HAL
open science

Un environnement graphique pour le langage SIGNAL

Patricia Bournai, Paul Le Guernic

To cite this version:

Patricia Bournai, Paul Le Guernic. Un environnement graphique pour le langage SIGNAL. [Rapport de recherche] RR-2040, INRIA. 1993. inria-00074631

HAL Id: inria-00074631

https://inria.hal.science/inria-00074631

Submitted on 24 May 2006

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Un environnement graphique pour le langage SIGNAL

Patricia Bournai, Paul Le Guernic

N° 2040

septembre 93

PROGRAMME 2

Calcul symbolique,
programmation
et génie logiciel

Rapport
de recherche

1993

ISSN 0249-6399

Un environnement graphique pour le langage SIGNAL

Patricia Bournai, Paul Le Guernic

Programme 2 — Calcul symbolique, programmation et génie logiciel
Projet EP-ATR

Rapport de recherche n° 2040 — septembre 93 — 41 pages

Résumé : Ce rapport présente un environnement de programmation polymorphe pour la conception d'applications temps réel décrites dans le langage SIGNAL*. Cet environnement offre à l'utilisateur des représentations graphiques et textuelles des structures du langage. Ces représentations peuvent être conjointement utilisées lors de la construction ou la "lecture" du programme, l'utilisateur dispose de formes adaptables à un niveau d'observation.

Une expression SIGNAL peut être considérée comme un ensemble de composants (représentés par des rectangles) munis de points de connexion (les ports représentés par des triangles) joints par des liens (suite de segments connexes).

La même expression SIGNAL est également un système d'équations sur suites de valeurs représenté par un terme construit à l'aide des opérateurs du langage (définition de variable, composition, changement de nom...).

Ces deux formes dénotent en fait un même objet, représenté plus abstraitement par un arbre que l'on munit d'attributs géométriques.

Le langage SIGNAL étant présenté par ailleurs, on se propose de développer les différents aspects de l'interface d'édition conçue en accord avec sa sémantique.

Mots-clé : interface graphique orientée "bloc-diagramme", édition graphico-textuelle, routage automatique, gestion d'arbres.

(Abstract: pto)

*Ce travail a bénéficié du soutien du CNET

Unité de recherche INRIA Rennes
IRISA, Campus universitaire de Beaulieu, 35042 RENNES Cedex (France)
Téléphone : (33) 99 84 71 00 – Télécopie : (33) 99 38 38 32

A graphical environment for the SIGNAL language

Abstract: This report presents a programming environment for the design of real-time applications with the SIGNAL language. This environment, developed at IRISA with the support of the CNET, allows the user to have graphical and textual representations of the language structures. These representations may be used together during the building or the "reading" of the programm.

A SIGNAL expression may be considered as a set of components (represented as boxes) with connection points (ports represented with triangles) joined by links (a succession of connex segments). The same SIGNAL expression is also a system of equations on series of values; this system is represented by a phrase built with the language operators (definitions of variables, composition, renaming, etc.)

The two representations can be unified by a representation as a tree with graphical attributes.

After a short presentation of the SIGNAL language, we develop the different aspects of the graphical interface.

Key-words: "block-diagram" interface, graphical and textual edition, automatic routing, tree management.

Table des matières

1 Introduction **3**

2 Le langage SIGNAL **4**

3 Environnement d'édition **6**
3.1 Le menu . . . 7
3.1.1 Les actions rémanentes . . . 8
3.1.2 les actions de modification immédiate . . . 9
3.1.3 Les actions de visualisation . . . 9
3.1.4 Les paramètres de visualisation . . . 9
3.2 L'interface système . . . 9
3.3 La fenêtre texte . . . 11
3.4 Les informations dans l'environnement . . . 12

4 Les objets graphiques **13**
4.1 Les boîtes . . . 13
4.2 Les ports . . . 14
4.3 Les liens . . . 15
4.4 Les halos . . . 16
4.5 Description de la structure des objets . . . 16
4.5.1 Structure graphique des objets . . . 17
4.5.2 Visibilité des objets . . . 18

5 Les fonctions d'édition **18**
5.1 Construction géométrique des objets . . . 18
5.1.1 Le programme . . . 18
5.1.2 Les sous-modèles d'un modèle (ou du programme) . . . 19
5.1.3 Les expressions de processus . . . 20
5.1.4 Les liens . . . 20
5.2 Modification de la structure géométrique . . . 21
5.2.1 Déplacement . . . 21
5.2.2 Changement de taille . . . 22
5.2.3 Justification . . . 22
5.3 Actions indépendantes de la forme de l'objet . . . 22
5.3.1 Copie . . . 22
5.3.2 Destruction . . . 23
5.3.3 Récupération . . . 23
5.4 Association d'un texte à un objet à représentation géométrique . . . 23
5.4.1 Expressions de processus . . . 24

5.4.2 Propriétés d'un port 24
5.4.3 Déclarations locales à un modèle 25

6 Mise en oeuvre **25**
6.1 La structure de données 25
6.2 Conversion de la forme géométrique à la forme alphabétique 26
6.2.1 Héritage 27
6.2.2 Synthèse 28
6.2.3 Synthèse des déclarations 29
6.3 Routage 29
6.3.1 Choix d'un routage 29
6.3.2 Principe de l'algorithme 30

7 Mémento des commandes possibles sous l'éditeur **32**
7.1 Rappel sur l'utilisation des boutons de la souris 32
7.2 Les différentes commandes 33
7.3 Paramétrisation de l'éditeur graphique 38
7.4 Touches accélératrices 38
7.5 Lancement de l'éditeur de SIGNAL 39

8 Conclusion **40**

1 Introduction

La facilité d'utilisation d'un logiciel, et par conséquent la réussite de sa diffusion, sont très dépendantes de la qualité de son interface utilisateur. Si les représentations graphiques sont très utilisées, et depuis longtemps, dans des domaines proches de l'électronique ou de l'automatique (conception de circuits, réseaux de Petri, Grafcet, blocs diagrammes...) , ce n'est que plus récemment qu'elles ont été employées dans les systèmes voire les langages comme interface offerte à l'utilisateur.

L'interface présentée dans ce rapport permet une représentation à la fois textuelle et graphique du langage SIGNAL [1], [6], [10]. L'intérêt majeur de la représentation graphique repose sur l'accès global à l'information portée, accès global le plus souvent permis par une représentation iconique. Cependant, l'utilisation des icônes ne doit pas conduire à leur multiplication et d'autre part, la forme textuelle d'un langage de programmation reste une nécessité ; elle permet une vision analytique des objets décrits, complémentaire de la vision globale. La possibilité de passer de l'une de ces perceptions à l'autre est tout-à-fait souhaitable.

Un acquis dans le domaine des langages de programmation est la possibilité de construction modulaire de programmes et d'imbrication de structures. Il ne faut pas, dans le passage à une représentation graphique, en perdre le bénéfice. Des environnements comme X-Window offrent une structure arborescente correspondant au concept de hiérarchies de boîtes bien adapté à la représentation visuelle d'objets structurés [5] [12]. Néanmoins, l'approche X-Window doit être adaptée pour la définition de la forme graphique d'un langage de programmation. D'une part, la vision partielle offerte d'un objet est relative au plan ; pour un langage structuré, il nous a semblé préférable d'adopter une approche duale relative à la profondeur (analogue aux holophrastres de CENTAUR [3]). D'autre part, les fenêtres de X-Window sont conçues comme interfaces entre utilisateurs et applications et il n'existe pas de communication directe entre fenêtres ; à l'inverse, les objets d'un programme SIGNAL communiquent et il est nécessaire de représenter ces communications. Ainsi, alors qu'une fenêtre X-Window est strictement un arbre, un programme SIGNAL est une hiérarchie de graphes et, à un niveau donné, il existe des contraintes topologiques qui doivent être respectées (si l'on veut représenter graphiquement ces communications).

Après une brève présentation du langage SIGNAL, nous en décrivons l'environnement multi-fenêtres, puis la structure de la représentation graphique. Quelques indications sur la mise en oeuvre sont ensuite données. Enfin, en dernière partie, est donné un récapitulatif de toutes les commandes possibles de l'éditeur graphique.

2 Le langage SIGNAL

SIGNAL est un langage temps réel orienté flots de données [8]. Nous en donnons ici une description sommaire (voir [2] pour une spécification complète). Pour écrire un algorithme en SIGNAL, le programmeur utilise deux catégories d'objets : les signaux et les processus.

- Les signaux
  Les objets de base du langage SIGNAL sont les signaux assimilables à des séquences LUCID resynchronisées [13]. Chaque signal peut être désigné par un nom, par exemple x, et est caractérisé par la suite ordonnée (x_t) finie ou infinie de ses valeurs, ainsi que par son horloge. Ces valeurs sont typées et leur type définit le type du signal. L'horloge d'un signal associe à chaque indice $_t$ de la suite ordonnée de ses valeurs, la date (dans une référence temporelle virtuelle) à laquelle (x_t) est défini : une valeur est non persistante. Les horloges permettent de calculer des relations temporelles entre les divers signaux.

- Les processus

  Un processus définit une relation entre des signaux d'entrée et de sortie ; cette relation est exprimée par une propriété invariante sur les éléments des suites. Les propriétés portent sur les valeurs (par exemple, "à tout instant, x est égal à la somme de y et de z"), mais aussi sur la présence ou l'absence d'un signal (par exemple, " à tout instant, x est présent si y est absent et c est présent et possède la valeur vrai.").

  Exemple : pour construire le signal x comme étant, à chaque instant, le signal zx augmenté de la constante 1, il suffit de définir le processus **x := zx + 1** représenté graphiquement par la figure 1. Le port d'entrée du processus a pour nom zx et le port de sortie x.

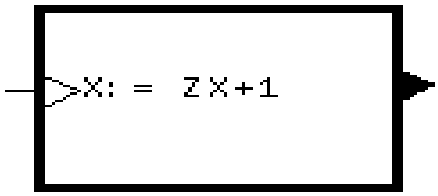

Figure 1 : processus plus1

Le lien avec la représentation des graphes à flots de données est claire : les signaux circulent sur les arcs et les actions sont effectuées au niveau des noeuds du graphe. Cette dualité de forme (textuelle et graphique) des processus résulte de la définition de SIGNAL en terme d'algèbre de graphes [7].

Il existe deux catégories de processus en SIGNAL : les générateurs et les processus composés.

1. Les générateurs (par exemple, celui de la figure 1) permettent de définir un signal de sortie au moyen d'opérateurs agissant sur les signaux d'entrée (par exemple, toutes les fonctions arithmétiques ou logiques).
2. Les processus composés obtenus au moyen d'expressions sur processus. Graphiquement, un processus est alors représenté par un réseau statique orienté de sous-processus interconnectés par des liens via leurs ports d'entrée et de sortie respectifs. L'opérateur de base est noté "|".

Etant donné deux processus P et Q, leur composition, notée **P** | **Q** construit un nouveau processus en connectant les ports de sortie de P (respectivement de Q) aux ports d'entrée de Q (respectivement de P) possédant le même nom. Le même signal est diffusé sur les ports d'entrée de même nom. Cet opérateur n'est défini que si P et Q n'ont pas de sorties de même nom, ce qui correspondrait à des définitions multiples de variables. En supposant que P et Q dénotent respectivement les systèmes de définition de signaux S et T, alors P|Q dénote l'union de S et de T. La figure 2 décrit graphiquement la composition de deux processus P et Q dont les signaux sont représentés par des ports nommés.

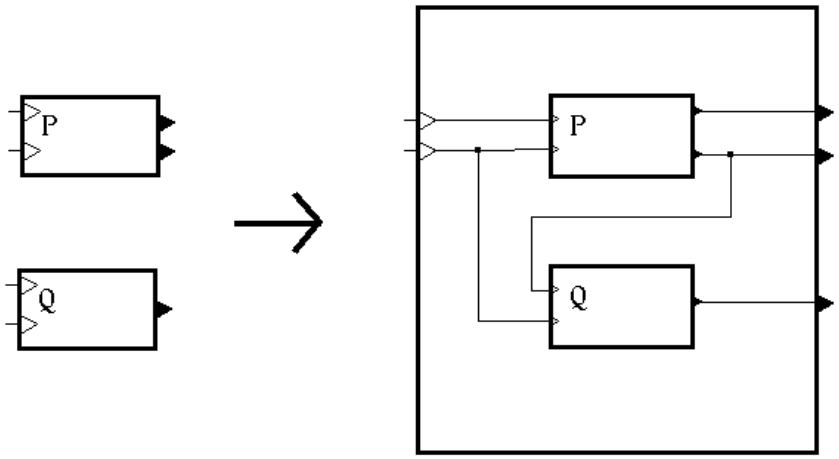

Figure 2 : composition de 2 processus

Cet opérateur est associatif et commutatif :

$$(A|B)|C \equiv A|(B|C)$$

$$A|B \equiv B|A$$

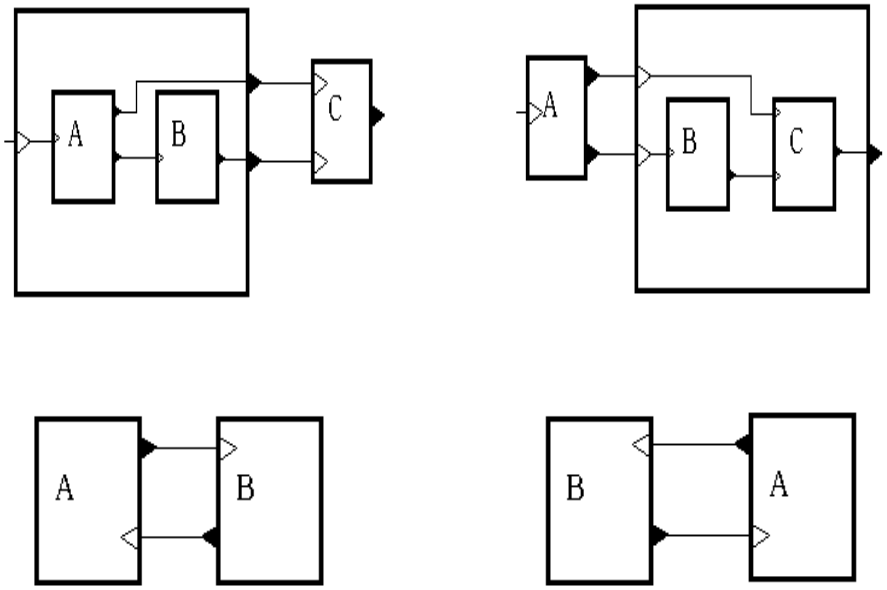

Figure 3 : associativité et commutativité de la composition

soit L'approche adoptée pour définir SIGNAL permet de déduire directement une représentation graphique des expressions : les propriétés géométriques d'un objet (coordonnées, taille) n'interviennent que dans cette représentation.

- Les modèles de processus

  Un modèle de processus P est une abstraction d'une expression E ; il est formé d'un nom, d'une liste de paramètres formels, des listes éventuellement vides de ses entrées et sorties. L'expression E peut utiliser des variables locales, des modèles de processus locaux qui lui sont associés et des modèles de processus définis visibles par le modèle dont P est sous-modèle (si P n'est pas le programme).

3 Environnement d'édition

L'interface est réalisée au-dessus du gestionnaire de fenêtres X-Window et elle utilise le clavier et la souris à trois boutons :

- le bouton gauche (MSL) permet de désigner les objets sur lesquels doit s'effectuer une action.
- le bouton central (MSM) permet de descendre dans l'arborescence qui représente le programme ; dans certains cas, il peut être associé à la touche Control (ou CTRL) du clavier.
- le bouton droit (MSR) permet de remonter dans cette même arborescence.

A l'appel de l'environnement SIGNAL, l'écran est constitué de deux fenêtres (Figure 4) :

1. la fenêtre principale (la plus grande) composée de deux sous-fenêtres :
   - un bandeau de menus déroulants et de boutons (qu'on appellera, dans la suite du rapport, fenêtre M),
   - une fenêtre d'édition graphique (qu'on appellera fenêtre D)
2. la fenêtre d'édition alphabétique, indépendante de la fenêtre principale, qui peut être déplacée, agrandie, cachée ou iconifiée. Elle est elle-même composée :
   - d'une fenêtre texte munie d'ascenceurs où l'utilisateur va rentrer du texte Signal (qu'on appelera fenêtre T)
   - d'un bouton **confirm** qui permet à l'utilisateur d'indiquer qu'il a fini de rentrer son texte.

D'autres fenêtres (affichage d'informations ou des messages d'erreurs, appel du compilateur SIGNAL) apparaissent lorsque nécessaire.

3.1 Le menu

Les fonctions de l'éditeur sont organisées de manière hiérarchique. Le bandeau de menus regroupe ces fonctions, soit dans des rubriques (menus déroulants), soit directement par des boutons (**undo**, **refresh**, **text_window**, **help**). Les rubriques sont :

- **file** pour les fonctions gérant l'interface avec le système (**save**, **load**, **compile**, **reset**, **quit**)
- **edit** pour les fonctions d'édition qui modifient la structure des programmes (**create**, **delete**, **copy**, **glue**, **break**)
- **draw** pour les fonctions modifiant uniquement l'aspect visuel des programmes (**justify**, **move**, **move**, **halo**)
- **visual** qui permet de modifier certains paramètres de l'affichage (**grid**, **police**, **details**, **names**)

Il existe en permanence un objet courant (cf 5.1) sur lequel s'applique une action courante. L'action courante est choisie par désignation MSL d'un bouton du menu. Certaines actions, dites rémanentes, peuvent ne pas être désignées à chacune de leur invocation.

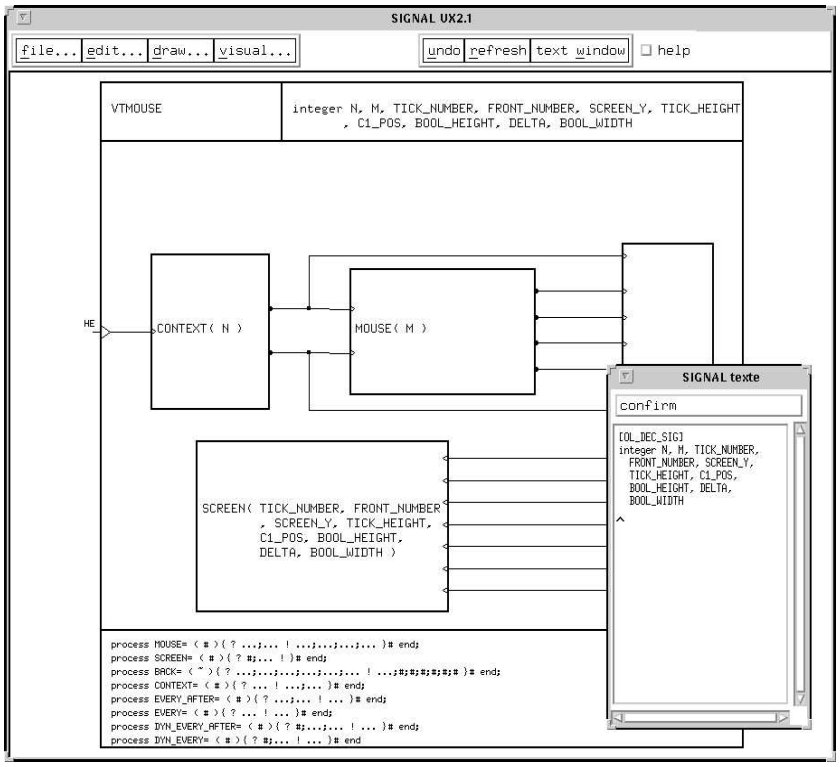

Figure 4 : les fenêtres d'édition graphique et textuelle

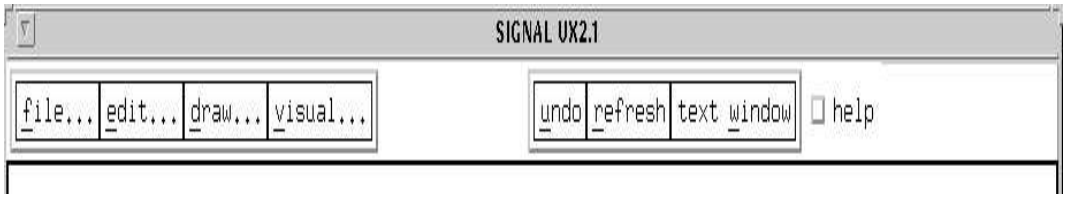

Figure 5 : le bandeau de menus

3.1.1 Les actions rémanentes

Lorsqu'une action rémanente est l'action courante, elle reste l'action courante pour toutes les désignations d'objets jusqu'à son remplacement par la désignation d'une nouvelle action courante parmi les actions rémanentes ou les actions de modification immédiate (cf 3.1.2). Les actions suivantes sont rémanentes : **create**, **copy**, **delete**, **move**, **size**, **justify**, **halo**, **glue**, **break**. En cours d'édition, le nom de l'action courante rémanente, si elle existe, est

affichée en permanence dans le bandeau de menus, en caractères gras, entre la rubrique **visual** et le bouton **undo**.

3.1.2 les actions de modification immédiate

Une action de modification immédiate cesse d'être l'action courante dès qu'elle a été appliquée ou qu'une nouvelle action a été sélectionnée. Certaines de ces actions modifient la structure du programme en cours d'édition à l'aide d'informations complémentaires, ce sont les actions : **load**, **compile**, **confirm**. Les autres sont **reset**, **undo** et **quit**.

3.1.3 Les actions de visualisation

Elles ne modifient ni la structure ni la géométrie du programme. Elles laissent invariante l'action courante ; ce sont : **save**, **refresh**, MSM (descendre dans la structure du programme) et MSR (remonter dans la structure du programme).

3.1.4 Les paramètres de visualisation

details est un indicateur booléen qui a pour conséquence, lorsqu'il est à VRAI, de visualiser la totalité du texte associé à une boîte (voir le paragraphe 4.5.2 sur la visibilité des objets).

names : lorsqu'il est positionné, cet indicateur permet de visualiser les noms des ports dans la fenêtre D, les noms étant affichés à côté des ports.

police : lorsqu'on désigne cet indicateur, un menu contenant différentes polices apparaît. Il permet de choisir la police d'affichage du texte SIGNAL dans les boîtes contenues dans la fenêtre D.

3.2 L'interface système

A l'initialisation du système, les fenêtres présentées Figure 4 sont affichées à l'écran. Il est alors possible de créer ou de charger un modèle de processus.

Lorsque, en l'absence d'objet courant :

- l'action **create** est invoquée, les quatre constituants géométriques d'un modèle de processus apparaissent dans la fenêtre D et celui-ci devient objet courant (cf 4.5),
- l'action **load** est invoquée, un menu déroulant apparaît, permettant à l'utilisateur de choisir entre des fichiers de type graphique (fichiers précédemment crées sous l'éditeur graphique) ou des fichiers de type texte (fichiers contenant du source Signal); ensuite un "browser" (ou fenêtre de sélection de fichiers) est affiché à l'écran (Figure 6) et l'utilisateur peut alors choisir un fichier, modifier le filtre ou annuler l'action. (cf 5.1),
- une action différente des deux précédentes est invoquée, elle reste sans effet.

L'action **save** peut être invoquée à tout moment. Un menu déroulant permet de choisir entre :

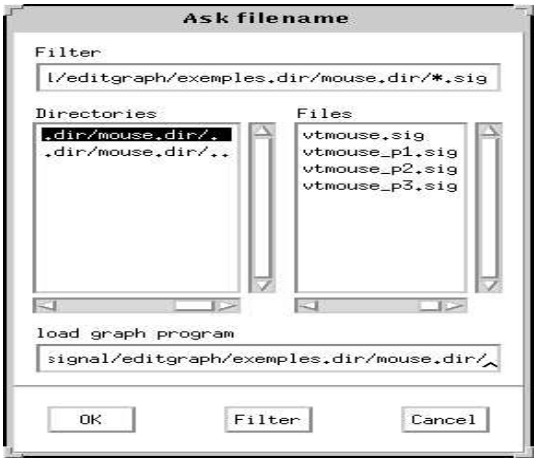

Figure 6 : le browser

- une sauvegarde complète du programme sous forme graphique,
- une sauvegarde du modèle courant (c'est-à-dire le modèle de processus le plus interne qui contient l'objet courant) sous forme graphique,
- une sauvegarde complète sous forme textuelle (source Signal),
- une sauvegarde du modèle courant sous forme textuelle.

Comme pour le chargement, un "browser" apparaît à l'écran et l'utilisateur peut choisir le nom d'un fichier, ou annuler l'action.

L'invocation de l'action **compile** provoque l'apparition d'un menu déroulant qui permet, soit de compiler le programme tout entier, soit de compiler le modèle de processus courant, soit de choisir les options de compilation. Avant d'appeler le compilateur, il est nécessaire de sauvegarder le programme sous une forme texte utilisable par le compilateur. Un "browser" permet de choisir le nom du fichier de sauvegarde ; une seconde fenêtre est utilisée, s'il y a lieu, pour fournir les paramètres de compilation. La traduction d'un programme graphique en un programme textuel est décrite en 6.2. Les résultats de la compilation sont fournis dans une autre fenêtre (Figure 7).

Le choix des options se fait par l'intermédiaire d'une fenêtre composée de boutons "tout ou rien" correspondant à l'ensemble des options possibles (Figure 8). Les options sélectionnées sont celles dont le bouton associé est noir.

Pour connaître la signification des différentes options du compilateur, consulter l'annexe A du manuel SIGNAL ([4]).

La désignation du bouton **reset** détruit l'arbre complet et effectue une réinitialisation générale. Une confirmation est demandée.

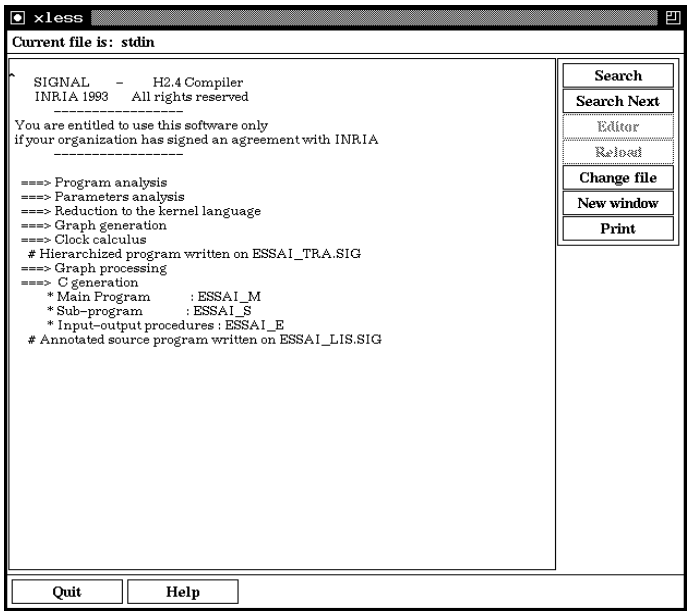

Figure 7 : les résultats de compilation

Il est mis fin à la session d'édition en désignant le bouton **quit** ; un "browser" apparaît, permettant d'effectuer une sauvegarde, si nécessaire.

3.3 La fenêtre texte

La fenêtre texte contient le texte associé à l'objet courant si cet objet est l'un des objets ci-dessous. Le texte est précédé d'un indicateur précisant la catégorie syntaxique à laquelle il appartient. Cet indicateur peut être :

- [NOM] s'il s'agit d'un identificateur
- [OL_DEC_SIG] s'il s'agit d'une liste optionnelle de déclarations de signaux (entrées, sorties, variables locales, paramètres).
- [E_PROC] s'il s'agit d'une expression de processus.
- [DEC_SIG] s'il s'agit de la déclaration d'un signal associé à un port.
- [INTE] s'il s'agit de l'interface d'un modèle ou d'une expression de processus.

Si l'objet courant n'est pas l'un des objets ci-dessus, la fenêtre n'est pas présente à l'écran.

C'est par l'intermédiaire de cette fenêtre que l'utilisateur peut modifier le texte associé à l'objet courant (cf 5.4).

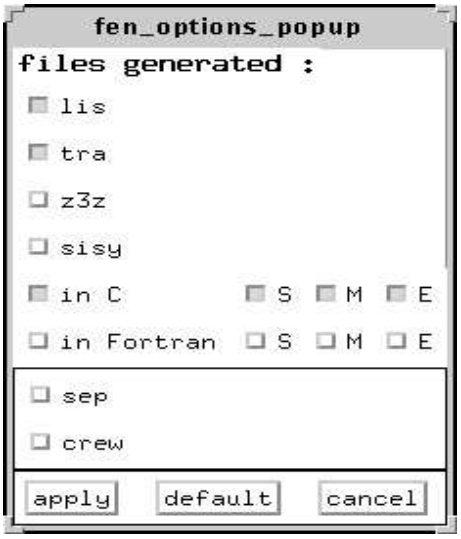

Figure 8 : les options de compilation

3.4 Les informations dans l'environnement

Le bouton **help** est un indicateur booléen qui, lorsqu'il est à VRAI, inhibe l'action courante et en décrit le mode de fonctionnement dans une fenêtre de messages. Lorsqu'une action est désignée alors que cet indicateur est vrai, le seul effet de cette désignation est donc de produire dans une fenêtre les informations nécessaires à utilisation de cette action. La figure 9 montre la fenêtre affichée lorsque l'utilisateur a désigné le bouton **delete** (avec **help** à vrai).

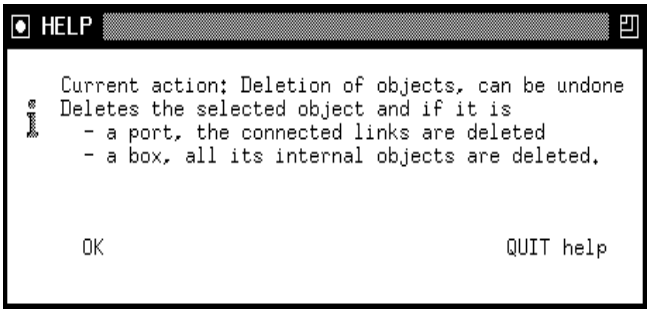

Figure 9 : une fenêtre d'information

4 Les objets graphiques

Graphiquement, les objets manipulés par l'éditeur sont de trois types : les boîtes, les ports et les liens.

4.1 Les boîtes

Les boîtes sont utilisées pour construire des modèles et des expressions de processus.

1. les modèles de processus

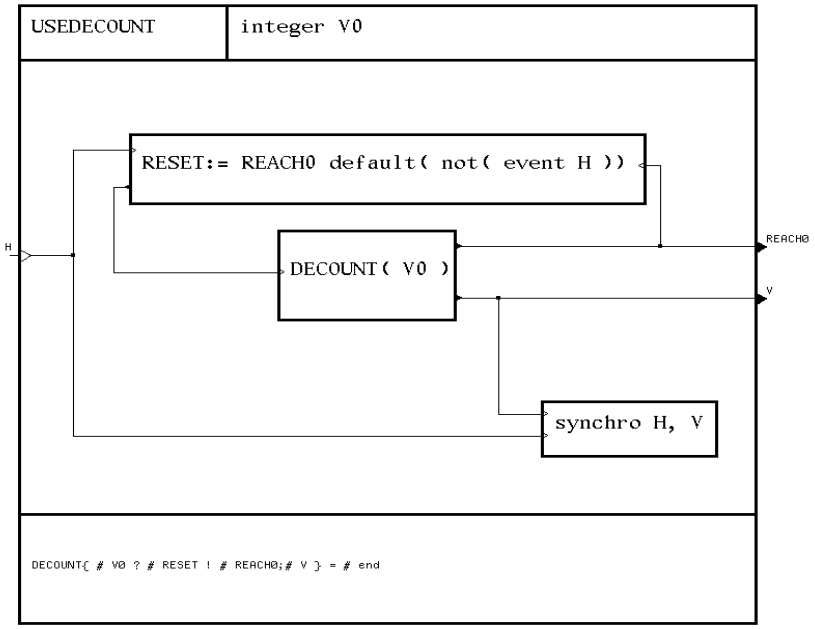

Figure 10 : un modèle de processus

En prenant comme exemple celui de la Figure 10, un modèle de processus établit une désignation entre un nom (zone N : **USEDECOUNT** dans l'exemple) et une expression de définition de signaux (zone C : corps du modèle) éventuellement paramétrée : les paramètres sont alors dans la zone P (**integer V0**) ; un modèle peut avoir des sous-modèles dont les déclarations se trouvent dans la zone DP (dans l'exemple, il y a un sous-modèle nommé **DECOUNT**). Un modèle de processus est ainsi représenté par quatre boîtes connexes et graphiquement invariantes.

2. les expressions de processus

   Une expression de processus est la composition de sous-expressions qui peuvent être :

- récursivement des expressions graphiques (en grisé dans la figure 11)
- du texte signal (en grisé dans la figure 12)

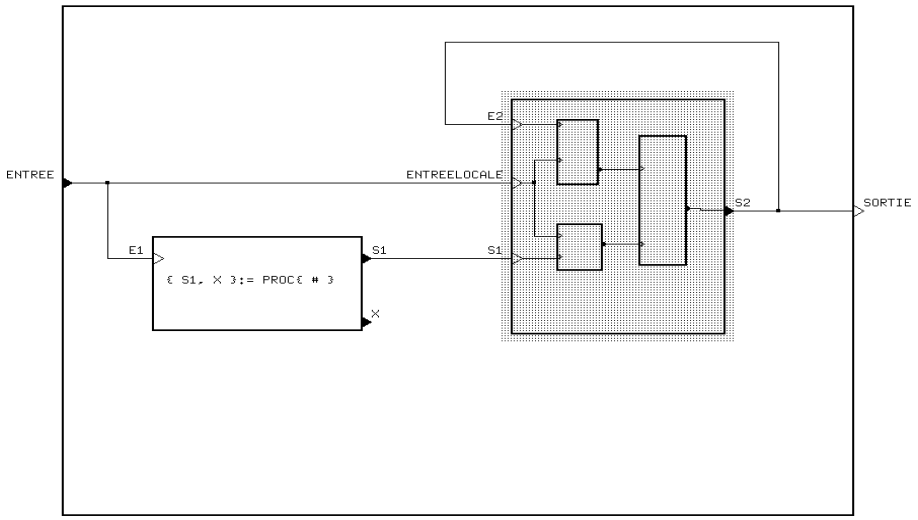

Figure 11 : boîte grisée hiérarchisée

4.2 Les ports

Un port est un point de connexion à une boîte soit en entrée, (ENTREE ou E1 dans la figure 13), soit en sortie (SORTIE ou S1 dans la figure 13). Il est représenté par un triangle à l'intérieur de la boîte pour les ports d'entrée, à l'extérieur de la boîte pour les ports de sortie. Un port représente un signal dont les valeurs sont consommées si elles sont en entrée, produites si elles sont en sortie. Un port peut être local à une boîte (comme E1 et S1 dans la figure 13) ou être un port d'interface (comme ENTREE et SORTIE dans la figure 13). Cette notion de localité dépend évidemment du contexte d'observation. Ainsi, ENTREELOCALE est un port local pour la boîte englobante de la figure 13 et un port d'interface pour la sous-boîte grisée de la figure 11.

La surface délimitée par le triangle de représentation d'un port est, au niveau considéré,

- noire lorsque le port est en position de diffusion, c'est-à-dire lorsque le port transmet le signal vers d'autres ports.
- blanche lorsque le port est en position de réception, c'est-à-dire lorsque le port reçoit le signal d'un autre port.

Suivant le contexte d'observation, un port peut être, soit diffuseur, soit récepteur.

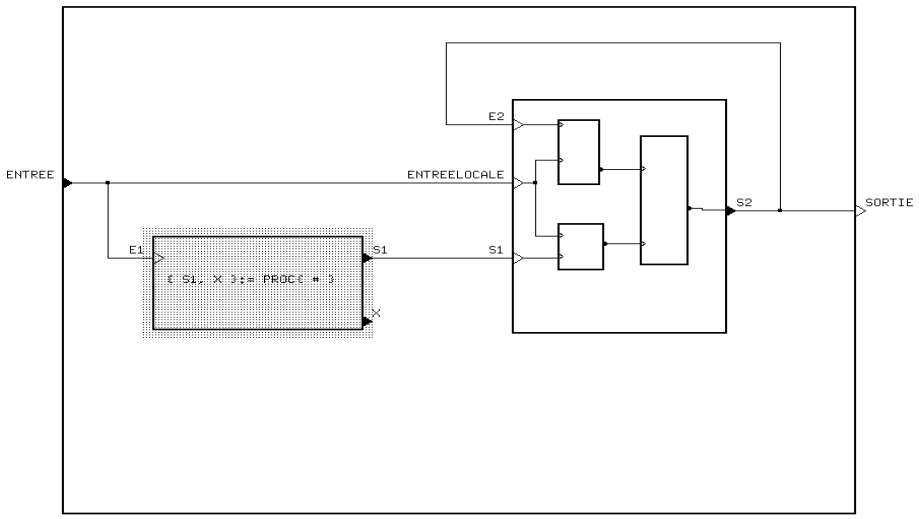

Figure 12 : boîte grisée terminale

4.3 Les liens

Un lien établit une connexion entre deux ports. Cette connexion définit les séquences de valeurs transmises par des ports interconnectés comme étant identiques pour chacun d'eux. Elle correspond à l'identité des noms de variables pour la composition textuelle. Un lien connecte un port, soit avec un port de même niveau, soit avec un port appartenant à une de ses boîtes internes. Conformément à la sémantique de SIGNAL, un port ne peut recevoir qu'un seul signal mais peut diffuser ce signal vers autant de ports que souhaité (dans la figure 13, S2 diffuse son signal vers E2 et SORTIE).

Un lien est formé d'un arbre de chemins (parties éventuellement communes à plusieurs liens qui partent d'un même port diffuseur), chaque chemin étant lui-même constitué d'une suite de segments verticaux ou horizontaux. Dans l'exemple ci-dessus (Figure 13), le port d'entrée externe ENTREE est connecté au port ENTREELOCALE par un lien formé des chemins A et B, et au port E1 par un lien formé des chemins A et C ; le chemin A est commun à ces deux liens, le chemin C est formé de deux segments, tandis que les chemins A et B ne sont composés que d'un seul segment chacun.

Afin d'éviter une confusion possible entre intersection de segments et composition de chemins, la confluence de deux chemins est marquée d'un petit carré noir.

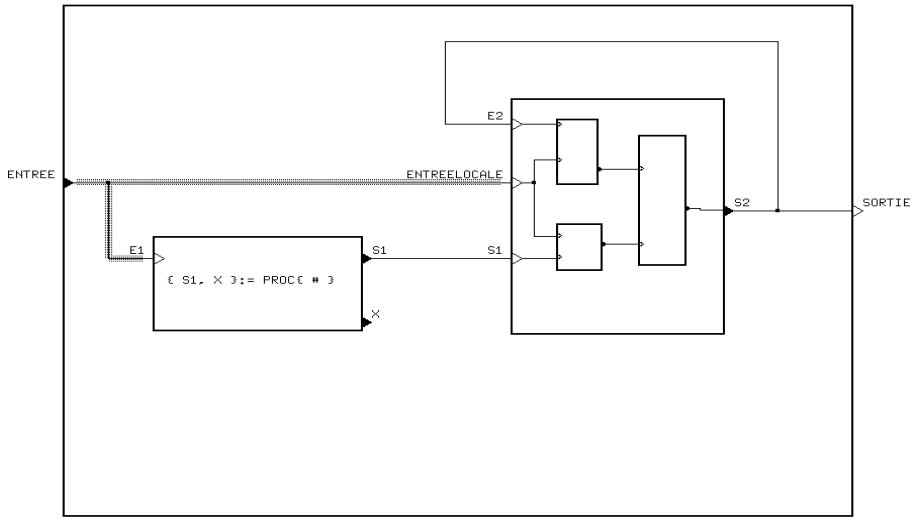

Figure 13 : les connexions entre boîtes

4.4 Les halos

Chaque objet accessible (cf 4.5) est muni d'une zone d'ombrage qui peut être rendue visible grâce au bouton **halo**. Cette zone d'ombrage est la zone de désignation de l'objet. Deux zones d'ombrage ne peuvent avoir une intersection non vide que si l'une est incluse dans l'autre (cas de la zone de désignation d'un port qui est incluse dans celle de la boîte à laquelle il appartient) ou si ce sont les zones d'ombrage de segments perpendiculaires (voir Figures 11, 12, 13). Deux segments de même direction ne peuvent pas se chevaucher.

4.5 Description de la structure des objets

Un objet possède une structure arborescente. C'est sur cette structure que repose l'édition interactive d'un objet. L'édition d'un objet se fait dans la fenêtre D lorsqu'il s'agit de dessin et dans la fenêtre T lorsqu'il s'agit de texte.

La fenêtre T contient, sauf indication contraire, la représentation textuelle d'un objet appelé **objet courant** ; la fenêtre D contient la représentation graphique du plus interne des objets à structure graphique qui contient l'objet courant.

Exemple : lorsque l'objet courant est un port, la fenêtre texte contient la déclaration de signal associé à ce port tandis que la fenêtre D contient soit l'expression de processus, soit le modèle auquel appartient le port.

L'ensemble des composants de l'objet affiché dans la fenêtre D constitue récursivement l'ensemble des objets visibles à l'écran. Lorsque objet affiché et objet courant coïncident, l'ensemble des composantes directes de cet objet (cf 4.5.1) forme l'ensemble des objets accessibles.

4.5.1 Structure graphique des objets

1. Parcours

   Le parcours de la structure est obtenu par utilisation de la souris :

   - le bouton de droite (MSR) permet de faire de l'objet contenant directement l'**objet courant**, le nouvel **objet courant** (on remonte dans l'arbre).
   - le bouton central (MSM) permet de faire de l'objet accessible contenant le point visé, le nouvel **objet courant** (on descend dans l'arbre).

2. Objets structurés graphiquement

   Les objets suivants possèdent une structure graphique :

   (a) un modèle contient les composantes directes suivantes (figure 10) :

   - Un nom (USEDECOUNT dans l'exemple de la figure 10, zone N)
   - Une liste de paramètres (integer V0 dans l'exemple, la zone P)
   - Une expression de processus le définissant (la grosse boîte centrale, zone C)
   - Les ports d'entrée et de sortie de cette expression
   - Une liste de déclarations de sous-modèles locaux (la boîte du bas, zone DP)
   - Une liste de déclarations locales. Ces déclarations sont affichées dans la fenêtre T en désignant la zone DP avec le bouton MSM, la touche CTRL du clavier étant enfoncée.
   - Les entrées et sorties externes de l'expression associée au modèle. Elles sont affichées dans la fenêtre D, au niveau des ports. Elles peuvent être atteintes globalement par MSM en visant la zone C, la touche CTRL du clavier enfoncée. Elles sont alors affichées dans la fenêtre T.

   (b) une liste de déclarations de sous-modèles locaux : elle a pour composantes directes chacune des déclarations représentée par une boîte contenant le texte du sous-modèle.

   (c) une expression de processus à structure graphique : elle a pour composantes directes

   - ses sous-expressions directes sous la forme de rectangles
   - chacun des segments qui composent les liens entre sous-expressions directes.
   - chacun des ports d'interface de ses sous-expressions directes.

   Les ports externes d'une expression de processus qui sont en fait des composantes directes de l'objet qui contient cette expression sont néanmoins désignables au niveau de celle-ci.

   Comme pour un modèle, il est possible d'accéder globalement aux entrées et sorties de chacune des sous-expressions directes d'une expression de processus en désignant un point de l'expression avec MSM, la touche CTRL étant enfoncée. Elles s'affichent alors dans la fenêtre T et dans le fenêtre D au niveau des ports.

3. Objets sans structure graphique

   Les autres objets référencés ci-dessous ne possèdent pas de structure graphique mais sont néanmoins accessibles ; lorsqu'un tel objet est désigné, il devient le seul objet accessible, mais l'ensemble des objets visibles est inchangé. Il s'agit de :

   - l'identificateur d'un modèle
   - la liste des déclarations de paramètres d'un modèle
   - l'interface d'une expression de processus
   - les variables locales d'un modèle
   - un port : s'il devient l'objet courant (en le désignant avec MSM), sa représentation textuelle formée de son nom, de son type et d'une éventuelle initialisation est affichée dans la fenêtre T, et son nom est affiché près de sa représentation graphique (dans la fenêtre D).

Remarque : La désignation d'un objet avec MSM, la touche CTRL du clavier enfoncée, modifie l'objet courant mais ne modifie pas l'ensemble des objets affichés. Il en est de même pour un objet sans structure graphique, lorsqu'on le désigne avec MSM.

4.5.2 Visibilité des objets

La règle de visibilité est inspirée des holophrastes à la CENTAUR : si un objet est visible jusqu'à une profondeur N (N-visible), alors ses composantes directes sont visibles jusqu'à une profondeur N-1. Un objet, visible à une profondeur 0, n'est pas affiché. Cette règle subit des exceptions visant à améliorer la lisibilité de l'écran. Un modèle est 2-visible. Lorsque le bouton **details** possède la valeur vrai, un texte contenu dans un élément visible dans la fenêtre D est entièrement visible (sauf s'il ne tient pas dans sa boîte) sinon il est visible selon une profondeur héritée. Si le texte complet ne tient pas dans son cadre, il est alors décompilé avec le plus grand holophraste possible de telle sorte que le texte ne déborde pas de la boîte. A tout instant, l'objet affiché peut être réaffiché grâce au bouton **refresh**.

5 Les fonctions d'édition

Ce paragraphe décrit la façon de construire et de modifier les objets graphiques.

5.1 Construction géométrique des objets

5.1.1 Le programme

Le programme peut être obtenu par **create** ou **load**. Pour associer un nom au programme, il faut désigner la zone N avec MSM ; un texte apparaît dans la zone T ; il peut être modifié. L'action **confirm** associe à l'arbre courant le contenu de T (s'il est correct syntaxiquement). De la même manière, pour associer des paramètres formels au programme, il faut désigner

la zone P avec MSM ; un texte apparaît dans la fenêtre T, ce texte peut être modifié et l'action **confirm** associe à l' arbre courant le contenu de T.

Les ports du programme sont ceux de sa zone C. L'action de valider (par le bouton **confirm**) l'ensemble des ports d'entrée et de sortie de l'expression contenue dans C construit la liste des entrées et la liste des sorties de l'interface du modèle (pour descendre dans les ports, cf 5.4.2).

Le corps du programme est l'ensemble des définitions de signaux décrit par la boîte C (cf 5.1.3).

Les signaux locaux utilisés dans ce corps peuvent être déclarés en désignant un point dans DP (en appuyant sur le bouton MSM de la souris avec la touche CTRL du clavier enfoncée). La liste des déclarations apparaît dans la fenêtre T. On peut la modifier (dans la fenêtre T) et la valider globalement par le bouton **confirm**.

5.1.2 Les sous-modèles d'un modèle (ou du programme)

Pour accéder à un sous-modèle, il faut descendre dans la zone DP. Un sous-modèle est représenté par un rectangle de taille variable qui contient sa définition sous une forme textuelle, partiellement masquée si le rectangle est trop petit. Les "..." (Figure 14) indiquent que l'affichage du texte n'est pas complet.

```
CONTROLEUR{ ~
          ? logical LAP, RUN, H
          ! logical VISIBLE, DLAP, RAZ, DRUN, H
          }
=     (| AFFICHE:= ZVISIBLE and( not ZACTIF )
       | (| ETAT_LOGIQUE
          | HS:= ( event H )when C_ETAT
...
```

```
MOTEUR{ integer M
        ? logical RAZ, H
        ! integer MN, S, B }
=     (| CPT_MOD( 60 )
       | CPT_MOD( 60 )
       | CPT_MOD( M )
...
```

Figure 14 : les sous modèles de processus

Un sous-modèle est obtenu par **create**, **load** ou **copy**. En création, l'utilisateur désigne un point dans la zone DP et "tire" la souris en maintenant le bouton MSL appuyé jusqu'à la taille désirée.

En insertion (chargement d'un programme déjà existant), il faut donner le nom d'un fichier qui contient, soit du texte SIGNAL, soit un modèle sauvegardé sous une forme géométrique, puis désigner un point dans la zone DP et "tirer" la souris comme en création. Si le fichier contient du texte SIGNAL, celui-ci est analysé ; s'il est syntaxiquement correct, l'arbre résultant est décoré d'attributs graphiques qui permettront de manipuler le sous-modèle sous sa forme graphique. Les 4 boîtes N, P, C et DP sont créées ainsi que les ports externes correspondant à l'interface, et ce, pour le modèle et récursivement pour ses sous-modèles.

Le sous-modèle est créé, inséré ou copié (cf 5.3.1) uniquement si l'intersection de son halo avec les halos des sous-modèles déjà existants est vide. Il est possible d'en changer la taille, de le déplacer, de le détruire.

Le fait de descendre dans un sous-modèle permet d'en obtenir sa représentation graphique (les quatre boîtes N, P, C et DP).

5.1.3 Les expressions de processus

Une expression de processus est représentée par un rectangle de taille variable contenant, soit du texte, soit du graphique. Ce rectangle (ou cette boîte) a des ports de sortie et/ou des ports d'entrée et ces ports sont connectés par l'intermédiaire de liens.

Pour créer une expression de processus, il faut descendre, soit dans la zone C du programme ou d'un sous-modèle, soit dans une boîte expression de processus. On procède alors comme pour un sous-modèle, c'est-à-dire qu'on désigne un point avec MSL et qu'on "tire" en maintenant le bouton appuyé jusqu'à la taille désirée. La boîte est créée si l'intersection de son halo avec les halos des boîtes et des liens déjà existants est vide.

Pour créer un port, il faut désigner un point près du bord gauche ou droit de la boîte, à l'intérieur de la boîte pour un port d'entrée, à l'extérieur (mais toujours dans l'ombrage) pour un port de sortie. Le nombre de ports que possède une boîte est limité par la taille de la boîte, une distance minimale est conservée entre deux ports ou entre un port et un coin de la boîte. Il n'y a pas de ports sur les bords haut et bas d'une boîte.

5.1.4 Les liens

Pour créer un lien, on peut procéder de deux façons : par routage automatique ou par routage manuel.

En routage automatique, on désigne avec le bouton MSL, soit les deux ports à connecter, soit un port et un bord d'une boîte (et il y a création automatique du port). L'algorithme de routage recherche un chemin entre ces deux points. Il y a échec (et donc, le lien n'est pas créé) si le routage ne trouve pas de chemin entre les deux ports.

En routage manuel, on désigne d'abord un premier port, puis les différents changements de direction et enfin soit le second port, soit le bord d'une boîte (et il y a création automatique du port).

Lorsqu'on connecte ensemble deux boîtes de même niveau, l'un des ports est un port de sortie et l'autre un port d'entrée. Par contre, lorsqu'on crée un lien entre une boîte interne et sa boîte englobante, les deux ports sont, soit tous les deux des ports d'entrée, soit tous les deux des ports de sortie. Visuellement, on connecte toujours un port représenté par un triangle blanc avec un port représenté par un triangle noir.

Restrictions conformes à la définition de SIGNAL :

- Il est interdit de connecter deux ports externes entre eux.
- Il est interdit de diffuser le même signal sur deux ports de sortie externes.

5.2 Modification de la structure géométrique

Lorsqu'une modification de géométrie intervient, elle peut entraîner une nouvelle recherche de chemins entre ports connectés ; la modification souhaitée échoue lorsque les règles d'intersection de halos ne sont pas respectées ou lorsque le routage échoue. Dans ce cas, la modification est sans effet.

5.2.1 Déplacement

Pour déplacer un objet, il faut désigner l'objet avec MSL et "tirer" en maintenant le bouton appuyé jusqu'à l'endroit désiré. Le déplacement d'un port connecté ou d'une boîte dont un des ports est connecté entraîne le déplacement des liens qui représentent ces connexions.

- les liens

  Le déplacement d'un lien est obtenu par déplacements successifs des segments qui le composent. On ne peut déplacer un segment vertical (respectivement horizontal) que horizontalement (respectivement verticalement). Il est possible d'aligner un segment sur un autre segment de même direction et appartenant au même lien (Exemple : aligner le segment 3 sur le segment 1, figure 15) ; pour cela, il faut amener le segment à aligner dans le halo de l'autre segment, les deux segments sont automatiquement joints.

- les ports

  Un port ne peut être déplacé que sur le bord de la boîte où il a été créé. Si le port est connecté, les liens qui arrivent ou partent du port déplacé (liens internes et externes) sont redessinés.

- les boîtes

  Si une boîte déplacée est connectée, les liens qui arrivent ou partent de la boîte sont redessinés.

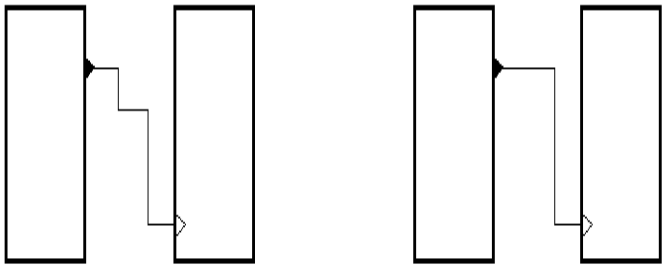

Figure 15 : alignement de liens

5.2.2 Changement de taille

Il est possible de changer la taille d'une boîte, soit dans les deux sens (hauteur et largeur) en désignant un coin de la boîte, soit dans un seul sens en désignant un bord de la boîte, et en "tirant" en maintenant le bouton appuyé jusqu'à la taille désirée.

Dans le premier cas (changement dans les deux sens), les positions des ports sont recalculées proportionnellement au changement de taille ; les liens externes de la boîte sont redessinés par routage automatique.

Dans le second cas :

- si on déplace le côté gauche ou droit de la boîte, seuls les liens externes qui arrivent sur ce côté sont redessinés par routage automatique.
- si on déplace le côté haut ou bas de la boîte, les positions des ports sont recalculées proportionnellement au changement de taille et les liens externes sont redessinés automatiquement.

5.2.3 Justification

Cette fonction ne concerne que les ports. Elle permet de répartir un ensemble de ports appartenant à une même boîte de façon équidistante entre 2 points donnés. Pour justifier, il faut donc désigner deux points appartenant à un bord vertical d'une boîte, les ports entre ces deux points sont alors équitablement répartis. Si les ports sont connectés, les liens concernés sont automatiquement redessinés. (figure 16).

5.3 Actions indépendantes de la forme de l'objet

5.3.1 Copie

Il est possible de dupliquer un sous-modèle ou une expression de processus. Il faut d'abord désigner l'objet à copier (avec MSL) puis l'endroit où on veut insérer l'objet (toujours avec MSL). Il est possible de descendre ou de remonter dans l'arbre entre les deux désignations, ce qui veut dire qu'on insère l'objet dupliqué là où on veut dans l'arbre, à la seule condition

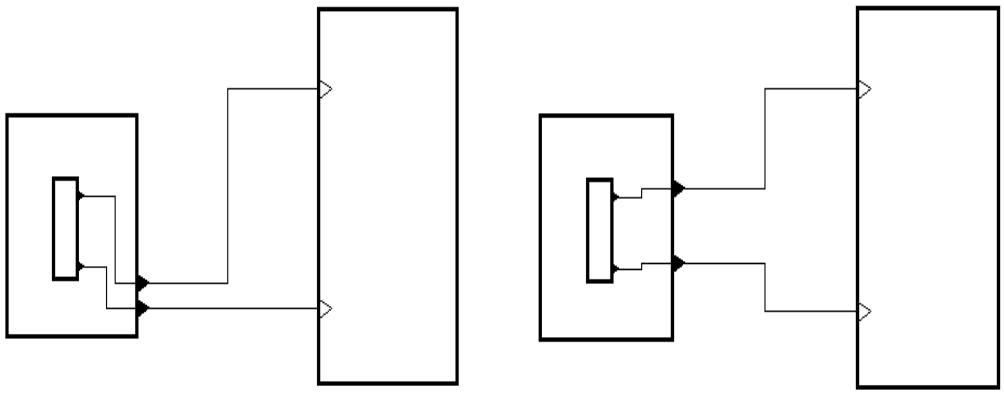

Figure 16 : justification de ports

que l'insertion soit syntaxiquement correcte. Un sous-modèle ne peut être inséré que dans la zone DP d'un modèle ; une expression de processus ne peut être insérée que dans la zone C ou à l'intérieur d'une boîte appartenant à la zone C d'un modèle.

La copie d'une boîte entraîne la recopie de ses ports et de toute la hiérarchie de ses objets internes.

Il est également possible de dupliquer un port et sa déclaration.

5.3.2 Destruction

La destruction est effectuée lorsque l'action courante est **delete**.

- Lorsqu'on désigne un segment, le lien à qui appartient ce segment est détruit ; si le segment est commun à plusieurs liens, tous les liens sont détruits.

- Lorsqu'on désigne un port, le port ainsi que les liens externes et internes qui arrivent ou qui partent de ce port sont détruits.

- Lorsqu'on désigne une boîte, la boîte, ses ports, les liens externes qui arrivent ou partent de la boîte ainsi que toute la hiérarchie de ses objets internes sont détruits.

5.3.3 Récupération

Il est possible de récupérer (avec le bouton **undo**) l'objet courant dans l'état où il était avant modification par l'une des actions suivantes : **delete**, **move**, **size** , **justify**, **confirm**. Si le bouton **undo** n'est pas désigné juste après l'action à annuler, il est sans effet.

5.4 Association d'un texte à un objet à représentation géométrique

Lorsqu'on descend dans un objet, le texte qui définit l'objet sous forme alphabétique est affiché dans la fenêtre T ; il y est précédé d'un indicateur de catégorie syntaxique (cf 3.3).

Ce texte peut être modifié au moyen de l'éditeur du système hôte associé. Si on remonte de cet objet courant, les modifications sont sans effet. Par contre, lorsqu'on décide de le valider grâce au bouton **confirm**, ce texte est associé à l'objet courant en remplacement de son contenu précédent (cette validation détruira la représentation géométrique éventuelle du contenu). Après validation réussie (analyse syntaxique correcte), l'objet courant est celui qui serait atteint par remontée ; il est inchangé en cas d'échec.

5.4.1 Expressions de processus

Pour associer du texte à une expression de processus, il faut y descendre. Le texte est alors affiché dans la fenêtre T. Par **confirm**, le texte est analysé et s'il est syntaxiquement correct, la liste des identificateurs externes de ce texte est déterminée. Il y a analyse du contexte afin de déterminer les noms des entrées et sorties des modèles appelés dans l'expression. Chaque identificateur Id est alors tour à tour affiché dans la fenêtre T et l'utilisateur peut :

1. soit désigner avec MSL
   - un point en dehors de la boîte : Id est alors un paramètre ; il est inséré dans la liste des paramètres du modèle englobant. Si Id correspond à un signal de sortie, une erreur est engendrée.
   - un port de la boîte ; si le port est un port d'entrée (respectivement de sortie) et si le nom correspond à un signal d'entrée (respectivement de sortie), alors le port est nommé Id, sinon un message d'erreur est affiché.
   - un point dans le halo de la boîte qui n'appartient pas à un port ; un nouveau port est créé et nommé Id.
2. soit désigner un point dans la fenêtre graphique avec le bouton MSM : l'identificateur suivant est soumis.
3. soit désigner un point dans la fenêtre graphique avec le bouton MSR : la procédure d'association est arrêtée.

5.4.2 Propriétés d'un port

On peut attribuer un type, un nom, une initialisation à un port :

1. individuellement : on descend dans le port, sa déclaration courante (si elle existe) apparaît dans la fenêtre T, et l'éditeur de texte en autorise la modification. La désignation de **confirm** affecte cette déclaration au port si elle est correcte syntaxiquement;
2. globalement par interface: pour cela, il faut descendre dans la liste des ports de la boîte (en désignant la boîte avec MSM, la touche CTRL du clavier enfoncée) : la liste des signaux d'entrée et la liste des signaux de sortie apparaissent dans la fenêtre T ; elles peuvent être modifiées. La désignation de **confirm** affecte chacune des déclarations à chaque port selon leurs ordres respectifs si le nombre de ports est égal au nombre de

déclarations ; dans le cas d'inégalité, l'association est effectuée déclaration par déclaration ; l'ordre des ports est le suivant : d'abord les ports d'entrée puis ceux de sortie et pour chacun de ces deux ensembles, d'abord les ports du bord gauche de la boîte, puis ceux du bord droit, et pour un bord donné, de haut en bas.

5.4.3 Déclarations locales à un modèle

Si l'utilisateur propose un texte contenant des oblitérations de variables (E/a, E?a:b, E!a:b), il doit construire la déclaration de ces variables. Pour ce faire, il faut descendre dans la zone DP du modèle englobant (la touche CTRL enfoncée), modifier le texte affiché dans la fenêtre T, puis faire **confirm** .

6 Mise en oeuvre

6.1 La structure de données

Un programme SIGNAL textuel est représenté par son arbre de syntaxe abstraite. De la même façon, lorsque l'utilisateur dessine des boîtes, des ports, des liens, un arbre SIGNAL est construit. Cet arbre est décoré d'attributs graphiques qui regroupent toutes les informations géométriques (taille et forme des boîtes, des ports, des liens, position des différents objets...). Ces attributs sont eux-même représentés par des arbres.

Ce choix de représentation interne unique a l'avantage de laisser à l'utilisateur la possibilité de rentrer son programme comme il l'entend : soit entièrement graphiquement jusqu'aux expressions de base (affectations, appels de modèle ...), soit graphiquement jusqu'à un certain niveau, textuellement ensuite. Ceci permet d'autre part, de pouvoir insérer dans un programme décrit sous une forme géométrique des modèles de processus décrits sous forme textuelle.

Cette représentation unique permet également de typer les boîtes par les sous-arbres auxquels elles sont rattachées : l'éditeur graphique est dirigé par la syntaxe du langage ; l'utilisateur ne peut pas construire n'importe quoi, l'arbre abstrait doit rester cohérent.

Exemples :

Un sous-modèle ne peut être inséré que dans la zone DP d'un modèle.

Une boîte créée dans la zone C d'un modèle ne peut contenir qu'une expression de processus.

Le texte associé à un port ne peut être qu'une déclaration de signal.

Le texte associé à la zone N d'un modèle de processus ne peut être qu'un nom.

Le texte associé à la zone P d'un modèle de processus ne peut être qu'une liste de déclarations de paramètres.

Seules les boîtes qui contiennent des expressions de processus peuvent avoir des ports.

Il faut noter cependant une différence entre l'arbre construit à partir d'un texte SIGNAL et celui construit avec l'éditeur graphique. En SIGNAL textuel, les connexions entre ports se

font par identité des noms de ports grâce à des opérateurs tels que renommage, masquage, rebouclage, validation. Pour l'arbre élaboré au fur et à mesure d'une construction graphique, les connexions sont représentées par les liens et l'arbre ne possède pas d'opérateurs tels que renommage, masquage...Le maintien de la cohérence eût été une opération coûteuse guère utile en phase d'édition.

Lorsque l'utilisateur veut **compiler** son programme graphique, celui-ci doit donc être complété car le compilateur ne travaille que sur les arbres SIGNAL et ne tient pas compte des attributs géométriques (voir le paragraphe 6.2).

Le choix de la représentation interne des programmes amène l'éditeur à effectuer de nombreuses manipulations d'arbres. Ces opérations sont réalisées par une machine spécialisée dans le traitement des arbres multi-langages ([9]).

Dans cette machine virtuelle, les arbres sont typés par le langage auquel ils appartiennent. Ils sont représentés comme une hiérarchie de nœuds étiquetés par un opérateur prédéfini du langage (l'opérateur processus du langage SIGNAL ou l'opérateur boîte du langage graphique par exemple) ou par un opérateur commun à tous les langages (entier, caractère, chaîne de caractères...). Les nœuds peuvent représenter des sous-arbres au nombre de fils fixe et prédéfini (le sous-arbre représentant les coordonnées d'un point a toujours 2 fils) ou des sous_arbres listes (listes de signaux d'une interface en SIGNAL). La machine de traitement d'arbres offre des manipulations :

. **de construction** par création et attachement de nœuds d'un même langage.

. **de navigation** vers les ascendants, les descendants et les frères d'un noeud.

. **de modification** par ajout ou destruction d'un nœud dans une liste, par changement de la valeur d'un fils dans les nœuds d'arité fixe.

. **sur les attributs**.

  Chaque nœud peut être décoré par une liste d'attributs. Les valeurs attribuées sont de type entier, caractère, chaîne de caractères, arbres... Il est possible d'associer un attribut à un nœud, de le détruire ou de modifier sa valeur.

. **de sauvegarde** d'arbres et **de restitution** en mémoire.

La structure d'arbre est un moyen de représentation puissant, mais coûteux en occupation mémoire ; aussi la machine possède un récupérateur mémoire. Les différentes manipulations réalisées par l'éditeur textuel et graphique de SIGNAL créent des arbres temporaires. L'emplacement de ceux-ci est récupéré périodiquement quand un certain taux d'occupation mémoire est dépassé.

6.2 Conversion de la forme géométrique à la forme alphabétique

L'identification des signaux entre deux sous-expressions **E1** et **E2** est obtenue, dans la composition de ces sous-expressions, par identité des noms des variables utilisées respectivement dans **E1** et **E2**. Si la variable **x** utilisée dans **E1** doit être identifiée à la variable **y** produite dans **E2** alors il est nécessaire d'associer à l'une le nom de l'autre, par exemple :

$$\mathbf{E1}\ ?\mathbf{x} : \mathbf{y} \mid \mathbf{E2}$$

ou

$$\mathbf{E1} \mid \mathbf{E2}\ !\mathbf{y} : \mathbf{x}$$

Or, en composition graphique, les ports qui représentent les variables ne sont pas nécessairement nommés, et s'ils le sont, ils ne le sont pas nécessairement de manière identique ; de plus, des ports représentant des variables distinctes peuvent posséder des noms identiques.

La création de texte à partir d'une description graphique se traduit donc, pour l'essentiel, par la création des instructions de changement de nom (**E ?x:y**), de restriction (**E /x**) et de fermeture (**E @x**) :

- par héritage à partir des noms associés par l'utilisateur aux ports d'un modèle ;
- par synthèse (pour les variables locales) à partir des noms associés aux ports d'une expression textuelle (boîtes feuilles dans l'arbre des boîtes).

Au cours de ce traitement, la synthèse des types et des initialisations des variables d'état est également effectuée et les déclarations associées engendrées.

6.2.1 Héritage

Soit **a** le nom d'un port α entrée de l'expression E associée à un modèle P. Alors le port α est diffuseur : il peut être connecté à plusieurs ports d'entrée $\alpha_1, ..., \alpha_n$ de sous expressions. Chaque port α_i est récepteur mais il peut aussi diffuser le signal reçu vers des ports des sous expressions de l'expression qui le contient. A un port intermédiaire α_i sont donc associés, d'une part l'ensemble des ports vers lequel il diffuse récursivement ses valeurs et d'autre part, le port (appelé père) dont il est récepteur. Un port d'entrée de modèle est uniquement diffuseur (racine d'arbre), un port de sortie de modèle est uniquement récepteur (feuille d'arbre).

Tous les ports associés à chaque entrée α héritent récursivement du nom **a** de ce port jusqu'à rencontre d'un port β, entrée d'une expression E, auquel est associé un nom **b**. Si **b** et **a** sont identiques, l'arbre du programme est inchangé ; si **b** est différent de **a**, alors **E ?b:a** remplace **E** dans cet arbre. L'héritage recommence à partir de β avec le nom **b**. L'algorithme se termine aux feuilles de l'arbre des boîtes.

A l'inverse, chaque port β père d'un port de sortie α nommé **a** est le port de sortie d'une expression **E**. Si β n'est pas nommé, le nom **a** lui est associé ; s'il possède déjà ce nom, il est inchangé ; dans les deux cas, l'arbre du programme est inchangé. Si β possède un nom **b** distinct de **a** alors **E!b:a** remplace **E** dans l'arbre. Le nom **a** est alors hérité par les ports d'entrée des sous-expressions auxquelles il est connecté comme pour les ports d'entrée d'une expression, et le nouveau nom de β est récursivement hérité par son père jusqu'à la racine de l'arbre des connexions (β est le nouveau germe de l'algorithme ci-dessus).

6.2.2 Synthèse

L'algorithme présenté en 6.2.1 ne permet pas d'atteindre les variables locales d'une expression (c'est-à-dire qu'il ne permet pas d'atteindre un port d'une boite interne non connecté à un port de la boite directement englobante) pour lesquelles il est de plus nécessaire de résoudre d'éventuels conflits de nom à chaque niveau. On procède donc par synthèse à partir des boîtes feuilles du modèle (on appelle boîte feuille une boîte qui ne contient que du texte).

Soit **E1** une sous expression de **E**. Si **E1** est un texte alors l'interface de **E1** possède des ports nommés. Si **E1** possède des sous-expressions boîtes alors les noms des ports de **E1** sont synthétisés.

Soit **b** le nom d'une sortie β de **E1** non connectée à l'interface du modèle qui contient **E**. Trois cas sont possibles :

1. β n'est pas diffuseur. Alors **E1 /b** est engendré et **b** n'est plus visible (Figure 17.a).

2. β est diffuseur uniquement vers des ports de **E1**. Alors **E1** est remplacé par **(E1 ?a1:b,a2:b...)@b /b** où **a1, a2** sont des noms distincts de **b** connectés à β) et **b** n'est plus visible (Figure 17.b).

3. dans les autres cas (Figure 17.c), **b** devient visible dans **E** ; si ce nom est déjà un nom visible dans **E** alors un nouveau nom **b'** (unique dans **E**) est construit à partir de **b** qui devient le nouveau nom de β après substitution de **E1 !b:b'** à **E1**. Le nom **a** (**b** ou **b'**) de β est alors hérité (au sens défini en 6.2.1) par tous les ports récepteurs (entrées) qui lui sont connectés dans **E**. Si β n'est pas connecté à un port de l'interface de **E**, **E /a** est engendré sinon **a** devient le nouveau nom de la sortie γ de **E** à laquelle β est connecté. La synthèse de **E** est terminée lorsque toutes ses sous expressions ont été ainsi traitées.

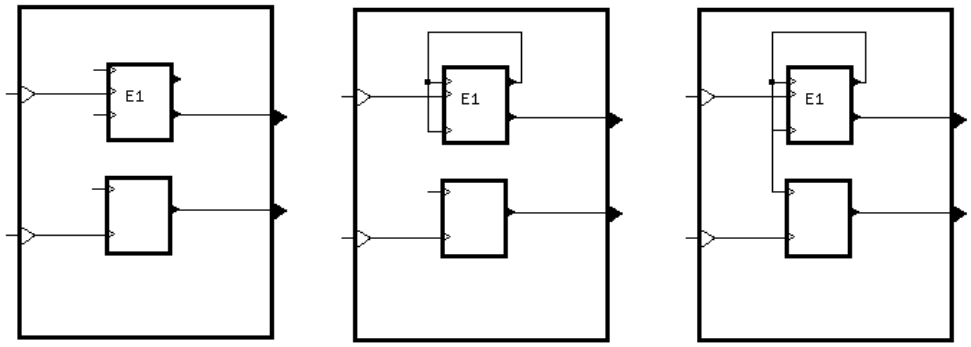

Figure 17 : synthèse des noms de ports

6.2.3 Synthèse des déclarations

Si à un port α est associé un type, ce type est alors associé à tous les ports de l'arbre de connexion qui contient α. De même, une initialisation de β (qui ne concerne que les sorties) est récursivement héritée par les ancêtres de β.

Pour chaque changement de nom **E !a:b** ou **E ?a:b** et pour chaque restriction **E /a**, une déclaration de la variable **a** munie de son type et de son initialisation éventuelle est engendrée dans la zone des variables locales du modèle qui contient E.

6.3 Routage

6.3.1 Choix d'un routage

Le déplacement d'une boîte ou d'un port et le changement de taille d'une boîte entraînent le déplacement des liens qui arrivent ou qui partent de cette boîte (les liens doivent "suivre" les boîtes, tout en évitant les obstacles). Pour cela, on utilise un algorithme de routage qui sera employé également en dessin automatique de programmes SIGNAL, pour le tracé des connexions.

Le routage consiste à trouver un chemin entre 2 points d'un plan, en évitant les obstacles et en minimisant une fonction de coût. Les coûts considérés peuvent concerner la longueur des connexions, le nombre de croisements entre liens, le nombre de changements de direction... La fonction de coût peut, plus généralement, être la composée de telles fonctions.

Pour une interface graphique, la fonction de coûts concerne des critères liés à la lisibilité du programme. Dans le cadre de l'interface en développement, il nous a semblé que la lisibilité était plutôt liée au nombre de changements de direction qu'à la longueur des chemins (voir les figures 18 et 19).

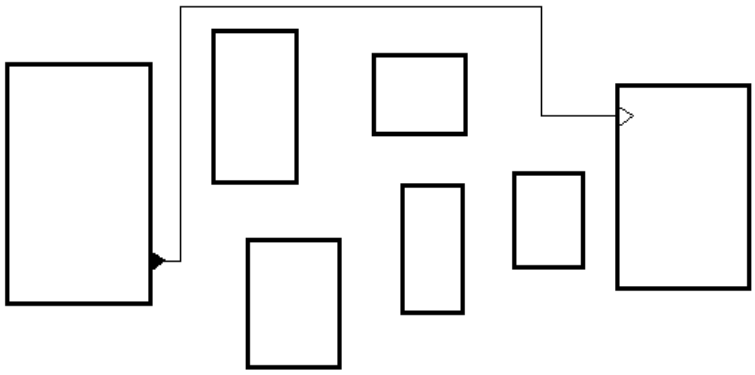

Figure 18 : un routage possible

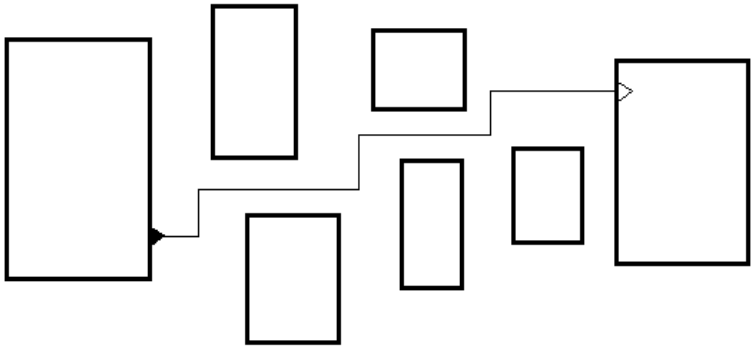

Figure 19 : un second routage possible

A coût égal, c'est-à-dire à nombre de changements de direction identique, l'algorithme choisit le chemin qui a le moins d'intersections avec les chemins déjà existants (un segment horizontal peut couper un segment vertical appartenant à un autre chemin ; par contre deux segments de même direction ne peuvent se chevaucher). A nombre d'intersections égal, c'est le chemin le plus court qui est choisi.

6.3.2 Principe de l'algorithme

On cherche à atteindre le point b à partir du point a.

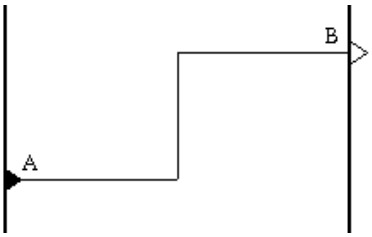

Figure 20 : Routage entre les points A et B

Le point b est affecté d'un sens d'arrivée sur ce point. Par exemple, si b appartient à un port qui se trouve sur le côté gauche d'une boîte, alors on ne peut l'atteindre que par la gauche (Figure 20).

Si a est un point appartenant à un port, on ne peut partir que dans une seule direction (comme pour le point b) ; par contre, si a est un point intermédiaire dans un chemin, alors on peut partir dans les 4 directions (gauche, droite, haut, bas) avec des coûts différents.

Le coût optimal du chemin est lié uniquement aux changements de direction obligatoires pour aller de a en b, en prenant en compte le sens d'arrivée en b et le sens de départ de a.

Initialement, on tente de joindre a et b par un des chemins de coût optimal (le chemin [(xa,ya), ((xa+xb)/2,ya), ((xa+xb)/2,yb), (xb,yb)]). Si ce chemin rencontre un obstacle, on gère une liste de tous les chemins partiels partant de a, avec leur coût optimal respectif. Un chemin partiel est constitué d'un chemin [a,x] déjà construit et d'un chemin non encore exploré [x,b].

Les différents chemins possibles sont calculés par projection des obstacles suivant l'axe de déplacement (axe des abscisses ou axe des ordonnées), ce qui est équivalent à construire un pavage irrégulier du plan en fonction des obstacles (figure 21). On essaie de prolonger le chemin courant le plus possible en direction du point à atteindre. Si on atteint ce point, on a donc trouvé un chemin entre b et a. Sinon, on cherche à prolonger le chemin partiel courant dans les deux directions perpendiculaires au dernier segment du chemin, en mettant à jour le nouveau coût optimal. En cours d'itération, on choisit en priorité le chemin le moins coûteux.

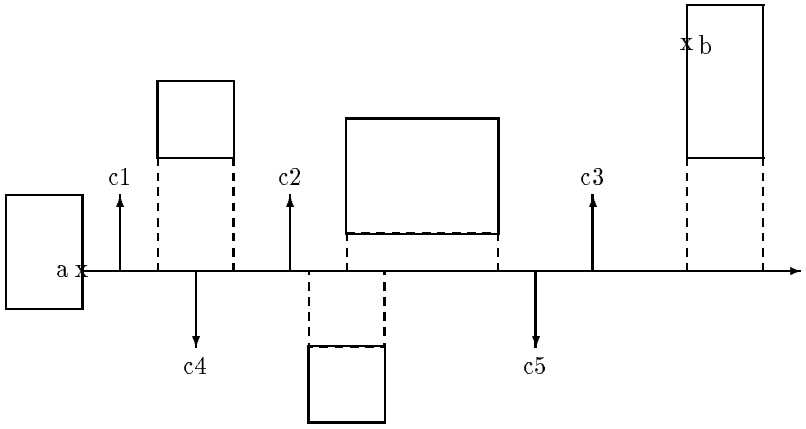

Figure 21 : les différents chemins possibles

7 Mémento des commandes possibles sous l'éditeur

Cette section est un récapitulatif de toutes les actions possibles de l'éditeur ; elles sont présentées dans l'ordre où elles apparaissent dans le bandeau de menus (Figure). Les noms indiqués sont ceux des boutons ; dans le cas de menus déroulants, on donne les noms de toute la cascade de boutons pour arriver à l'action. L'information associée à chaque action peut être retrouvée dans les sections précédentes.

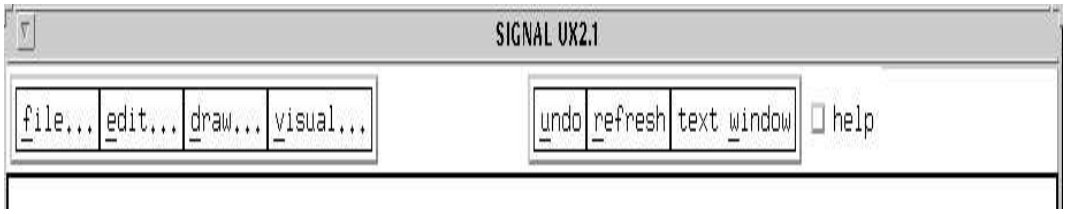

Figure 4

On donnera ici les différentes anomalies ou bugs rencontrés, ainsi que les actions non implémentées.

7.1 Rappel sur l'utilisation des boutons de la souris

1. le bouton de gauche (MSL) : il permet de désigner, soit un bouton dans le bandeau de menus, soit un point dans la fenêtre d'édition graphique, ce point pouvant appartenir à un objet graphique (boîte, lien ou port).
2. le bouton du milieu (MSM) : il permet de descendre dans une boîte ou dans un port, en désignant cette boîte ou ce port.
   Il peut être associé à la touche Control du clavier pour descendre :
   - soit dans l'interface (ensemble des ports d'entrée et de sortie) d'une boîte, en désignant cette boîte
   - soit dans la liste des signaux locaux d'un modèle de processus en désignant la zone DP du modèle.
3. le bouton du milieu (MSR) : il permet de remonter dans l'arborescence du programme, c'est-à-dire remonter d'une boîte, d'un port, d'une interface ou de la liste des déclarations locales de signaux.

▽ **Attention:** le fait de descendre dans un port, dans une interface, dans la liste des déclarations locales, dans une boîte contenant le nom ou les paramètres d'un modèle de processus ne modifient pas le contenu de la fenêtre d'édition graphique ; par contre, le contenu de la fenêtre texte est modifié, et bien sûr, l'objet courant.

7.2 Les différentes commandes

- **file**
  - **save**
    - **complete Graph**
      Sauvegarde complète de tout le programme sous forme graphique ; ce fichier pourra être relu sous l'éditeur par la commande **file-load-Graph**. Pour cette action, ainsi que les cinq suivantes, une fenêtre de sélection de fichiers (ou "browser") s'affiche, permettant à l'utilisateur de se déplacer dans ses répertoires, de filtrer ses fichiers et d'en choisir un.
    - **current graph**
      Sauvegarde du modèle de processus courant ; ce fichier pourra lui-aussi être relu sous l'éditeur par la commande **file-load-Graph**.
    - **complete Text**
      Sauvegarde complète de tout le programme sous forme textuelle : toute l'information graphique est supprimée, il ne reste que du source SIGNAL. Ce fichier peut être relu sous l'éditeur par la commande **file-load-Text** ; il peut aussi être directement compilé par le compilateur SIGNAL, en dehors de l'environnement graphique.
    - **current text**
      Sauvegarde du modèle de processus courant sous forme textuelle. Ce fichier peut être relu sous l'éditeur par la commande **file-load-Text** ; il peut aussi être directement compilé par le compilateur SIGNAL, en dehors de l'environnement graphique.
  - **load**
    - **Graph**
      Chargement d'un fichier contenant un modèle de processus sous forme graphique (ce fichier a forcemment été construit sous l'éditeur graphique de SIGNAL).
      Deux cas peuvent se produire :
      1. la fenêtre d'édition graphique est vide ; le fichier est donc chargé, en tant que programme, dès que l'utilisateur a donné son nom.
      2. la fenêtre d'édition graphique contient déjà un programme SIGNAL ; le modèle de processus contenu dans le fichier est inséré en tant que modèle local. L'utilisateur, doit donc se placer dans la zone DP des déclarations locales et créer une boîte en désignant un point avec MSL et en tirant avec la souris jusqu'à la taille désirée.
    - **Text**
      Chargement d'un fichier contenant un modèle de processus sous forme textuelle. Le texte est analysé et s'il est syntaxiquement correct, des informations graphiques sont rajoutées à l'arbre résultant afin de pouvoir le manipuler sous

sa forme graphique. Les deux cas précedemment cités (fenêtre graphique vide ou non vide) s'appliquent.

- **compile**
  - **all**
    Appel du compilateur SIGNAL. Un "browser" s'affiche afin que l'utilisateur puisse donner le nom du fichier dans lequel le programme va être stocké sous une forme textuelle (lisible par le compilateur). Une seconde fenêtre s'affiche, si besoin, pour demander les valeurs effectives des paramètres du programme. Les résultats de la compilation sont affichés à l'écran dans une fenêtre temporaire.
  - **current** : non implémenté
  - **options**
    Choix des options de compilation. Une fenêtre composée d'un ensemble de boutons apparaît à l'écran (voir Figure 8). Chaque bouton correspond à une option ; une option sélectionnée a son bouton noir ; dans le cas contraire, il est blanc. Les trois boutons du bas permettent de valider le choix effectué (bouton **apply**), de remettre les valeurs par défaut (bouton **default**), ou d'annuler les dernières modifications (bouton **cancel**).
- **reset**
  Réinitialisation complète ; une confirmation est demandée.
- **quit**
  Sortie de l'éditeur ; un "browser" s'affiche permettant de faire une sauvegarde.

• **edit**

- **create**
  Les objets que l'on peut créer sont des boîtes, des ports ou des liens. Lorsque la fenêtre graphique est vide, la désignation du bouton **create** crée automatiquement les quatre boîtes d'un modèle de processus.
  Lorsque la fenêtre n'est pas vide, l'objet créé dépend du point désigné par l'utilisateur :
  1. si le point n'appartient à aucun objet, l'objet créé sera une boîte ; pour cela, il faut tirer avec la souris, le bouton gauche appuyé jusqu'à la taille désirée. Les boîtes ont une taille minimale.
  2. si le point appartient à un des deux bords gauche ou droit d'une boîte, l'objet créé est un port (d'entrée si le point désigné est à l'intérieur de la boîte, de sortie si le point est à l'extérieur, mais néanmoins dans le halo de la boîte).
  3. si le point appartient à un port, l'objet créé sera un lien. L'utilisateur peut alors soit désigner un autre port (le lien est créé par routage automatique), soit désigner le bord d'une boîte (un port est créé et le lien est construit par routage automatique), soit désigner différents points qui correspondent aux différents changements de direction du lien et enfin un port ou le bord d'une boîte.

▽ **Attention :** pour créer un objet à l'intérieur d'une boîte, il faut d'abord descendre dans cette boîte (avec le bouton du milieu de la souris).

- **delete**
  Pour détruire un objet, il suffit de cliquer sur l'objet en question. La destruction d'un segment de lien détruit le lien ; la destruction d'un port détruit le port et les liens qui arrivent ou qui partent de ce port ; la destruction d'une boîte détruit la boîte et tous ses objets internes. En cas de destruction intempestive, il faut utiliser la commande **undo**.
- **copy**
  Pour copier un objet, il faut d'abord désigner l'objet puis désigner l'endroit où insérer la copie de l'objet. Entre les deux désignations, il est possible de monter ou descendre dans l'arborescence du programme. On peut dupliquer, soit une boîte, soit un port.
  1. dans le cas d'une boîte, tous ses ports et ses objets internes sont dupliqués. Il est possible de changer sa taille au moment où on désigne l'endroit où la copier ; il suffit de maintenir le bouton gauche appuyé et de tirer avec la souris jusqu'à la taille voulue.
  2. dans le cas d'un port, l'endroit où copier le port est forcément le bord d'une boîte. Deux cas se présentent: soit, le point désigné est déjà un port, alors la déclaration associée au premier port est recopiée et associée au second ; soit le point désigné n'est pas un port, alors il y a d'abord création d'un port (port d'entrée si le point désigné est à l'intérieur de la boîte, port de sortie sinon) et ensuite association de la déclaration.
- **glue**
  Cette action rajoute un niveau de hiérarchie intermédiaire dans la structure arborescente des boîtes (et donc du programme). Pour cela, il faut créer une boîte (voir ci-dessus) qui englobe les boîtes que l'on veut rassembler. Les liens entre les boîtes concernées et leur boîte mère sont reconstruits en rajoutant des ports intermédiaires sur la nouvelle boîte.
- **break**
  C'est l'opération duale de la précédente. Elle supprime un niveau de hiérarchie. Il suffit de désigner la boîte à supprimer et elle sera remplacée par ses boîtes internes.

• **draw**

- **justify**
  Cette action permet de répartir un ensemble de ports appartenant à une même boîte de façon équidistante entre deux points donnés. Les ports appartiennent bien sûr à un même bord de boîte. Pour cela, il suffit de désigner deux points appartenant à un bord vertical ; si les deux points sont très rapprochés, l'ensemble des ports est réparti sur toute la hauteur de la boîte. Les liens qui arrivent ou partent de ces ports sont reconstruits automatiquement.

- **size**
  - **proportional**
    Cette action ne concerne que les boîtes. Pour changer la taille d'une boîte, il faut désigner un point près d'un bord ou d'un coin de la boîte et tirer avec la souris jusqu'à l'obtention de la taille voulue. Si le point désigné appartient à un coin, le changement de taille se fait dans les deux directions (hauteur et largeur) ; si le point appartient à un bord, le changement de taille ne se fait que dans une seule direction. La position relative des ports reste inchangée.
  - **fix** : non implémenté
- **move**
  Pour déplacer un objet, il faut désigner l'objet, maintenir le bouton appuyé et tirer avec la souris jusqu'à l'emplacement voulu. Un segment horizontal (vertical) ne se déplace que horizontalement (verticalement). Un port ne peut pas se déplacer en dehors du bord de la boîte où il a été créé.
  ▽ **Attention:** actuellement, le déplacement d'un segment qui appartient à un ensemble de liens ayant le même port diffuseur n'est pas autorisé.
- **halo**
  Visualise en grisé la zone de désignation (ou halo) de l'objet cliqué par l'utilisateur. Si l'objet est une boîte ou un port et si l'indicateur **details** est positionné, le halo des liens connectés à la boîte ou au port est aussi visualisé.

• **visual**

- **grid** : non implémenté
- **police**
  1. **tiny**
  2. **small**
  3. **normal**
  4. **large**
  5. **huge**

  Choix d'une fonte pour le texte contenu dans les boîtes. Les fontes par défaut sont, de la plus petite à la plus grande, 6x9, 6x12, 8x13, 9x15bold et 10x20. Pour modifier ces fontes, voir le paragraphe suivant sur la paramétrisation de l'éditeur graphique.
- **details**
  Cet indicateur, lorsqu'il est positionné (bouton associé noir), permet de visualiser tout le texte, dans la limite du possible, contenu dans les boîtes. Dans le cas contraire (bouton associé blanc), seule une partie du texte est visible, ce qui rend l'affichage plus rapide.
- **names**
  Cet indicateur, lorsqu'il est positionné (bouton associé noir), permet de visualiser

les noms des ports dans la fenêtre d'édition graphique. Les noms sont affichés à côté des ports, en dehors des boîtes et avec la fonte **small**. Si l'indicateur n'est pas positionné (bouton associé blanc), les noms ne sont pas visibles.

- **undo**
  Permet de "défaire" la dernière action uniquement si elle fait partie des actions suivantes: **delete**, **move**, **size**, **justify**, **confirm**.

- **refresh**
  Réaffiche le contenu de la fenêtre d'édition graphique.

- **text window**
  La fenêtre texte n'étant pas toujours présente, cette action permet de la rendre visible. Ceci est nécessaire si l'utilisateur désire remplacer le contenu graphique d'une boîte par du texte.

- **help**
  Ce bouton permet de rentrer dans le mode d'aide en ligne. Une fois dans ce mode, la désignation d'une action (en cliquant sur le bouton correspondant) produit l'affichage d'une fenêtre qui contient toute l'information nécessaire à son utilisation (voir la figure 9 pour l'action de destruction). Pour sortir du mode d'aide, cliquer sur le bouton **QUIT help** ; pour changer d'action, cliquer sur le bouton **OK**. Le fait de cliquer sur le bouton **help** tout en étant dans le mode d'aide permet d'obtenir des informations générales sur l'éditeur graphique (utilisation de la souris, touches accélératrices associées aux boutons...).

- **confirm**
  Cette action permet d'associer du texte à un objet graphique ; cet objet peut être une boîte, un port, une interface (l'ensemble des ports d'une boîte) ou une liste de signaux locaux. Pour associer du texte à un objet, il faut d'abord descendre dans cet objet ; la fenêtre contient alors en première ligne la classe de l'objet courant (ne jamais enlever cette ligne) ; ensuite, il faut taper le texte et appuyer sur le bouton **confirm**. Si le texte est syntaxiquement correct, il est associé à l'objet courant.
  Dans le cas où l'utilisateur associe une expression de processus à une boîte, dès que le texte a été analysé et associé à la boîte, les identificateurs des entrées et sorties de l'expression sont affichés un à un dans la fenêtre texte et l'utilisateur peut désigner un point dans la fenêtre d'édition graphique,

  1. avec le bouton gauche de la souris :
     trois cas se présentent:
     le point est en dehors de la boîte : l'identificateur est considéré comme un paramètre et inséré dans la liste des paramètres du modèle englobant ;
     le point est un port de la boîte : le port est nommé par l'identificateur ;
     le point appartient au halo de la boîte (sans appartenir à un port), un port est créé et nommé par l'identificateur.
  2. avec le bouton du milieu : on passe à l'identificateur suivant.

3. avec le bouton droit : la procédure d'association est terminée.

7.3 Paramétrisation de l'éditeur graphique

Il est possible de choisir ses propres fontes pour le texte affiché dans la fenêtre d'édition graphique en rajoutant dans son fichier de configuration .Xdefaults les lignes suivantes :

```
Editsignal.font0: nom de la fonte
Editsignal.font1:       "
Editsignal.font2:       "
Editsignal.font3:       "
Editsignal.font4:       "
```

Exemple :

```
Editsignal.font0:-adobe-new century schoolbook-medium-r-normal--10-100-75-75-p-60-iso8859-1
Editsignal.font1:-adobe-new century schoolbook-medium-r-normal--12-120-75-75-p-70-iso8859-1
Editsignal.font2:-adobe-new century schoolbook-bold-r-normal--14-100-100-100-p-87-iso8859-1
Editsignal.font3:-adobe-new century schoolbook-bold-r-normal--20-140-100-100-p-113-iso8859-1
Editsignal.font4:-adobe-new century schoolbook-bold-r-normal--25-180-100-100-p-149-iso8859-1
```

Les fontes 0, 1, 2, 3 et 4 correspondent aux fontes **tiny**, **small**, **normal**, **large**, **huge**.

Il est possible de modifier la couleur du fond de l'éditeur en rajoutant dans son fichier .Xdefault la ligne :

```
Editsignal*background: white
```

7.4 Touches accélératrices

Les touches accélératrices permettent de choisir une action du menu uniquement avec les touches du clavier, sans utiliser la souris.

Toutes les actions rémanentes sont directement accessibles par l'utilisation de la touche CTRL suivie d'une lettre (la lettre est indiquée à côté du nom de l'action dans le menu). Il s'agit de :

- CTRL c pour **create**
- CTRL d pour **delete**
- CTRL y pour **copy**
- CTRL g pour **glue**
- CTRL b pour **break**

- CTRL j pour **justify**
- CTRL s pour **size proportional**
- CTRL f pour **size fix**
- CTRL m pour **move**
- CTRL h pour **halo**

Les autres actions sont accessibles par la touche Meta suivie d'une lettre (la lettre est la lettre soulignée dans les boutons du bandeau). Il s'agit de :

- Meta f pour **file**
- Meta v pour **visual**
- Meta u pour **undo**
- Meta r pour **refresh**
- Meta w pour **text_window**

Pour les boutons **file** et **visual**, le menu s'affiche et il suffit de taper directement la lettre soulignée qui correspond à l'action choisie. Il s'agit de :

- s pour **save**
- l pour **load**
- c pour **compile**
- r pour **reset**
- q pour **quit**

et ainsi de suite pour les sous-menus.

7.5 Lancement de l'éditeur de SIGNAL

Il suffit de taper : **xsignal**.

8 Conclusion

Les différents outils qui composent l'environnement SIGNAL (outils graphiques ou non) travaillent tous sur une représentation arborescente unique des programmes, ce qui permet de passer d'un outil à l'autre sans avoir besoin de structure de données intermédiaire. L'éditeur permet une conception hiérarchique (avec les boîtes imbriquées) et modulaire (on peut construire un modèle, le sauvegarder et l'utiliser dans différents programmes) des algorithmes. Un programme SIGNAL est construit, soit de manière descendante : on dessine les boîtes de plus haut niveau et on les définit ensuite, soit de manière ascendante : on définit les boîtes de base et on les assemble ensuite. On peut noter que le superviseur de l'interface graphique a été entièrement décrit en SIGNAL; ce superviseur gère l'interaction entre l'utilisateur et le gestionnaire de fenêtres X-Windows ainsi que l'enchaînement des actions effectuées par l'utilisateur (pour plus de détails, voir [11]).

Mais cet environnement ne serait pas complet sans une visualisation graphique des résultats de la simulation séquentielle (principalement, affichage de courbes) ni sans une animation de cette simulation sur le schéma graphique du programme (avec, par exemple, mise en évidence des signaux qui circulent sur les liens), ce qui permettrait de détecter d'éventuelles erreurs de conception des algorithmes.

Une fois réunis tous ces outils, on pourra alors fournir à l'utilisateur un environnement graphique complet pour la conception et la réalisation de ses applications temps réel.

References

[1] Albert Benveniste and Gérard Berry. *The Synchronous Approach to Reactive and Real-Time Systems.* Publication Interne 581, IRISA, April 1991.

[2] Loïc Besnard. *Compilation de* SIGNAL*: horloges, dépendances, environnement.* PhD thesis, Université de Rennes 1, France, Septembre 1992.

[3] P. Borras, D. Clément, T. Despeyroux, J. Incerpi, G. Kahn, and V. Pascual. *CENTAUR : the system.* Research report 777, INRIA, décembre 1987.

[4] Patricia Bournai, Bruno Chéron, Bernard Houssais, and Paul Le Guernic. *Manuel* SIGNAL. Technical Report 128, INRIA, April 1991.

[5] J. Coutaz. Abstraction pour interfaces interactives. *T.S.I.*, 5(4):239–250, 1986.

[6] Bruno Dutertre and Paul Le Guernic. *Description et simulation d'un système de contrôle de passage à niveau en SIGNAL.* Publication Interne 580, IRISA, March 1991.

[7] Thierry Gautier. *Conception d'un langage flot de données pour le temps réel.* PhD thesis, Université de Rennes 1, France, 1984.

[8] Thierry Gautier, Paul Le Guernic, and Loïc Besnard. SIGNAL: a declarative language for synchronous programming of real-time systems. In Gilles Kahn, editor, *Functional*

programming languages and computer architecture, pages 257–277, Lecture Notes in Computer Science, 274, Springer-Verlag, 1987.

[9] Dalila Hattab and Viviane Kerscaven. *Conception et réalisation d'une machine de traitement d'arbres.* PhD thesis, Université de Rennes 1, France, 1989.

[10] Paul Le Guernic, Thierry Gautier, Michel Le Borgne, and Claude Le Maire. Programming real-time applications with SIGNAL. *Proceedings of the IEEE*, 79(9):1321–1336, septembre 1991.

[11] Pascal Lemonnier. *Ecriture en* SIGNAL *du superviseur de l'éditeur graphique de* SIGNAL. Technical Report, IRISA, septembre 1992. Rapport de DEA informatique, IRISA, Université de Rennes 1.

[12] V. Quint. *Une approche à l'édition structurée de documents.* PhD thesis, Université de Grenoble, 1987.

[13] W. W. Wadge and E. A. Ashcroft. LUCID, *the Dataflow Programming Language.* Academic Press, 1985.

Unité de recherche INRIA Lorraine, Technôpole de Nancy-Brabois, Campus scientifique,
615 rue de Jardin Botanique, BP 101, 54600 VILLERS LÈS NANCY
Unité de recherche INRIA Rennes, IRISA, Campus universitaire de Beaulieu, 35042 RENNES Cedex
Unité de recherche INRIA Rhône-Alpes, 46 avenue Félix Viallet, 38031 GRENOBLE Cedex 1
Unité de recherche INRIA Rocquencourt, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex
Unité de recherche INRIA Sophia-Antipolis, 2004 route des Lucioles, BP 93, 06902 SOPHIA-ANTIPOLIS Cedex

Éditeur
INRIA, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)
ISSN 0249-6399