristique principale est sa capacité d'abstraction des détails d'implémentation. SDL est un langage à la fois procédural et déclaratif. Il a été conçu pour spécifier les systèmes de télécommunication et il convient particulièrement bien aux systèmes distribués comportant des MEFs étendues étroitement liées. Les descriptions ESTELLE sont procédurales, proches de celles d'un langage de programmation du type PASCAL. Il s'agit plutôt d'un bon langage pour la spécification de programmes, mais mal adapté pour spécifier un système. ESTEREL est un langage synchrone qui possède une sémantique mathématique rigoureuse. Il est un langage événementiel où un événement est l'émission d'un ou de plusieurs signaux en même temps. VDM est basée à la fois sur la théorie des

ensembles et la logique des prédicats. Ses points faibles restent la non prise en compte des aspects de concurrence, il est verbeux et de nombreux outils sont manquants. Z est un langage déclaratif basé sur la logique des prédicats et très ressemblant à VDM. Il est basé lui aussi sur la théorie des ensembles. B, qui est un langage déclaratif, se décompose en une méthode et un environnement. Malgré son environnement d'utilisation encore insuffisant, B est cependant dans un état d'industrialisation bien supérieure à VDM et Z. Les StateCharts sont un formalisme visuel pour les systèmes complexes. Les transitions entre états s'effectuent à la suite de l'apparition d'événements. Dans StateCharts, un signal émis est diffusé instantanément à l'ensemble du système. VHDL permet de décrire un système, à réaliser en matériel, sous plusieurs formes : comportementale, et structurelle. Un modèle présente deux parties : une vue externe (spécification d'entité) et une vue interne (architecture). D'autre part, VHDL permet des descriptions séquentielles et parallèles [BaEc93]. C est un langage de programmation qui se situe entre l'assembleur et les langages évolués du type PASCAL. Son principal avantage est de posséder une bibliothèque de fonctions préétablie. En revanche, il nécessite un noyau d'exécution supplémentaire pour gérer le parallélisme. C++ permet la programmation orientée objet, c'est à dire d'aborder une structure hiérarchisée avec des notions d'héritages et d'associer à chaque objet des déclarations de variables et des fonctions qu'il peut utiliser. Ce langage n'autorise pas la communication et le parallélisme, il ne convient donc que pour la spécification de logiciels.

Cette analyse semble indiquer qu'actuellement quatre langages de spécification sont prédominants dans ce domaine : SDL, VHDL, StateCharts, et Réseaux de Petri.

Choix de SDL

Compte tenu de cette étude, le choix pris au cours de ces travaux s'est porté logiquement sur SDL pour les raisons suivantes qui semblent essentielles :

• Il est le langage de spécification des systèmes de télécommunications standardisé par le CCITT. Il fait actuellement l'objet de recommandations Z. 100 et Z. 110 du CCITT [SDL88].

• Il permet de décrire de façon hiérarchisée les systèmes, mais aussi de modéliser les fonctionnalités d'un système par des machines à états finis communicantes entre elles par des signaux ou des messages.

• Enfin, une extension à ce langage qui inclut les aspects orientés objets est en cours d'élaboration, ce qui permet d'envisager plus tard une extrapolation à ce type d'approche.

2.5. Conclusion

Dans ce chapitre différents modèles, langages, et systèmes de conception mixte logiciel/matériel ont été présentés. Les modèles les plus couramment utilisés ont été décrit brièvement. La taxonomie des langages de spécification ainsi qu'une comparaison entre ces langages ont été effectuées. Les langages de spécification de haut niveau ont été évalués en fonction de leur puissance d'expression, leur puissance d'analyse, et leurs arguments commerciaux. Ensuite, les justifications qui ont conduit, au cours de ces études, au choix du langage SDL ont été données. Ainsi, pour l'approche de conception logiciel/matériel, développée au cours de cette thèse, la spécification initiale d'un système à concevoir sera donnée en langage SDL.

1.1. Motivations

Au cours de ces dernières années, les techniques de développement de circuits intégrés (ASIC) ont connu une très grande évolution. En même temps, on assiste actuellement à une évolution sans précédent dans les architectures matérielles notamment à base de processeurs (e.g. RISC) ou d'éléments programmables (e.g. FPGA, PLD). Le progrès de la technologie de fabrication de circuits combiné à l'évolution des architectures matérielles ont fait que la conception de circuits est en train de migrer vers la conception de systèmes entiers.

Le mot *système* n'est pas un terme très précis — un système pour une personne donnée peut être considéré comme un composant pour une autre personne. Cette difficulté linguistique est surpassée en définissant un système par un ensemble (collection) de sous-systèmes qui sont en interaction. Un exemple de système peut être un contrôleur du tableau de bord d'un avion. Ce système peut consister en un ensemble de cartes (sous-systèmes) qui comportent des microprocesseurs standards et des circuits annexes (e.g. ASICs, mémoires, alarmes, etc.).

En pratique, un système est souvent formé par une collection d'ensembles matériels (circuits) et logiciels (code exécutable) en étroite interaction. L'utilisation conjointe de processeurs à usage général dont les performances atteignent aujourd'hui des niveaux très élevés, et de circuits spécialisés nécessite de nouvelles méthodes de conception [BIJe95]. Ces méthodes doivent en particulier permettre au concepteur de faire un découpage d'un système afin de répondre au mieux aux performances requises, et assister le concepteur à trouver un meilleur compromis entre une réalisation logicielle et une réalisation matérielle. La conception conjointe de logiciel/matériel contribue à la réalisation de systèmes fiables à travers une validation appliquée au modèle du système. Ce type de conception permet aussi d'augmenter la productivité (dans un milieu industriel) en permettant la conception simultanée des parties logicielles et des parties matérielles. Dans les prochaines années, avec l'émergence de nouvelles applications, ces méthodes vont prendre une importance de plus en plus grande. Les technologies de réalisation de circuits intégrés en CMOS (0,35 micron) sont déjà disponibles. Les années à venir sont donc très prometteuses pour ce domaine.

Pour faire face à la complexité grandissante des systèmes électroniques qui peuvent comporter à la fois du logiciel et du matériel (e.g. dans le domaine de télécommunication ou aérospatial) et pour répondre aux critères de performance attendus, il est souhaitable de réaliser la conception de telles applications à un niveau d'abstraction

élevé. Ceci permettra de réduire le nombre de composants que le concepteur aura à manipuler. Ici, “abstraction” signifie un codage symbolique du comportement avec un certain langage sans avoir à se préoccuper des détails d'implémentation ou de la structure physique des composants matériels. En élevant le niveau d'abstraction jusqu'au niveau système, le concepteur peut traiter des applications plus complexes avec une maîtrise totale des technologies. Il peut choisir la technologie qui réalise le mieux (en tenant compte des critères de performance) chaque partie de son système. Généralement, les parties critiques qui nécessitent un temps de réponse rapide (e.g. temps réel) sont réalisées en matériel par des ASICs ou des FPGAs; les parties moins critiques sont réalisées en logiciel par des structures micro-informatiques classiques (e.g. micro-processeurs plus co-processeurs). D'autre part, au niveau système le concepteur n'a plus à se soucier de la description explicite de niveau physique tel que le niveau transfert de registres.

Bien que la conception de systèmes contenant à la fois du logiciel et du matériel ne soit pas nouvelle (le logiciel a toujours été exécuté sur du matériel) le développement conjoint du logiciel et du matériel est un domaine de recherche récent où presque tout reste à découvrir. Il s'agit de permettre une automatisation, quasi complète, de la synthèse à partir de spécifications au niveau système sur une architecture mixte. Cette architecture cible peut être un circuit, une carte, ou un réseau de processeurs distribués (certains peuvent être des microprocesseurs standards ou spécifiques). Le coût des circuits intégrés dédiés a maintenant baissé de manière significative, tandis que sa vitesse de fabrication a augmenté. Pour des circuits simples il est maintenant possible d'avoir une idée précise de la puce finale en quelques semaines en partant d'une spécification de haut niveau. La non-utilisation de processeurs standards peut se justifier par des contraintes d'Entrée/Sortie (E/S), de vitesse, ou encore de consommation d'énergie.

1.2. La conception conjointe de logiciel/matériel

La stratégie de conception et de développement conjoint de systèmes contenant à la fois des composants matériels et logiciels nécessite plusieurs étapes. Il s'agit de permettre la spécification, la validation (exhaustive par exemple), la synthèse, la simulation (ou co-simulation) et la réalisation de systèmes sur des architectures cibles.

La synthèse au niveau système comporte deux phases principales qui sont le découpage et la synthèse de la communication (voir figure 1.1). La première consiste à découper la spécification des besoins (fonctionnalités) d'un système en un ensemble de

partitions qui seront transposées sur une architecture cible. La deuxième phase réalise la synthèse d'interfaces entre des composants communicants. Plus précisément, elle permet de définir des protocoles de communication et des interfaces d'E/S entre les partitions. Naturellement, c'est un processus itératif. Du fait que les processus des diverses partitions qui constituent un système peuvent coopérer, le concepteur doit coordonner les communications entre processus. Les mécanismes de coordination des processus assurent par exemple l'exclusion mutuelle lors de l'accès à des ressources partagées. Lorsqu'un processus est en train de modifier une ressource partagée, les autres processus ne peuvent pas en faire de même.

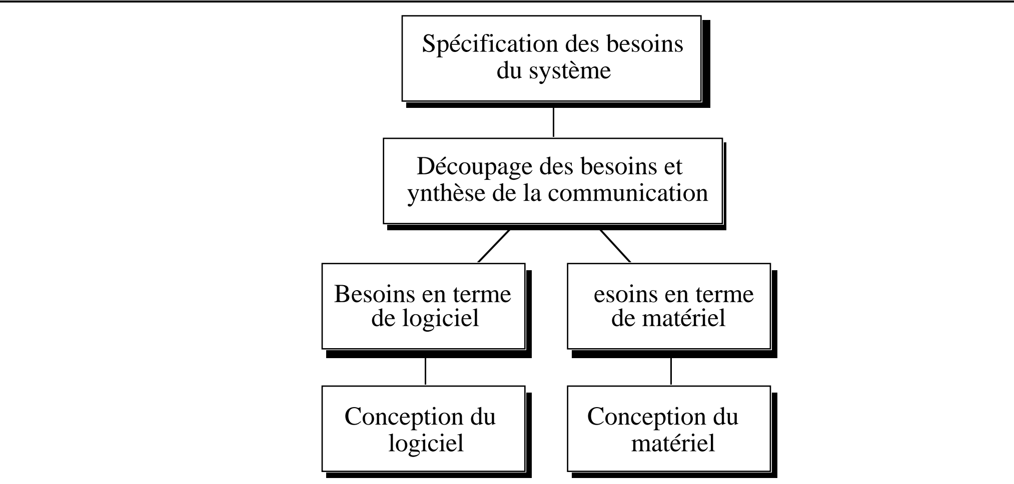


Figure 1.1. *La stratégie de conception d'un système.*

Deux types de conception conjointe peuvent être distingués; le premier appelé génération de micro-code consiste à transposer des algorithmes sur des architectures spécifiques; le deuxième réalise la conception de systèmes distribués composés de modules (ou processus) communicants où chaque module est à réaliser soit en logiciel soit en matériel. Dans le premier cas, l'architecture cible est un processeur standard existant (fixe) ou spécifique. Dans le second cas, l'architecture cible est composée de composants matériels et de composants logiciels. Une telle architecture peut être un ou plusieurs circuits, une ou plusieurs cartes, ou bien un réseau de processeurs distribués. Ces processeurs peuvent être homogènes, c'est à dire avec des configurations identiques, ou hétérogènes donc avec des configurations différentes. Ces deux types de conception conjointe de logiciel/matériel constituent deux domaines de recherches qui, bien que complémentaires, restent différents. Le premier peut être considéré comme le croisement des évolutions des techniques de microprogrammation et des recherches sur la synthèse de haut niveau. Le second peut être considéré comme le rapprochement des méthodes de conception de circuits intégrés et des méthodes de développement de logiciels.

1.3. Objectifs

Le but de ces travaux est de maîtriser la complexité croissante des systèmes ainsi que d'accélérer le processus de conception. En fait, en raison des pressions du marché (*time-to-market*), pour être le premier sur le marché avec un produit qui est en cours de normalisation, il est nécessaire de réduire le temps de conception (e.g. quelques semaines au lieu de plusieurs mois). Plus encore, la réutilisation de composants ou circuits existants peut aider à réduire considérablement le cycle de conception. Une approche de développement qui favorise la réutilisation est donc souhaitable.

L'objectif principal de cette thèse est de mettre en oeuvre une approche de conception de systèmes mixtes logiciels/matériels communicants entre-eux. Le processus complet de conception mixte comporte les points suivants :

- Spécification et validation du système mixte au niveau fonctionnel; il s'agit d'utiliser des langages de spécification de niveau système tels que SDL, StateCharts, Estelle, Lotos, Esterel, CSP ou SML.

- Découpage des spécifications en partitions logicielles et matérielles.

- Modélisation et synthèse des communications, c'est à dire développement des éléments (interfaces et composants) de communication entre les différentes technologies de réalisation choisies pour chaque partition qui résulte du découpage.

- Spécification hétérogène exécutable (e.g. co-spécification ou co-simulation C/VHDL).

- Génération d'architectures pour systèmes mixtes.

- Maintenance des systèmes mixtes; il s'agit de privilégier la maintenabilité en la considérant parmi les caractéristiques les plus importantes d'un système. Ceci est d'autant plus vrai pour des applications en milieu hostile (e.g. spatial) là où il peut y avoir des risques pour la survie du systèmes et même des équipages humains (e.g. domaine de l'aviation).

Bien que tous ces points soient présents, cette thèse est plus particulièrement dédiée aux trois premiers points. Actuellement, trois thèses, qui ont démarré depuis un an, sont en cours. Elles constituent une continuation aux travaux présentés dans cette thèse. La première porte sur le développement d'outils d'estimations pour le découpage de spécifications, afin d'aider le concepteur à affecter les partitions logicielles ou

matérielles à des processeurs spécifiques. La deuxième thèse a pour objet de valider la méthodologie développée dans cette thèse à travers une application ATM. La troisième thèse est consacrée à la génération et à la co-simulation C/VHDL, et à l'ordonnancement de plusieurs programmes en C (partitions logicielles) sur un seul processeur. Tous ces travaux font partie d'un projet, appelé COSMOS, dont la réalisation est un travail collectif. Plusieurs personnes ont contribué à ce projet et continuent à travailler.

1.4. Contribution

La principale contribution de ces travaux de thèse est le développement d'une approche pour la conception conjointe de systèmes logiciels/matériels. Le point de départ pour la conception de tels systèmes qui sont le plus souvent complexes consiste en une spécification exécutable validée de façon exhaustive ou par un simulateur de niveau système (celui de SDL). Ensuite, un découpage fonctionnel en logiciel et en matériel est choisi par le concepteur. Cette étape du processus de conception de systèmes consiste à partager les fonctions du système entre logiciel et matériel. Il s'agit de transformer la spécification initiale en processus concurrents qu'il est possible de simuler par une co-simulation (C/VHDL). Pendant le découpage, le concepteur aura à choisir la réalisation de chaque partition résultat du découpage. Une partition peut être un circuit ASIC, un circuit programmable (e.g. FPGA), ou un programme logiciel exécutable par un microprocesseur (éventuellement sur une carte). Le concepteur aura aussi à définir (modéliser, choisir et réaliser) un ensemble de schémas et protocoles de communication entre les différents processus communicants du système en question.

Un outil de découpage interactif de spécifications décrites au niveau système a été développé. Cet outil offre au concepteur le choix d'un certain nombre d'opérations de transformations et de décompositions d'une spécification initiale en un ensemble de partitions qui concordent avec une architecture distribuée. Ces transformations sont réalisées par le concepteur en suivant un schéma de raffinement incrémental.

Afin de faciliter l'automatisation du processus de conception au niveau système, un modèle de représentation de haut niveau a été utilisé. Cette solution a l'avantage de rendre la conception indépendante des langages utilisés. Le modèle en question, appelé Solar, a été conçu pour représenter à la fois les concepts systèmes (sémantique logicielle) et ceux utilisés par les langage de description de matériels (sémantique matérielle). L'approche choisie est donc basée sur les concepts et non sur les langages.

Le domaine d'application visé par les travaux de la présente thèse est celui des systèmes hétérogènes communicants. Ces systèmes sont initialement décrits sans spécification des détails de réalisation physique ni du comportement ni de la communication. Les étapes de synthèse vont permettre d'une part de fixer les protocoles de communication, qui permettront de satisfaire les performances, et d'autre part de choisir la technologie de réalisation de ces protocoles. Le modèle de description utilisé permet la manipulation d'une large gamme de protocoles de communication allant du simple échange de signaux (*handshake*) aux protocoles complexes tels que les ATMs ou l'Ethernet. On s'intéressera plutôt aux applications dominées par le contrôle. Ainsi, les langages et les méthodes relatives au flot de données (*Data Flow*) ne seront pas traités dans cette thèse.

1.5. Plan de la thèse

Le chapitre 2 présente l'état de l'art pour la modélisation de systèmes et les langages de spécification de niveau système. Les caractéristiques des langages sont détaillés. Ensuite, une comparaison entre ces langages de spécification sera réalisée. Les principales approches de conception conjointe de logiciel/matériel seront aussi décrites brièvement afin de dégager les points communs et les différences entre-elles.

Le chapitre 3 détaille le modèle utilisé pour la synthèse de systèmes mixtes logiciels/matériels. Ce modèle est une extension au modèle de machines à états finis et sert comme format intermédiaire unifié entre la représentation du logiciel et la représentation du matériel. Ce format permet d'accommoder les aspects d'une sémantique logicielle et ceux d'une sémantique matérielle.

Le chapitre 4 présente l'approche de conception conjointe de logiciel/matériel qui a été développée au cours de cette thèse. Il décrit une vue globale des différentes phases d'une conception en partant d'une spécification jusqu'à sa réalisation sur une architecture qui comporte des processeurs et des composants matériels.

Dans le chapitre 5, la phase de découpage et de répartition des fonctionnalités d'un système entre le logiciel et le matériel sera présentée. Cette phase décrit l'approche suivie pour le découpage ainsi que les primitives permettant de réaliser un découpage. La méthode de découpage adoptée se base sur l'application interactive de ces primitives de découpage et de transformation. Un exemple complet d'un système découpé ainsi que l'outil graphique permettant de réaliser un découpage seront présentés.

Dans le chapitre 6, la phase de synthèse de la communication sera présentée. Les différentes étapes nécessaires à savoir la sélection des protocoles et la synthèse d'interfaces seront détaillées. Les principaux résultats obtenus après une formulation du problème seront montrés. Les algorithmes développés pour sélectionner des protocoles de communication seront aussi présentés.

Le chapitre 7 présente les conclusions et les perspectives attachées aux travaux menés dans le cadre de cette thèse. Les aspects qui commencent à devenir d'actualité et qui n'ont pas été traité dans cette thèse seront discutés.

Plusieurs détails relatifs aux travaux menés dans le cadre de cette thèse seront présentés en annexes. L'annexe A donne quelques exemples décrits dans le format intermédiaire, Solar, qui sert à tous les outils de synthèse au niveau système. Ensuite, dans l'annexe B, une étude des outils (universitaires ou développés dans des centres de recherches et développements) pour la conception logiciel/matériel sera décrite. L'annexe C est consacrée à la présentation des différents algorithmes des primitives de découpage et de synthèse d'interfaces. Enfin, l'annexe D donne deux exemples décrits en SDL. Le premier est un système émetteur/récepteur et le deuxième est un bloc simplifié de télécommande d'un satellite.

Les travaux de cette thèse ont été en partie financés par France-Telecom (projet SOLIST) et la Direction Générale de l'Armement (DGA) (projet de maintenance de systèmes électroniques).

(b) Modèle du programme C généré 56
4.11. Co-simulation de l'application Émetteur/Récepteur basée sur IPC 57
4.12. Génération d'architecture 58
4.13. Bloc télécommande d'un satellite 60
5.1. Le découpage au niveau-système 64
5.2. Approche de synthèse 65
5.3. Partitionnements possibles :
(a) contrôle distribué, (b) contrôle centralisé, (c) compromis 68
5.4. Le système PARTIF 71
5.5. Méthode inhérente à PARTIF 74
5.6. Exemple d'application de la primitive Move
(a) Représentation hiérarchique, (b) Représentation par un StateChart 77
5.7. Fusion de machines 78
5.8. Découpage d'une machine AB
(a) Représentation hiérarchique, (b) Représentation par un StateChart 79
5.9. La machine A' 79
5.10. La machine ABC à découper 80
5.11. La machine ABC après le découpage
(a) Représentation hiérarchique, (b) Représentation par un StateChart 80
5.12. Machine AB avant l'application du Cut 82
5.13. Machine AB après l'application de Cut(AB) 82
5.14. Menu graphique de l'outil PARTIF 83
5.15. Structure du répondeur de téléphone 85
5.16. Hiérarchie de la machine behaviour du contrôleur
(a) Copie d'écran, (b) Représentation par un StateChart 86
5.17. Découpage du système de contrôle du répondeur téléphonique
(a) Résultat de Move, (b) Résultat de Merge,
(c) Résultat de Split, (d) Résultat de Cut 89
6.1. Approche de synthèse de la communication :
(a) processeurs communicant au niveau conceptuel, via des canaux abstraits,
(b) communication au niveau spécification après la synthèse des protocoles,
(c) communication au niveau physique après la synthèse d'interfaces 95
6.2. Étapes de la synthèse de communication du système Prod/Cons 96
6.3. Sélection de composants de communication 98
6.4. Sélection de composants de communication avec l'algorithme ALLOC 103
6.5. Alternatives à la synthèse d'interfaces 109

6.6. Machine AB avant et après l'application de Map :
(a) machine AB avant l'application de Map,
(b) machine AB après l'application de Map 110
6.7. Répondeur téléphonique avant et après l'application de Map :
(a) répondeur téléphonique avant l'application de Map,
(b) répondeur téléphonique après l'application de Map.............................111
6.8. Système Émetteur/Récepteur..112
6.9. Bibliothèque des unités de communication pour l'algorithme SELECT..........112
6.10. Alternatives d'allocation par l'algorithme SELECT 113
6.11. Synthèse d'interfaces du résultat d'allocation par l'algorithme SELECT.........114
6.12. Bibliothèque des unités de communication pour l'algorithme ALLOC 115
6.13. Arbre de décision obtenu par l'algorithme ALLOC 116
6.14. Alternatives d'allocation par l'algorithme ALLOC 117
6.15. Synthèse d'interfaces du résultat d'allocation par l'algorithme ALLOC..........118

Chapitre 4: Méthodologie de conception dans COSMOS 41
4.1. Introduction 42
4.2. Saisie de spécifications 43
4.3. Découpage au niveau système 46
4.3.1. Définition du problème de découpage 46
4.3.2. La méthode interactive dans COSMOS 48
4.3.3. Les primitives 49
4.4. La synthèse de communication 50
4.4.1. La synthèse de protocoles 50
4.4.2. La synthèse d'interfaces 51
4.5. Le prototypage virtuel 52
4.5.1. La génération de code VHDL comportemental 53
4.5.2. La génération de code C 56
4.5.3. La co-simulation matérielle/logicielle C/VHDL 57
4.6. La génération d'architecture 58
4.7. Conception orientée vers la maintenance 59
4.8. Conclusion 61

Chapitre 5: Le découpage de systèmes au niveau système 63
5.1. Introduction 64
5.2. L'outil Partif 67
5.2.1. Le principe 70
5.2.1.1. Les contraintes de conception 71
5.2.1.2. La bibliothèque de modèles de communication 72
5.2.1.3. Estimation de coût 72
5.2.2. La méthode de découpage 73
5.2.3. L'affectation logiciel/matériel 75
5.3. Les primitives de découpage 76
5.3.1. La primitive Move 76
5.3.2. La primitive Merge 77
5.3.3. La primitive Split 78
5.3.4. La primitive Cut 80
5.4. Réalisation 82
5.5. Exemple d'application 84
5.6. Conclusion 90

Chapitre 6: La synthèse de communication 93
6.1. Introduction 94
6.2. Les étapes de la synthèse de la communication 95
6.3. La synthèse de protocoles 97
6.3.1. Formulation du problème 97
6.3.2. Algorithmes 98
6.3.2.1. Algorithme SELECT 100
6.3.2.2. Algorithme ALLOC 103
6.4. La synthèse d'interfaces 108
6.4.1. Formulation du problème 108
6.4.2. La primitive Map 109
6.5. Exemple 111
6.5.1. Application de l'algorithme SELECT 112
6.5.2. Application de l'algorithme ALLOC 114
6.5.3. Comparaison des algorithmes d'allocation 118
6.6. Conclusion 119

Chapitre 7: Conclusions et perspectives 121
7.1. Conclusions 122
7.2. Perspectives 124

Bibliographie 127

Publications personnelles 141

Glossaire 145

Annexe A: Exemples en SOLAR 151
A.1. Description en Solar d'un répondeur téléphonique 152
A.2. Exemple de canal Solar 159

Annexe B: Étude des principales approches de conception de logiciel/matériel 161
B.1. Introduction 162
B.2. Méthodologie de l'université de Californie/Berkeley 162
B.3. Méthodologie de l'université de Californie/Stanford 164
B.4. Méthodologie de l'université de Californie/Irvine 165
B.5. Méthodologie de l'université de Carnege Mellon 166

B.6. Méthodologie de Siemens 168
B.7. Méthodologie de l'université de Tübingen 169
B.8. Méthodologie de l'université de Cincinnati.....................................170
B.9. Méthodologie de l'université de Linkoping 170
B.10. Méthodologie de l'université de Manchester 171
B.11. Méthodologie de Italtel...172
B.12. Méthodologie de l'université de Braunschweig...............................172

Annexe C: Algorithmes des primitives...175
C.1. Introduction 176
C.2. Primitive Move 177
C.3. Primitive Merge...178
C.4. Primitive Split...179
C.5. Primitive Cut...180
C.6. Primitive Map..181

Annexe D: Exemples en SDL graphique...183
D.1. Système émetteur/récepteur 184
D.2. Bloc télécommande d'un satellite..213

Remerciements

Je tiens à remercier :

Monsieur Bernard Courtois, Directeur de recherche au CNRS et Directeur du Laboratoire TIMA, de m'avoir accueilli dans l'équipe d'architecture des ordinateurs.

Monsieur Guy Mazaré, Professeur et Directeur de l'ENSIMAG - Institut National Polytechnique de Grenoble, pour m'avoir fait l'honneur de présider ce jury.

Monsieur Ahmed Amine Jerraya, chargé de recherche au CNRS et Directeur de l'équipe "System-Level Synthesis", pour son amitié, pour les nombreuses discussions, et pour les encouragements et l'aide qu'il m'a donné ce qui a permis un bon déroulement à cette thèse.

Messieurs Patrice Quinton, Professeur à l'IFSIC (Institut de Formation Supérieure en Informatique et Communication - Université de Rennes 1), et Ivo Bolsens, Directeur de la division VSDM à l'IMEC en Belgique, de m'avoir fait l'honneur d'être rapporteurs de cette thèse, et membres du jury.

Monsieur Jean-Louis Lardy, Responsable du Groupement Circuits Intégrés pour les Télécommunications au CNET de Grenoble, d'avoir accepté d'être membre du jury.

Je ne saurais oublier dans mes remerciements tous les membres de l'équipe "System-Level Synthesis" à TIMA : Mohamed Abid, Mohamed Ben Mohamed, Élisabeth Berrebi, Wander Cesario, Adel Changuel, Jean-Marc Daveau, Hong Ding, Oyvind Gillingsrud, Philippe Guillaume, Polen Kission, Clifford Liem, Gilberto Marchioro, Abdellatif Mtibaa, François Naçabal, Richard Pistorius, Vijay Raghavan, Maher Rahmouni, Mohamed Romdhani, et Carlos Valderrama.

Mes amis Ali Bouzouita et Jamel Nouiri qui m'ont tellement fait rire, ainsi que tous les autres qui ont participé de près ou de loin à cette thèse; Mohamed Aichouchi, Mokhtar Boudjit, Abderrazak Jemai, Khalil Kchouk, Goro Wakana, et surtout Kevin O'Brien.

Philippe Guillaume et Ali Bouzouita qui m'ont consacré une partie de leur temps précieux pour m'aider à préparer ce manuscrit.

Le reste du Laboratoire dans son ensemble, notamment Hicham Boutamine, Patricia Chassat, Ricardo Duarte, Corinne Durand-Viel, Isabelle Es salhiène, Omar Kebichi, Salvador Mir, Imed Moussa, Renato Ribas, Kholdoun Torki, André Vacher, et j'en oublie sûrement.

Je tiens encore à remercier ma très chère mère qui prie toujours Dieu pour le succès de son fils et qui ne saura jamais à quel point elle a apporté à cette thèse.