Environnement pour la Programmation Synchrone des Systèmes Réactifs

Charles André † Hédi Boufaïed † Daniel Gaffé †
Jean-Paul Marmorat ‡

20 novembre 1995

† Laboratoire Informatique, Signaux, Systèmes (I3S)
Université de Nice-Sophia Antipolis / CNRS
41 Bd Napoléon III, F-06041 Nice cedex

‡ Centre de Mathématiques Appliquées (CMA)
École des Mines de Paris – BP 207
06560 Sophia Antipolis cedex

Résumé

Nous nous intéressons à la programmation des systèmes réactifs temps-réel. L'approche choisie est synchrone. Pour la mettre en œuvre, nous avons créé une plate-forme de développement (SPORTS : Synchronous Programming Of Real Time Systems) rassemblant divers outils.

Cet article est une introduction à l'environnement SPORTS qui permet la description des comportements dans divers formalismes, leur compilation et leur validation par simulation ou preuves formelles. La plate-forme intègre des outils (importés ou spécialement développés) en vue de leur coopération. Un soin particulier a été porté pour rendre transparents les traitements internes de traduction entre les divers formats.

Le but recherché est de faciliter la conception multiformaliste d'applications réactives temps-réel s'appuyant sur des outils et méthodes bien-fondés. L'utilisation de SPORTS devrait apporter un gain de productivité et une meilleure qualité du code produit.

Mots clef : Systèmes réactifs, langages synchrones, environnement logiciel, compilation, simulation, validation.

1 Introduction

Les *systèmes réactifs* [HP85] sont en interaction permanente avec leur environnement : les stimuli reçus provoquent leurs évolutions et en réaction, ils agissent sur l'environnement. Les langages traditionnels n'apportant pas de solution satisfaisante à la programmation réactive, des langages dits réactifs ont été introduits. Ces langages peuvent adopter un style déclaratif ou impératif. Dans le cas des langages réactifs impératifs on trouve des primitives spécialement créées pour supporter différents types d'exceptions et interrompre des activités en cours (*avortement* ou *suspension*). De nombreuses applications réactives ont également un caractère *temps-réel* : les temps de réaction sont contraints, les comportements doivent être prévisibles et sûrs. C'est pourquoi ces langages sont généralement associés à la programmation des systèmes temps-réel.

Dans un "Proceeding of the IEEE" paru en 1991 et intitulé *"Another Look at Real-Time Programming"*, l'*approche synchrone* [BB91] a été présentée comme une réponse au problème de la programmation réactive. Ce numéro introduisait deux langages synchrones déclaratifs (LUSTRE [CHPR91] et SIGNAL [LGLL91]) et un langage synchrone impératif (ESTEREL [BD91]). Ces langages font l'hypothèse de synchronisme parfait qui suppose que les réactions sont instantanées et que les communications se font par diffusion instantanée.

Il existe d'autres approches de la programmation réactive. Certains langages n'adhèrent pas à l'hypothèse de forte synchronicité ; ils se contentent d'une notion d'*instant* et de réactions atomiques (citons REACTIVE C[Bou91]). D'autres enfin ont carrément des approches asynchrones (ELECTRE [RCCE92]). On rencontre également des formalismes graphiques pour exprimer les comportements réactifs. Parmi les modèles synchrones citons le GRAFCET [AFN82] qui est largement utilisé dans les systèmes automatisés et les STATECHARTS [Har87] qui ont été introduits plus récemment pour spécifier des comportements complexes.

Contexte de l'étude

Notre équipe s'intéresse plus particulièrement à la conception de *contrôleurs*. Ce sont des systèmes réactifs, soumis à des *contraintes temps-réel*. Les stimuli sont essentiellement des événements sporadiques et les comportements "exceptionnels" sont à programmer avec un très grand soin. Pour cela, nous avons opté pour une *approche synchrone multiformalisme* : GRAFCET, ESTEREL, LUSTRE, objets synchrones [BdVN94]. Nous mettons à profit le style de programmation impératif du GRAFCET (pour le contrôle de séquences simultanées) et du langage ESTEREL (richesse de traitement des préemptions). Toutefois dans les contrôleurs, certains comportements sont de nature plus combinatoire ou fonctionnelle. Dans ce cas, nous employons le langage synchrone LUSTRE qui est déclaratif et fonctionnel. Actuellement, aucun compilateur n'intègre ces trois langages. Les *objets synchrones*, introduits par F. Boulanger [BdVN94] peuvent servir d'intermédiaires : un programme synchrone est compilé en une classe d'objets synchrones ; divers objets peuvent être instanciés et connectés par le langage MDL.

Notre objectif a été de créer une plate-forme de développement d'applications réactives dans un cadre multiformalisme synchrone. Cette plate-forme, nommée SPORTS (Synchronous Programming Of Real Time Systems), rassemble des éditeurs, des compilateurs, des générateurs de code. Les aspects sûreté étant centraux dans nos applications, nous avons également intégré des moyens de *validation* mettant en œuvre des simulateurs ou des outils de preuves.

Plan

La présentation débute par une justification du choix fait pour une approche synchrone. Les limites de ces méthodes sont également explicitées. Après avoir donné l'architecture générale de la plate-forme, nous expliquons les problèmes (sémantiques et techniques) que nous avons eu à résoudre. La partie suivante est consacrée à la présentation plus détaillée de quelques uns des outils intégrés.

2 Choix de l'approche synchrone

2.1 L'approche synchrone

Les *langages synchrones* [BB91] autorisent une approche de haut niveau, bien fondée et efficace.

– Le caractère *haut-niveau* est apporté par les primitives et les constructeurs des langages. Dans un langage comme ESTEREL, le parallélisme, la préemption et les communications sont des concepts de première classe.

– Ces approches sont *bien-fondées* car les langages s'appuient sur des sémantiques mathématiques qui leur confèrent élégance et rigueur.

– Enfin, les programmes synchrones peuvent être compilés en des formes efficaces à l'exécution (automates, systèmes d'équations).

Un des objectifs de la programmation synchrone est d'apporter une *programmation sûre*. Les fondements mathématiques des langages permettent d'utiliser des méthodes formelles de preuves. On peut utiliser des outils de vérification généraux comme MEC [ABC94] sur les systèmes de transitions ou dédiés comme BAC [Hal94] pour des systèmes d'équations booléennes. À côté des vérifications formelles, la *simulation* présente un intérêt évident pour tester rapidement les comportements attendus. La simulation devient même indispensable pour des propriétés trop difficiles à exprimer de façon formelle.

Les langages synchrones sont compilés dans des formats intermédiaires (par exemple le code oc pour la forme automate). À partir de ces formats, des traducteurs peuvent générer des programmes pour différentes cibles. Il faut alors développer des *machines d'exécution* ad-hoc, par exemple pour un microcontrôleur cible.

En fait, chaque langage synchrone a vu se développer une chaîne de production complète avec possibilités de simulation et de connexion à des outils de preuves.

2.2 Langages et formalismes supportés

Nous avons retenu deux langages : ESTEREL et LUSTRE. Plusieurs raisons motivent notre choix :

– Ces deux langages peuvent produire du code oc. Ce format commun est facilement traduit en langage d'entrée pour les outils de preuves AUTO [dV89] et MEC [ABC94] qui sont respectivement basés sur les automates et les systèmes de transitions. Le format oc est également celui qui permet de créer des objets synchrones de F. Boulanger.

– Les deux langages sont complémentaires. Les styles de programmation sont différents. Par son caractère impératif, ESTEREL est particulièrement bien adapté à l'expression des comportements réactifs complexes avec traitements d'exception pouvant entraîner des préemptions (définitives ou temporaires). Au contraire, LUSTRE est un langage déclaratif. L'entité de base du langage est le nœud qui représente une fonction. Ce langage nous apporte la possibilité d'exprimer des comportements combinatoires et séquentiels très précieux lorsqu'il s'agit d'exprimer des propriétés de sûreté.

À ces deux langages, nous avons adjoint le formalisme GRAFCET, lui aussi synchrone. Le GRAFCET constitue un standard industriel. Il est très largement utilisé dans la programmation des automates programmables industriels. On peut le considérer comme un langage métier dans le domaine des systèmes automatisés de production. L'exécution en parallèle de plusieurs séquences est très facile à exprimer visuellement dans le formalisme graphique du GRAFCET.

Enfin, dans un souci de réutilisabilité, les classes synchrones associées aux modules synchrones écrits en ESTEREL, LUSTRE ou GRAFCET peuvent être instanciées sous forme d'*objets synchrones* interconnectés. Le langage MDL [Bou93] permet ces compositions.

2.3 Limitations

La possibilité de décrire à haut niveau et de façon rigoureuse des comportements réactifs complexes n'est pas l'apanage des langages synchrones. Par contre, l'hypothèse de synchronisme parfait faite par ces langages induit des avantages et des inconvénients spécifiques. Parmi les avantages signalons la facilité de manipulations formelles résultant du haut degré d'abstraction apporté par le synchronisme. La vérification des propriétés de *correction logique* est plus aisée que pour des approches asynchrones. La sémantique des langages s'en trouve également simplifiée. Par contre, les hypothèses synchrones peuvent être à l'origine de phénomènes indésirables nouveaux : les dialogues instantanés en ESTEREL peuvent provoquer des *cycles de causalité*, pas toujours faciles à interpréter par le programmeur. Les programmes présentant ces comportements sont rejetés par le compilateur. Il s'agit donc là d'un moindre mal. Plus grave est le non respect par le système réel de l'hypothèse de synchronisme parfait. Les réactions réelles ne pouvant pas se faire en temps nul, une approximation raisonnable est que les durées de réaction restent négligeables par rapport à la plus petite constante de temps du système contrôlé. Ceci suppose donc qu'on vérifie la *correction temporelle* d'une implémentation basée sur un programme synchrone.

Pour les applications réactives du type interface homme-machine ou les systèmes temps réel “pas très rapides”, les conditions précédentes sont satisfaites. La programmation synchrone apporte alors un réel confort. Pour les applications temps réel strict (échéances inviolables) à faible constante de temps, on ne peut plus se contenter du temps logique manipulé par les programmes synchrones. Les implémentations doivent être analysées en termes de temps physique.

3 La plate-forme de développement SPORTS

3.1 Généralités

L'originalité et l'intérêt de notre approche est de *rassembler les outils* (éditeurs, compilateurs, prouveurs, simulateurs et générateurs de codes) de programmation synchrone, dans une plate-forme unique : SPORTS. Nous avons opté pour une interface graphique tournant sous X-windows. Les appels se font par menus déroulants implémentés en TCL-TK [Ous94]. Parmi les outils, certains sont le résultat de nos propres développements et ils répondent à des besoins non couverts par l'existant. Cette plate-forme est bien sûr ouverte aux intégrations futures. La figure 1 précise les principales composantes intégrées dans SPORTS ainsi que les relations entre ces composantes.

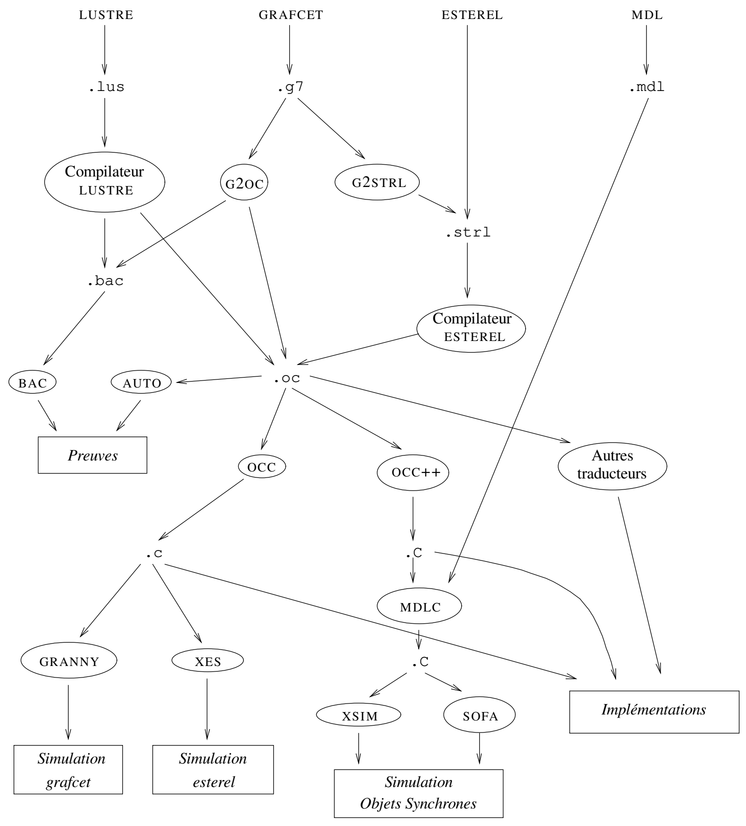


FIG. 1 - *Plate-forme de développement*

3.2 Outils supportés

– *Formalismes d'entrée et compilateurs*

  – Entrées textuelles : ESTEREL, LUSTRE, MDL.

  - Entrées graphiques : GRAFCET avec éditeur graphique et compilateurs spécialement développés (G2STRL, G2OC).
- *Outils de Preuves*
  - Accès à MEC et AUTO,
  - Génération automatique à partir des programmes sources des expressions de propriétés pour les “Model checkers” BAC et TEMPEST.
- *Mise au point et Simulations*
  - Facilités d'exécution pas-à-pas et de trace avec remontées dans le programme source (XES pour ESTEREL ; GRANNY pour le GRAFCET).
  - Simulateurs interactifs d'objets synchrones fonctionnant sous MOTIF (XSIM) ou TCL-TK (SOFA).
- *Génération de machines d'exécution.*

4 Problèmes rencontrés

4.1 Sémantique

Les sémantiques des langages ESTEREL, LUSTRE et MDL sont bien évidemment conformes aux hypothèses synchrones. Le GRAFCET qui est antérieur à ces langages, n'a pas été défini dans le même esprit. Comme nous l'avons expliqué dans un précédent article [AG94], ce modèle, bien que synchrone, présente des incohérences avec les langages synchrones. Nous avons été amenés à définir une sémantique mathématique compatible avec celle des langages [AG95]. Le modèle adopté, appelé S-GRAFCET, diffère du GRAFCET de la norme sur les notions d'instant et d'évolutions internes. Les compilateurs de GRAFCET que nous avons été amenés à développer sont basés sur la sémantique du S-GRAFCET.

4.2 Coopération entre langages

L'approche préconisée consiste à exprimer les comportements de façon modulaire. Un programme synchrone résulte de la coopération de plusieurs modules. Il était hors de question d'écrire un compilateur supportant les divers formalismes. Nous avons donc conservé les compilateurs existants. Lorsque les modules d'une application sont exprimés dans des formalismes différents nous avons adopté deux solutions :

- Soit tous les modules sont *traduits automatiquement* dans un formalisme unique (c'est le cas du compilateur G2STRL qui traduit du GRAFCET en ESTEREL),
- Soit on compile séparément les modules avec leur propre compilateur (pour le GRAFCET nous avons créé le compilateur G2OC) et on génère du code commum OC. On utilise ensuite le compilateur OCC^{++} de F. Boulanger qui associe des *classes synchrones* aux modules. La coopération entre modules est ensuite exprimée en MDL et compilée en C^{++}.

Pour faciliter l'enchaînement des diverses opérations, nous avons mis au point des *"makefiles"* génériques.

4.3 Coopération entre outils

Pour les outils de preuves généraux comme MEC et AUTO, il suffit de faire des traductions du format `oc` vers leur format d'entrée. Le compilateur ESTEREL offre une option pour générer des fichiers d'entrée pour AUTO. Le compilateur G2OC peut engendrer des systèmes d'équations booléennes dans un format accepté par BAC (Boolean Automaton Checker).

La coopération entre simulateurs est plus délicate car il s'agit de faire interagir des processus UNIX sans modifier leur code. En ayant recours à l'instruction "`send`" de TCL nous avons pu réaliser de telles interactions pour les modules ESTEREL simulés à l'aide de XES et des modules GRAFCET simulés par GRANNY.

4.4 Convivialité des interfaces

Nous avons évoqué plus haut l'existence de `makefiles`. Ces derniers sont en fait cachés à l'utilisateur. Les appels à ces constructeurs se font par un programme TCL qui présente tous les services et leurs options au travers d'une interface graphique écrite en TK. La figure 2 est une copie d'écran du panneau d'accueil.

FIG. 2 - *Panneau principal de* SPORTS

5 Présentation de quelques outils

Faute de place, nous ne décrivons que certains des outils récemment développés. En particulier les interfaces avec les outils de preuves ne seront pas traités.

5.1 Éditeur et Compilateurs GRAFCET

En GRAFCET, l'aspect graphique est très important pour modéliser puis implanter les comportements souhaités. Il est donc essentiel de fournir à l'utilisateur des outils d'édition et de génération conviviaux pour tirer un meilleur parti de ce formalisme. L'éditeur EG7 tient compte de ces préoccupations. Outre la manipulation d'icones (édition, déplacement, suppression), il intègre aussi les appels aux compilateurs G2STRL et G2OC pour générer le code du simulateur GRANNY.

Éditeur de Grafcet

EG7[Pas94] est écrit en TCL-TK. La figure 3 montre une fenêtre d'édition.

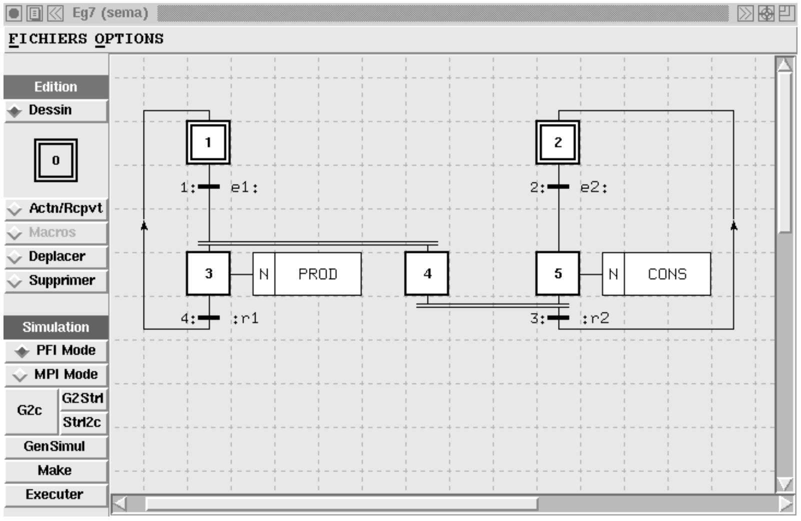


FIG. 3 - *Éditeur de* GRAFCET

– Les boutons d'`Édition` permettent de dessiner le grafcet et de le modifier. Les réceptivités ont été étendues aux événements et se présentent sous la forme "`événement : condition`".

– Les boutons de `Simulation` permettent de gérer complétement la chaîne de compilation du grafcet jusqu'à sa simulation. L'utilisateur peut en effet choisir le mode de compilation (GRAFCET classique ou S-GRAFCET) et le compilateur (G2OC ou G2STRL + ESTEREL). Il peut ensuite demander la génération du code pour GRANNY.

Remarque : L'interface EG7 simplifie le travail du concepteur qui n'a pas besoin de connaître les nombreuses options des compilateurs appelés ni de gérer lui-même les différents makefiles.

G2STRL

Ce compilateur permet de générer un fichier ESTEREL de comportement équivalent au grafcet initial. Ce code peut ensuite être lié de manière synchrone à d'autres modules ESTEREL avant d'être compilé.

G2STRL sait également détecter des instabilités potentielles sur simple étude formelle du graphe. Il s'appuie pour cela sur les BDDs.

G2OC

Le compilateur Grafcet-OC (G2OC) permet d'obtenir un automate à état fini à partir de l'étude des situations accessibles. La recherche de chaque état peut se faire suivant la sémantique d'évolution classique du GRAFCET ou celle du S-GRAFCET. G2OC peut également générer la machine d'états sous forme *d'un système d'équations booléennes* au format BAC. Ceci est très intéressant pour prouver le code et éviter l'explosion en taille de l'automate.

Au niveau preuves, G2OC détecte et rejète *tout grafcet instable*. Comme G2STRL, il utilise les BDDs. Il s'appuie également sur la sémantique du S-GRAFCET.

5.2 Génération de simulateurs

La simulation interactive est particulièrement efficace pour éprouver les comportements réactifs. Les générateurs de simulateurs que nous offrons permettent un *prototypage rapide* des applications. Différents styles sont proposés selon le(s) format(s) de description adapté(s) à l'application. La figure 4 est une copie du tableau de sélection des générateurs de simulateurs.

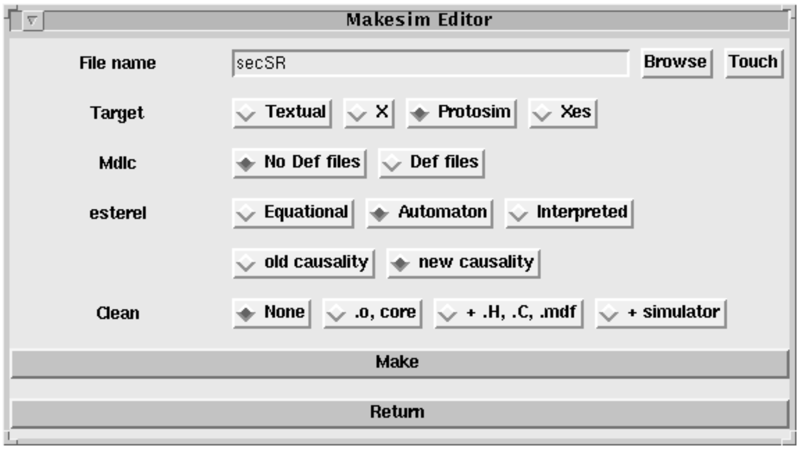


FIG. 4 - *Panneau de construction de simulateurs*

Les générateurs de simulateurs supportés sont :

– GRANNY (GRafcet ANImation) est dédié au GRAFCET,

– XES (X ESTEREL Simulator) est un des simulateurs de la nouvelle distribution ESTEREL,

– Simulation d'*objets synchrones* qui est la plus générale et admet tout format en entrée.

5.3 GRANNY : simulateur de GRAFCET

Le simulateur GRANNY (pour GRafcet ANImation) est engendré directement par EG7 à partir de la compilation d'un grafcet. L'interface basée sur TCL-TK est très conviviale.

Elle permet notamment de visualiser pas à pas les étapes et les actions actives comme le montre la figure 5.

Le panneau de contrôle se construit automatiquement en fonction du nom et du type (événementiel, conditionnel) des entrées et des sorties. Il peut fonctionner soit en mode “réflexe” dans lequel le grafcet évolue dès que l’on clique sur une entrée, soit en mode “multiple”. Dans ce dernier cas l’utilisateur peut positionner la valeur des entrées, puis valider ses choix par le bouton `transiter`.

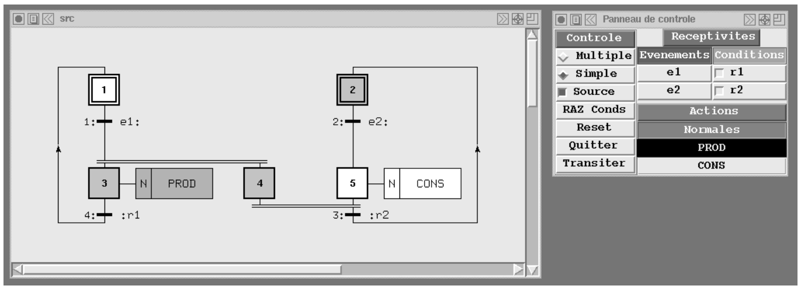


FIG. 5 - *Simulation de* GRAFCET

5.4 XES : débogeur de programmes ESTEREL

La compilation d’un programme ESTEREL correct permet de générer un code séquentiel exécutable dans divers langages cibles. On peut en particulier générer un code C spécialement instrumenté pour la simulation, qui par une édition de liens avec une librairie de collecte et d’émission de signaux, fournira un simulateur exécutable du programme. Sur ce principe, existent actuellement deux librairies d’interfaçage :

– **libcsimul.a** qui crée un simulateur ligne : les signaux d’entrée/sortie sont lus et écrits sous forme de chaînes de caractères sur les flux d’entrée/sortie standards du système.

– **libxes.a** qui crée un simulateur graphique sous **X** dans lequel les signaux d’entrée sont des boutons et les signaux de sortie des afficheurs.

Nous décrivons ici ce deuxième type de simulation.

Simulation graphique sous XES

XES est un acronyme pour X Esterel Simulator.

– **Utilisation sous** UNIX**:** Un programme ESTEREL compilé avec l’option de simulation, fournit par défaut un code C dont on peut éditer les liens avec la librairie de simulation sous X pour obtenir un simulateur exécutable. Ce simulateur comporte un panneau principal d’entrée-sortie où on retrouve tous les signaux d’entrée-sortie du programme ESTEREL, ainsi que diverses fenêtres additionnelles pour les signaux locaux, les exceptions, les variables, le code source du programme et l’aide en ligne.

– Le panneau d'entrée/sortie

  Outre une barre de menus et un bouton d'activation `RUN` ce panneau est découpé en six zones :

  – signaux d'entrée purs
  – signaux d'entrée valués
  – signaux de sortie purs
  – signaux de sortie valués
  – capteurs
  – tâches asynchrones

  Seulement trois de ces six zones existent dans l'application illustrée figure 6. Une simulation est une exécution pas à pas des transitions du programme ESTEREL. L'utilisateur prépare les événements d'entrée en sélectionnant ses signaux à la souris. Les émissions de la transition courante sont affichées.

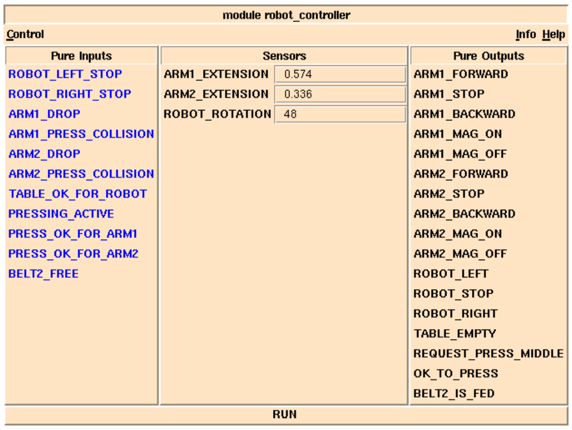


FIG. 6 - *Panneau principal de* XES

– Le menu de contrôle (Figure 7)

  Le simulateur peut fonctionner en deux modes :

  – `AutoSend` : la sélection d'un signal provoque immédiatement la transition
  – `NoAutoSend` : plusieurs signaux peuvent être présélectionnés jusqu'à une action d'activation `RUN`. Ceci permet de simuler les simultanéités d'occurrences.

  Ces modes sont contrôlés par un menu qui permet également de déclencher diverses actions comme `Reset` ou `Quit` ou de faire monter des fenêtres additionnelles (signaux locaux, variables, exceptions, arbre du programme). Une fenêtre additionnelle d'importance est la fenêtre de code source du module principal qui est affichée à l'initialisation. Les fenêtres des modules secondaires sont appelables en sélectionnant à la souris leur nom dans l'arbre du programme.

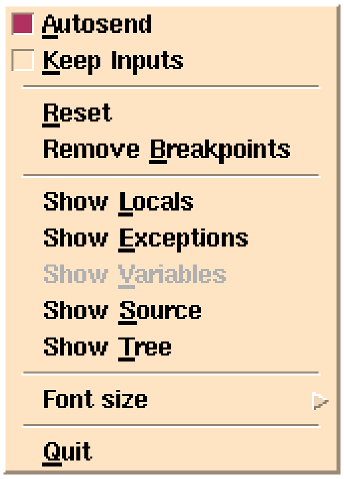


FIG. 7 - *Menu de contrôle de* XES

– L'arbre du programme (Figure 8)

  La structure modulaire du programme s'affiche dans une fenêtre auxiliaire. La sélection d'un sous-module fait apparaître sa fenêtre de code source.

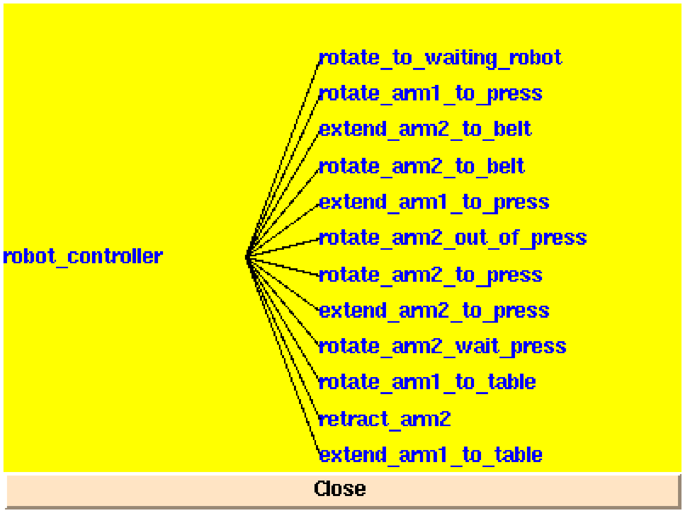


FIG. 8 - XES *: Arbre du programme*

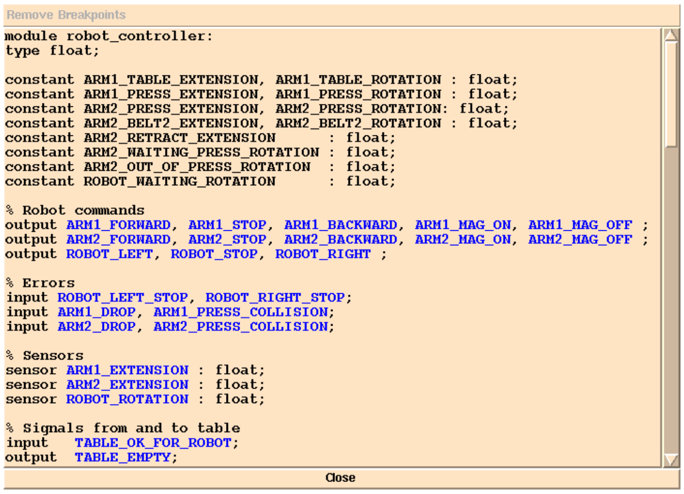
– Les fenêtres de code source (Figure 9)

  Les fenêtres de source sont mises à jour à chaque transition et comportent un certain nombre de renseignements sur l'état courant de l'exécution du programme.

  – les signaux présents dans l'instant courant sont colorés en rouge,
  – les instructions d'attente sont soulignées et peuvent recevoir un point d'arrêt,
  – les instructions d'attente qui sont actives sont rouges,

– on inspecte les valeurs courantes des signaux valués et des variables par sélection directe sur leur nom.

Ces fonctionnalités font de XES un outil très puissant de *mise au point des programmes* ESTEREL.


Remove Breakpoints

```
module robot_controller:
type float;

constant ARM1_TABLE_EXTENSION, ARM1_TABLE_ROTATION : float;
constant ARM1_PRESS_EXTENSION, ARM1_PRESS_ROTATION : float;
constant ARM2_PRESS_EXTENSION, ARM2_PRESS_ROTATION: float;
constant ARM2_BELT2_EXTENSION, ARM2_BELT2_ROTATION : float;
constant ARM2_RETRACT_EXTENSION       : float;
constant ARM2_WAITING_PRESS_ROTATION : float;
constant ARM2_OUT_OF_PRESS_ROTATION  : float;
constant ROBOT_WAITING_ROTATION      : float;

% Robot commands
output ARM1_FORWARD, ARM1_STOP, ARM1_BACKWARD, ARM1_MAG_ON, ARM1_MAG_OFF ;
output ARM2_FORWARD, ARM2_STOP, ARM2_BACKWARD, ARM2_MAG_ON, ARM2_MAG_OFF ;
output ROBOT_LEFT, ROBOT_STOP, ROBOT_RIGHT ;

% Errors
input ROBOT_LEFT_STOP, ROBOT_RIGHT_STOP;
input ARM1_DROP, ARM1_PRESS_COLLISION;
input ARM2_DROP, ARM2_PRESS_COLLISION;

% Sensors
sensor ARM1_EXTENSION : float;
sensor ARM2_EXTENSION : float;
sensor ROBOT_ROTATION : float;

% Signals from and to table
input   TABLE_OK_FOR_ROBOT;
output  TABLE_EMPTY;
```

Close

FIG. 9 - XES *: Une fenêtre de code source*

5.5 SOFA : simulateur d'objets synchrones

SOFA (Synchronous Object Friendly Animation) permet de simuler des applications conçues comme un ensemble d'objets synchrones coopérants. Le compilateur MDLC construit une machine d'exécution qui traduit ces comportements. SOFA assure l'interfaçage de cette machine avec l'opérateur. Par défaut les entrées et sorties apparaissent sous forme de boutons et afficheurs dans un panneau de contrôle (signal `top` de la figure 10). Grâce au sous-menu `connection`, l'opérateur peut associer des widgets réactifs à ces terminaux. Une bibliothèque adaptée aux automatismes est disponible. Elle peut être étendue par l'utilisateur. Il en résulte une simulation interactive plus réaliste et conviviale.

6 Conclusion

Les utilisateurs de GRAFCET sont généralement réticents à l'usage d'un langage synchrone comme ESTEREL qui fait largement appel à des constructions "informatiques" telles que les `trap ...end trap`. Inversement, le GRAFCET a été longtemps négligé par les informaticiens qui voyaient dans ce modèle un langage métier, n'offrant pas toutes les garanties sémantiques. Or, GRAFCET et langages synchrones apportent des solutions effectives dans leurs domaines respectifs. Nous sommes convaincus que la *coopération* entre ces deux approches facilite la spécification puis la conception des applications réactives temps-réel.

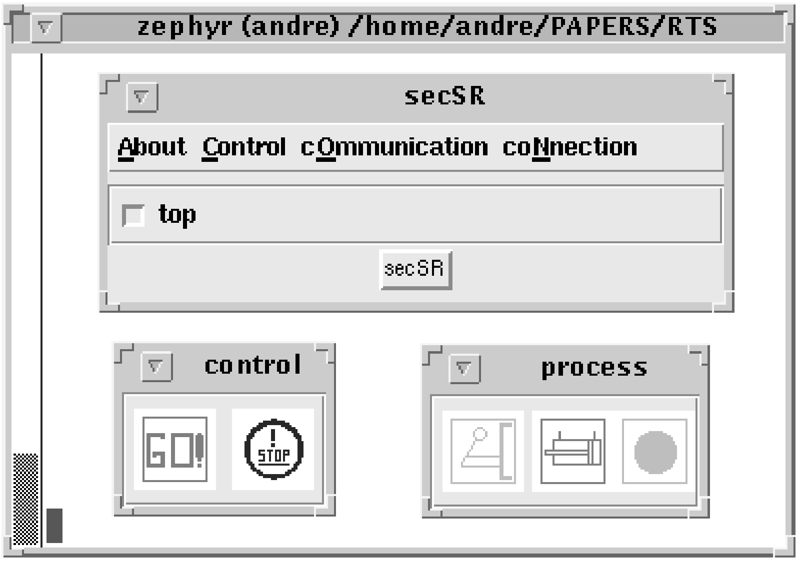


FIG. 10 - SOFA *: Simulation d'objets synchrones*

Pour réussir cette collaboration, nous les avons intégrés dans un environnement unique qui ne se contente pas de rassembler des outils. Il s'appuie sur une forte cohérence sémantique entre les modèles et permet ainsi le partage d'outils de simulation et de preuves formelles.

Ce que nous attendons d'un tel environnement :

– **Conception plus sûre des applications :** favorisée par l'usage de langages fondés sur des sémantiques rigoureuses, par l'enchaînement automatique des traitements internes et le recours à des outils de validation.

– **Gain de productivité :** grâce au prototypage rapide offert par les différents simulateurs, à la réutilisabilité des objets synchrones et à la convivialité de l'environnement.

Références

[ABC94] A. Arnold, D. Begay, and P. Crubillé. *Construction and Analysis of Transition Systems with* MEC, volume 3 of *AMAST Series in Computing*. World Scientific, Singapore, 1994.

[AFN82] AFNOR, Paris (France). *Diagramme fonctionnel* GRAFCET *pour la description des systèmes logiques de commande*, juin 1982. Norme Française, NF C 03 190.

[AG94] C. André and D. Gaffé. Evénements et conditions en GRAFCET. *A.P.I.I.*, 28(4):331–352, 1994.

[AG95] Charles André and Daniel Gaffé. Sequential function charts: a synchronous point of view. Technical Report RR 95–08, I3S, Sophia-Antipolis, France, february 1995.

[BB91] A. Benveniste and G. Berry. The synchronous approach to reactive and real-time systems. *Proceeding of the IEEE*, 79(9):1270–1282, September 1991.

[BD91] F. Boussinot and R. De Simone. The ESTEREL language. *Proceeding of the IEEE*, 79(9):1293–1304, September 1991.

[BdVN94] Frédéric Boulanger, Henri Delebecque, and Guy Vidal-Naquet. Intégration de modules synchrones dans un cycle de développement par objets. pages 245–260, Paris, 11–14Janvier 1994. RTS'94.

[Bou91] F. Boussinot. REACTIVE C: an extension of C to program reactive systems. *Software Practice and Experience*, 21(4), 1991.

[Bou93] F. Boulanger. *Intégration de Modules Synchrones dans la Programmation par Objets*. PhD thesis, Supélec / Université de Paris-sud, Centre d'Orsay, Décembre 1993.

[CHPR91] P. Caspi, N. Halbwachs, D. Pilaud, and P. Raymond. The synchronous data flow programming language LUSTRE. *Proceeding of the IEEE*, 79(9):1305–1320, September 1991.

[dV89] R. de Simone and D. Vergamini. Aboard AUTO. Technical Report 111, INRIA, October 1989.

[Hal94] N. Halbwachs. BAC: A boolean automaton checker. Technical report, VERIMAG, Montbonnot (France), February 1994.

[Har87] D. Harel. STATECHARTS: A visual formalism for complex systems. *Science of computer programming*, 8:231–274, 1987.

[HP85] D. Harel and A. Pnueli. On the development of reactive systems in logic and models of concurrent systems. *NATO ASI Series, K.R Apt Ed., Springer-Verlag*, 13:477–498, 1985.

[LGLL91] P. Le Guernic, T. Gautier, M. Le Borgne, and C. Le Maire. Programming real-time applications with SIGNAL. *Proceeding of the IEEE*, 79(9):1321–1336, September 1991.

[Ous94] J.K. Ousterhout. *Tcl and the Tk Toolkit*. Professional Computing Series. Addison-Wesley, 1994.

[Pas94] L. Pastorelli. Machine d'exécution pour le formalisme synchrone GRAFCET. Technical report, Ecole Doctorale SPI – DEA Informatique, Juin 1994. Rapport de stage de DEA.

[RCCE92] O. Roux, D. Creusot, F. Cassez, and J.P. Elloy. Le langage réactif asynchrone ELECTRE. *APII*, 11(5):35–66, 1992.