SyncCharts : un modèle graphique synchrone pour systèmes réactifs complexes

C. ANDRÉ[†] H. BOUFAÏED[†] S. DISSOUBRAY[‡]

[†] Laboratoire Informatique, Signaux, Systèmes (I3S)
Université de Nice-Sophia Antipolis / CNRS
41, bd Napoléon III – F – 06041 NICE Cedex
e-mail: andre@unice.fr

[‡] SIMULOG
Les Taissounières HB2
Route des Dolines – F – 06650 VALBONNE
e-mail: sylvan.dissoubray@simulog.fr

Cette présentation a été

- faite à RTS'98 (Real-Time Systems 1998), Paris, 14–16 janvier 1998.
- publiée dans les Actes de la conférence, p 175–193, Teknea.
- récompensée comme la meilleure présentation scientifique de la conférence.

Résumé

Les systèmes réactifs nécessitent communication, parallélisme et préemption. Peu de modèles supportent ces trois concepts. L'approche synchrone qui autorise leur coexistence est d'un intérêt majeur pour les systèmes réactifs.

Dans cette présentation nous introduisons un modèle graphique, appelé SYNCCHARTS, qui s'appuie sur le paradigme synchrone. Nous illustrons son emploi pour le contrôle d'un système automatisé. Nous introduisons également l'environnement de développement associé aux SYNCCHARTS.

Mots clefs : modèle graphique, systèmes réactifs, paradigme synchrone, SYNCCHARTS.

1 Introduction

Les systèmes réactifs

Un système réactif est un système dans lequel le contrôle est primordial. Ce type de système est en interaction permanente avec son environnement et il est souvent composé de sous-systèmes parallèles interagissants. La communication et la synchronisation sont essentielles pour assurer ces interactions. La *préemption* [1] joue également un rôle majeur. Nous nous intéressons plus particulièrement aux *contrôleurs* qui sont des systèmes réactifs soumis à des *contraintes temps-réel*. Les stimuli sont essentiellement des événements sporadiques et les comportements "exceptionnels" sont à traiter avec le plus grand soin.

L'approche synchrone

Les problèmes posés par la modélisation et la programmation des systèmes réactifs ne trouvent pas de solutions satisfaisantes dans le cadre de la programmation traditionnelle. Les *langages synchrones* [2], en particulier ESTEREL [3], ont été introduits pour répondre à ces problèmes :

- Prise en compte des *spécificités des systèmes réactifs* : le parallèlisme, la communication et la préemption sont des concepts de base du langage ESTEREL. Ses primitives réactives et son style impératif permettent d'exprimer naturellement des comportements réactifs complexes.
- *Fondements théoriques* : il ne s'agit pas uniquement d'une exigence académique. La nécessité d'établir des preuves formelles commence à s'imposer dans les milieux industriels pour les systèmes critiques. ESTEREL est basé sur une sémantique mathématique rigoureuse qui permet de prouver certains comportements des systèmes.
- *Passage à l'échelle* : les modèles théoriques doivent pouvoir être mis en œuvre de manière efficace sur des applications de taille industrielle. Les compilateurs ESTEREL récents font appel à des techniques performantes telles que les BDDs qui supportent des systèmes logiques de grande taille.

Les approches graphiques

Malgré toutes ces qualités, les langages synchrones restent peu utilisés dans les milieux industriels. Les structures de programmes telles que les `trap` et les diverses préemptions s'adressent plutôt à des spécialistes de programmation. Les *formalismes graphiques* qui paraissent plus simples à aborder, sont généralement préférés. Il en existe un grand nombre plus ou moins bien adaptés à la conception des systèmes réactifs. Le modèle GRAFCET [4] est largement utilisé pour la programmation des automatismes industriels. STATECHARTS [5] est un modèle hiérarchique qui fait partie de l'environnement STATEMATE et qui permet l'expression de comportements réactifs complexes. D'autres modèles semi-formels tels que SA/RT sont aussi largement utilisés pour la spécification des systèmes de grande taille. La méthodologie ROOM [6] pour applications temps réel, qui allie objets et réactivité sous une forme graphique, connaît actuellement un certain succès aux États-Unis. Toutefois, ces modèles ne sont pas aussi satisfaisants que le langage ESTEREL pour l'expression des différents comportements réactifs. De plus, la plupart de ces formalismes n'ont pas de sémantiques rigoureuses et ils se prêtent mal aux preuves formelles.

Contribution

Nos travaux portent sur l'introduction d'un **modèle graphique synchrone** : les SYNCCHARTS (*"Synchronous Charts"*) qui intègrent les concepts de haut niveau des langages synchrones dans un formalisme graphique expressif. Le développement de ce modèle devrait contribuer fortement à faire connaître et utiliser le paradigme synchrone dans les milieux industriels.

2 Modélisation des comportements réactifs

2.1 Un système réactif très simple

G. Berry propose dans son livre d'introduction au langage ESTEREL [10] un exemple très simple mais caractéristique de système réactif. Nous lui empruntons cet exemple connu sous le nom "ABRO".

Ce système a trois entrées A, B, R et une sortie O. O doit être émis dès qu'il y a eu occurrences de A et B, depuis l'instant initial ou depuis l'occurrence précédente de R. R est un signal prioritaire qui remet le système dans son état initial (Reset).

On trouve dans cet exemple un mélange de séquentialité, parallélisme et préemption :

- Contrôle *séquentiel* : O doit être émis *après* la réception de A et B,
- Contrôle *parallèle* : on attend les occurrences de A et B séparément,
- *Préemption* : l'occurrence de R fait avorter (immédiatement) les attentes de A et B.

2.2 Modèles

Machines de Mealy

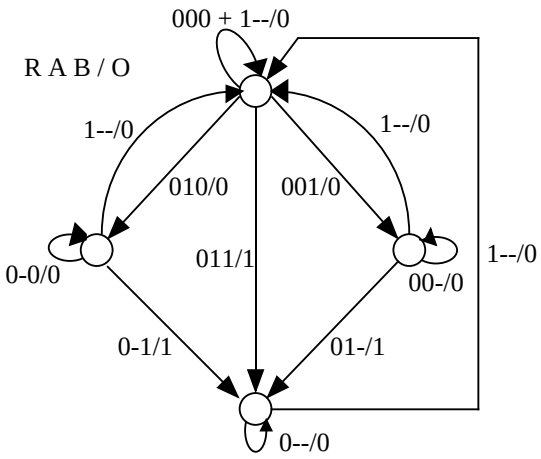


Figure 1: Modélisation par Machine de Mealy.

La machine de Mealy figure 1 est une modélisation possible de ce système. Bien que déterministe et mathématiquement bien-fondé, ce modèle présente de sérieuses limitations pour les systèmes réactifs :

1. pas de représentation directe du parallélisme,
2. pas de description hiérarchisée,
3. pas de représentation directe des préemptions.

Ces limitations font que le nombre d'état "explose" en présence de parallélisme et que les arcs deviennent pléthoriques en présence de préemption :
• si on attend non pas deux mais n signaux en parallèle, le nombre d'états devient 2^n,
• quant à la préemption par R elle nécessite un arc depuis tout état vers l'état initial.

Réseaux de Petri

Pour exprimer facilement le parallélisme, on peut utiliser des réseaux de Petri. Ce modèle n'est toutefois pas satisfaisant sur plusieurs points :

1. non déterminisme, dû à la nature asynchrone du modèle,
2. franchissement des transitions possible mais pas obligatoire (règles de franchissement),
3. pas de description hiérarchisée,
4. pas de représentation directe des préemptions.

Certaines extensions des réseaux de Petri répondent partiellement à ces exigences. Toutefois il ne s'agit plus du modèle de base et de nombreuses propriétés mathématiques sont perdues.

le Grafcet

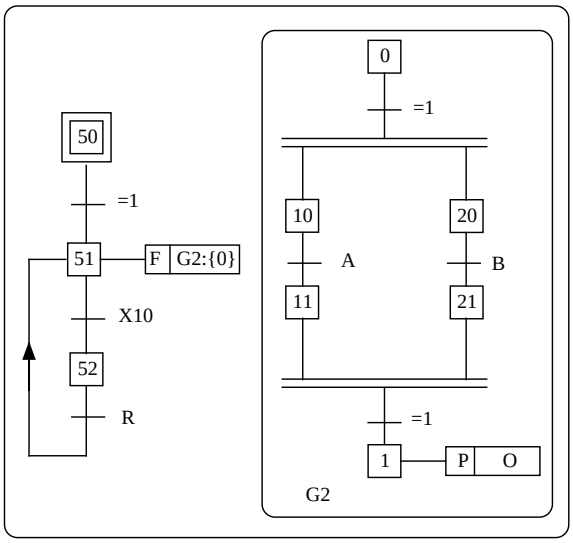


Figure 2: Modélisation par Grafcet.

Ce modèle, largement utilisé pour les automatismes logiques représente très clairement les séquences exécutées en parallèle. De plus ce modèle est synchrone ; plusieurs transitions peuvent être franchies simultanément. Des extensions du modèle permettent des descriptions hiérarchisées (grafcets partiels) et si on utilise la notion de *forçage* on peut

prendre en compte certaines formes de préemptions (Figure 2). Toutefois, la sémantique du GRAFCET n'est pas totalement compatible avec celle des langages synchrones, comme nous l'avons expliqué dans un article consacré aux problèmes sémantiques du GRAFCET [7]. Quant aux forçages ils font encore l'objet de discussions.

Statecharts

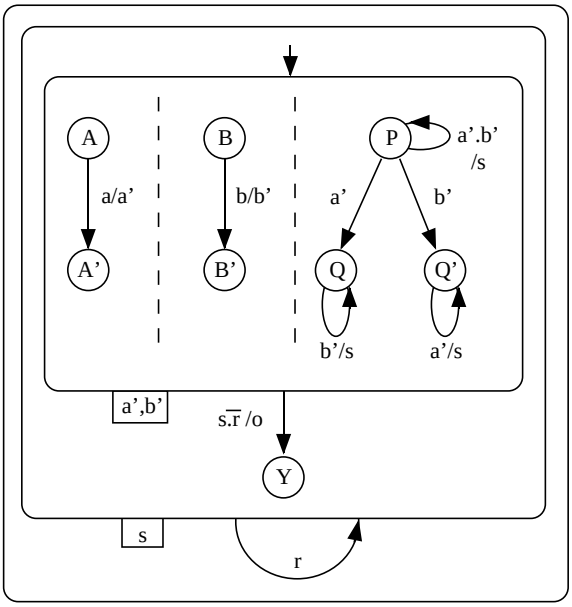


Figure 3: Modélisation par StateCharts.

Ce modèle est souvent employé pour les systèmes réactifs complexes. Il a d'ailleurs été créé dans ce but. Il sait exprimer le parallélisme, la hiérarchie et certaines formes de préemption. Les transitions sont étiquetés par des expressions de la forme

`trigger[condition]/effect`

qui conditionnent la sortie d'un état (préemption). Le `trigger` est une expression de test de présence (analogue des occurrences en ESTEREL), la `condition` est un prédicat qui doit être évalué à `true` pour que la transition puisse être franchie. L'`effect` correspond aux signaux à émettre lors du franchissement. Les franchissements sont instantanés.

Les STATECHARTS ne respectent pas totalement les hypothèses synchrones. Certaines configurations conduisent à des évolutions non déterministes. De plus, on dénombre plus d'une vingtaine de sémantiques différentes [8].

Nous avons modélisé dans la figure 3 le système ABRO à l'aide du formalisme ARGOS [9] qui reprend de nombreuses possibilités des statecharts mais avec une sémantique rigoureuse.

Cette modélisation exploite hiérarchie et parallélisme. Par contre, aussi bien les STATECHARTS qu'ARGOS ont une seule forme de préemption directement exprimable qui est l'avortement faible (weak abort). La préemption forte réalisée par R s'exprime au prix de

l'ajout d'automate de synchronisation (celui de droite sur la figure 3) et de modification des "triggers".

2.3 Notre proposition

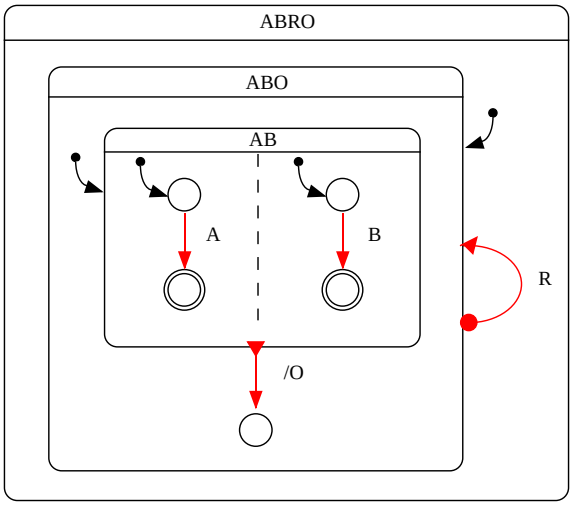


Figure 4: SyncCharts pour ABRO.

Les SYNCCHARTS reprennent des notations introduites dans les STATECHARTS. Comme le montre le syncChart de la figure 4, en présence de préemption forte et de terminaison normale, on obtient des représentations plus simples. Les divers éléments de ce modèle sont expliqués dans la section suivante.

3 SyncCharts

3.1 Modélisation par syncCharts

États-Transitions

Comme les modèles présentés précédemment, les SYNCCHARTS sont basés sur la notion d'état et de transitions entre états. Les informations (ou signaux) d'entrée provoquent les changements d'état (occurrence de A par exemple). Les signaux de sortie sont émis par le syncChart qui a à la fois les possibilités des machines de Mealy (le signal O est émis lors d'une transition) et des machines de Moore (un signal peut être *maintenu*, c'est à dire émis en permanence tant qu'un état est actif. L'exemple ABRO n'utilise pas cette possibilité, par contre, l'exemple d'automatisme étudié plus loin en fait grand usage).

Notons que certains états sont *initiaux* (repérés par des fléches venant d'un petit rond noir, appelé connecteur). Il peut exister des états *finals* dénotés par une enveloppe double. Ils jouent un rôle particulier dans la terminaison normale. Nous l'expliquerons dans le paragraphe sur les préemptions.

Hiérarchie

Comme pour les STATECHARTS, on peut *raffiner* un état. Les *macro-états* sont dessinés sous forme de rectangles à coins arrondis. Un macro-état a un nom (par exemple le macro-état AB et un corps qui est fait d'un ou plusieurs automates en parallèle.

Il est possible de définir des signaux locaux à un macro-état. ces signaux sont très utiles en synchronisation.

Parallélisme

Le macro-état AB contient deux composantes parallèles séparées par une ligne en pointillés (notation empruntée aux STATECHARTS).

Préemption

La préemption est la possibilité de tuer un agent ou bien de le suspendre temporairement. La première forme s'appelle *avortement* (abort), la seconde *suspension*.

La sémantique de la préemption a été étudiée par G. Berry [1]. Il ne s'agit pas d'un simple mécanisme comme dans les systèmes d'exploitation lorsque le processeur est enlevé à une tâche pour être attribué à une autre. La préemption est un concept de même niveau que le parallélisme. Un des apports majeurs de la programmation synchrone est justement d'avoir conçu séquentialité, parallélisme et préemption comme des *concepts orthogonaux*, c'est à dire qu'ils peuvent s'appliquer à tout niveau sans restriction. Un langage synchrone, comme ESTEREL, est capable de donner une sémantique très claire aux programmes mêlant parallélisme et préemption. En particulier les comportements peuvent rester déterministes.

Les SYNCCHARTS savent exprimer la suspension et les avortements. La suspension s'applique à un macro-état ou un état. Elle permet un véritable "figeage" [1] des évolutions internes du macro-état. La suspension est représentée par un arc venant d'un connecteur (point noir) et se terminant sur un état ou un macro-état par un rond rouge (en grisé dans le texte).

L'*avortement*, par contre, met définitivement fin à l'exécution d'un agent. On distingue deux formes d'avortement appelées respectivement faible et forte. Cette distinction est propre à l'approche synchrone qui s'appuie sur une notion précise d'*instant*. Les deux formes d'avortement sont directement exprimables en SYNCCHARTS. Ceci constitue un atout par rapport aux STATECHARTS. Un arc d'avortement a pour origine l'état ou le macro-état à préempter et a pour terminaison l'état ou macro-état suivant. L'*avortement fort* interdit toute exécution de l'agent préempté, dans l'instant de préemption. Il est représenté par un arc ayant comme origine un rond rouge (en grisé sur les figures). L'*avortement faible* laisse au contraire terminer l'instant avant la destruction effective ; l'agent tué peut, en fait, exprimer ses "dernières volontés". Graphiquement un arc d'avortement faible est un arc ordinaire.

Dans le macro-état AB, dans chaque composante on a un avortement faible respectivement par A et B. Le macro-état est avorté fortement par l'occurrence de R. Quand cette

[1] Ce terme est emprunté au GRAFCET.

préemption a lieu, le corps du macro-état est instantanément détruit et immédiatement réinstancié (à cause de la boucle) ; on se retrouve ainsi dans une nouvelle attente des occurrences de A et B, quoi qu'il se soit passé antérieurement.

L'arc sortant du macro-état AB est un arc de *terminaison normale*. On le distingue graphiquement par le triangle à son origine. Alors que les avortements sont provoqués explicitement par des "triggers", une terminaison normale est "spontanée". Elle résulte du fait que chacune des composantes du macro-état correspondant est dans un état final. Il est clair sur la figure 4 que c'est lorsque A et B se seront produits que le macro-état AB terminera. La terminaison est instantanée et s'accompagne ici de l'émission du signal O, qui est mentioné comme "effect" sur l'arc de terminaison normale.

Autres possibilités

L'exemple précédent est un exemple de synchronisation pure : les seules informations d'occurrences sont suffisantes. Dans la plupart des applications il faut également échanger des informations. Les SYNCCHARTS peuvent manipuler des *signaux valués*, c'est à dire que les signaux sont (fortement) typés. En plus de leur état de présence, ils sont porteurs d'une valeur. Les signaux, dans les modèles synchrones, sont *diffusés instantanément*. Toutes les composantes du modèle perçoivent donc les mêmes informations, au même instant. Les signaux sont donc les supports de la *communication*.

On peut également utiliser des variables typées pour mémoriser des informations et appeler des fonctions et procédures externes.

Les SYNCCHARTS ont reçu une sémantique mathématique pleinement compatible avec celle du langage ESTEREL. Une conséquence intéressante pour les concepteurs est qu'il est possible de mixer descriptions graphiques et descriptions textuelles (sous forme de code ESTEREL in-line).

3.2 Syntaxe

La figure 5 résume les éléments graphiques qui composent les SYNCCHARTS. Ils ont été illustrés partiellement dans l'exemple "ABRO". Les SYNCCHARTS ont été définis avec des vocables anglais. Les éléments de base sont *star*, *constellation* et *macro-state*[2].

Une étoile (star, Figure 5.B) est plus qu'un état : c'est un état et les façons de quitter cet état. Les arcs d'avortement issus d'un état constituent les "rayons" de l'étoile.

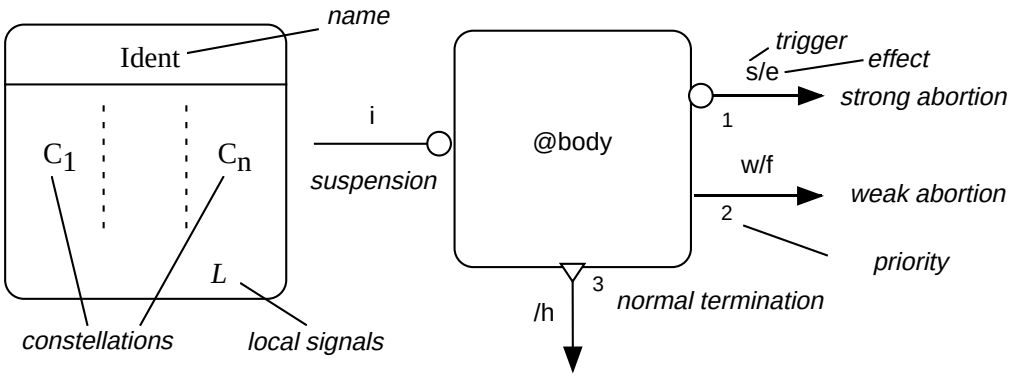
Une constellation (Figure 5.C) est un ensemble d'étoiles tel qu'une et une seule étoile est active à l'intérieur d'une constellation.

Un macro-état (Figure 5.A) est constitué d'une ou plusieurs constellations en parallèle.

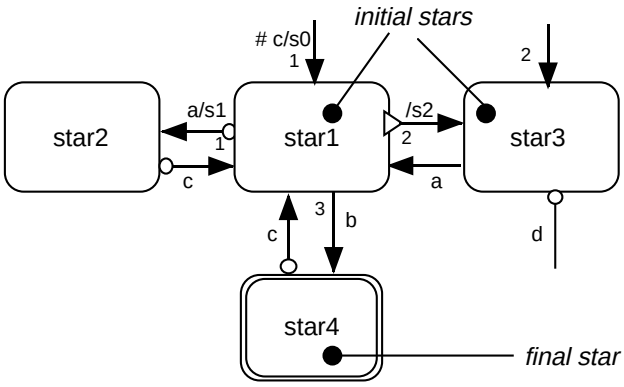
Les arcs sont de deux types :

- les arcs d'avortement, entre deux étoiles ou macro-états,
- les arcs auxiliaires, depuis un connecteur vers une étoile ou un macro-état.

[2]Un macro-état est parfois appelé "firmament" pour des raisons d'homogénéité.



(A) Firmament (B) Star



(C) Constellation

Figure 5: SYNCCHARTS: Éléments de syntaxe

Les arcs sont étiquetés. La forme générale d'une étiquette est :

`trigger`[`condition`]/`effect` avec

- `trigger` : une expression booléenne (utilisant les opérateurs `and, or, not` et des parenthèses) portant sur la présence des signaux,
- `condition` : un prédicat pouvant faire intervenir des valeurs de signaux,
- `effect` : une liste de signaux, éventuellement valués, à émettre lors de la transition.

Ces différents champs sont optionnels. Le défaut pour le trigger est le signal `tick`, un signal prédéfini présent à chaque instant. Le trigger peut avoir un effet immédiat si il est préfixé par le symbole #.

`A; /O; T [?T > nextTime]` sont des exemples d'étiquettes. Les deux premières ont déjà été vues. La troisième déclenche la transition sur l'occurrence du signal `T` si la valeur de ce signal (disons de type `time`) est supérieure à la valeur de la constante `nextTime` du même type.

Un syncChart a un *comportement entièrement déterministe*. Dans le cas où il existe plusieurs façons de quitter une étoile, on doit donner un ordre d'évaluation. Les entiers $1, 2, \cdots, n$ situés près de l'origine d'un arc précisent la priorité de celui-ci (le plus petit entier est le plus prioritaire)[3].

4 Plate-forme de développement

Le formalisme SYNCCHARTS est développé en tant que prototype de recherche de l'I3S et industrialisé en parallèle par la société SIMULOG.

4.1 Environnement de développement

Un environnement de programmation a été développé pour les SYNCCHARTS. Il comprend un éditeur graphique et un compilateur vers le langage ESTEREL. Il intègre également les appels au compilateur ESTEREL, à un compilateur C et au simulateur XES qui permet l'exécution interactive de programmes ESTEREL avec remontée dans le source.

L'éditeur/compilateur est écrit en TCL-TK [11].

Le compilateur ESTEREL doit être de version 5.01 ou supérieure, le processeur `sc-causal` de la chaîne de compilation ESTEREL est également nécessaire pour certains syncCharts.

Le statut actuel de l'éditeur/compilateur de SYNCCHARTS est celui d'un prototype de recherche. Il nous a permis des expérimentations sur des exemples d'école et sur des automatismes plus complexes comme la cellule de production présentée ci-dessous et le contrôle d'un processus physique publié par ailleurs [12].

La version actuelle v2.082 est opérationnelle et disponible pour les universitaires. Elle est programmée en TCL-TK (Tk4.1).

Afin que le lecteur puisse se faire une idée de l'interface offerte, nous avons fait des copies d'écran de SYNCCHARTSv2, sur lesquelles nous avons rajouté des commentaires.

Panneau principal : Ce panneau donne accès par des menus déroulants aux différents services précisés sur la figure 6.

Arbre des macro-états : Ce panneau visualise l'arbre de l'application et offre des possibiltés de modification de la hiérarchie, par élagage ou greffe de sous-arbres (Figure 7). Il permet également l'accès direct à un macro-état.

Dessin : Chaque macro-état est dessiné dans un canvas indépendant. Une série de boutons permet de choisir les principales fonctions d'édition. Depuis un macro-état on peut remonter vers son ancêtre ou ouvrir un de ses descendants. Le dessin d'un macro-état peut être iconifié ou même retiré à tout moment.

[3] Avec la version v2, la règle a été modifiée : les avortements forts sont plus prioritaires que les avortements faibles, qui sont eux-mêmes plus prioritaires que la terminaison normale. Les priorités permettent de choisir parmi les avortements de même "force" (faible ou fort).

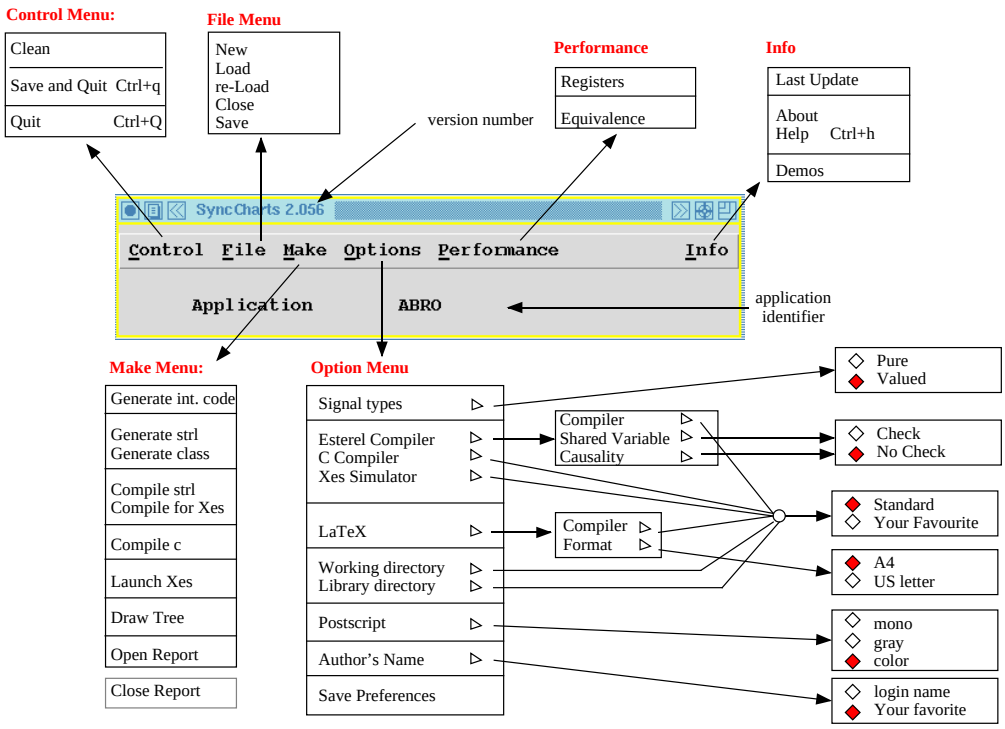


Figure 6: SYNCCHARTS : panneau principal.

4.2 Industrialisation

SIMULOG industrialise l'environnement prototype de recherche existant. La plate-forme de développement commerciale est réalisée en Ilog Views et sera disponible au premier trimestre 98 sur station de travail Unix et PC/NT.

L'éditeur

L'éditeur graphique permet d'entrer les diagrammes SYNCCHARTS, de vérifier un modèle, de générer du code pour simulation ou du code optimisé pour l'exécution sur plusieurs langages cibles (C/C++, Java, assembleur pour micro-contrôleurs et DSP, ...). L'éditeur

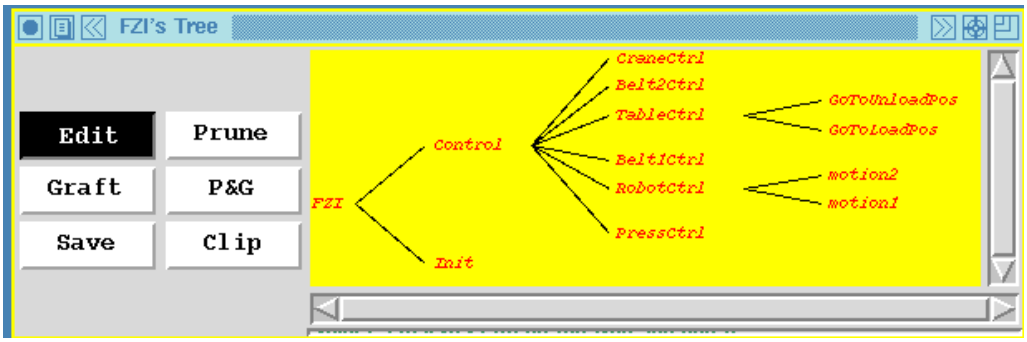


Figure 7: SYNCCHARTS : arbre.

permet de gérer des bibliothèques de composants métier (robotique et production automatisée, protocoles de communication, drivers, traitement du signal, interfaces homme-machine,...).

Le générateur de documentation

Le générateur de rapports est un outil de communication essentiel pour valider la conception et sa conformité par rapport aux spécifications. Il permet de contrôler les vues et d'aggréger ou d'expanser les macro-états à la demande.

Mise au point

Pour la mise au point, on utilise un simulateur interactif de SYNCCHARTS. Celui-ci permet, grâce à un panneau de contrôle, d'entrer des événements (signaux et valeurs) et d'observer les changements d'états directement sur le graphique, en allumant les états actifs et les transitions empruntées. Un outil magnétophone permet de plus d'enregistrer et de rejouer des séquences, notamment pour comparer les comportements de versions successsives d'un modèle.

4.3 Vérification

Les outils de vérification du langage ESTEREL (XEVE) sont mis à disposition du concepteur de système. Ils permettent de s'assurer de la satisfaction de propriétés de sûreté et de vivacité du modèle. On peut exprimer des propriétés d'atteignabilité, de simulatnéïté ou d'exclusion mutuelle de certains états. L'analyse couvre exhaustivement tout l'espace d'états et permet ainsi de détecter des problèmes parfois difficiles à mettre en évidence à l'aide du test. En cas de violation d'une propriété, une séquence d'entrée menant à la situation indésirée est fournie et peut être directement examinée pas à pas à l'aide du simulateur.

5 Contrôle d'un atelier de production automatisé

5.1 Présentation de l'application

Il s'agit d'un atelier d'usinage de pièces métalliques. Cet exemple a été proposé par les chercheurs de la FZI [4] comme modèle pour l'évaluation des différentes méthodes formelles de développement des systèmes réactifs [13]. L'atelier existe réellement, mais nous utilisons pour le test un logiciel de simulation qui a été développé à la FZI. C'est une fenêtre graphique TCL-TK (figure 8) qui simule les différents dispositifs de l'atelier et leurs évolutions.

L'atelier est constitué de 6 éléments : un convoyeur d'alimentation, une table rotative-élévatrice, un robot à deux bras, une presse, un convoyeur de dépôt et une grue mobile.

[4] Forschungszentrum Informatik de Karlsruhe

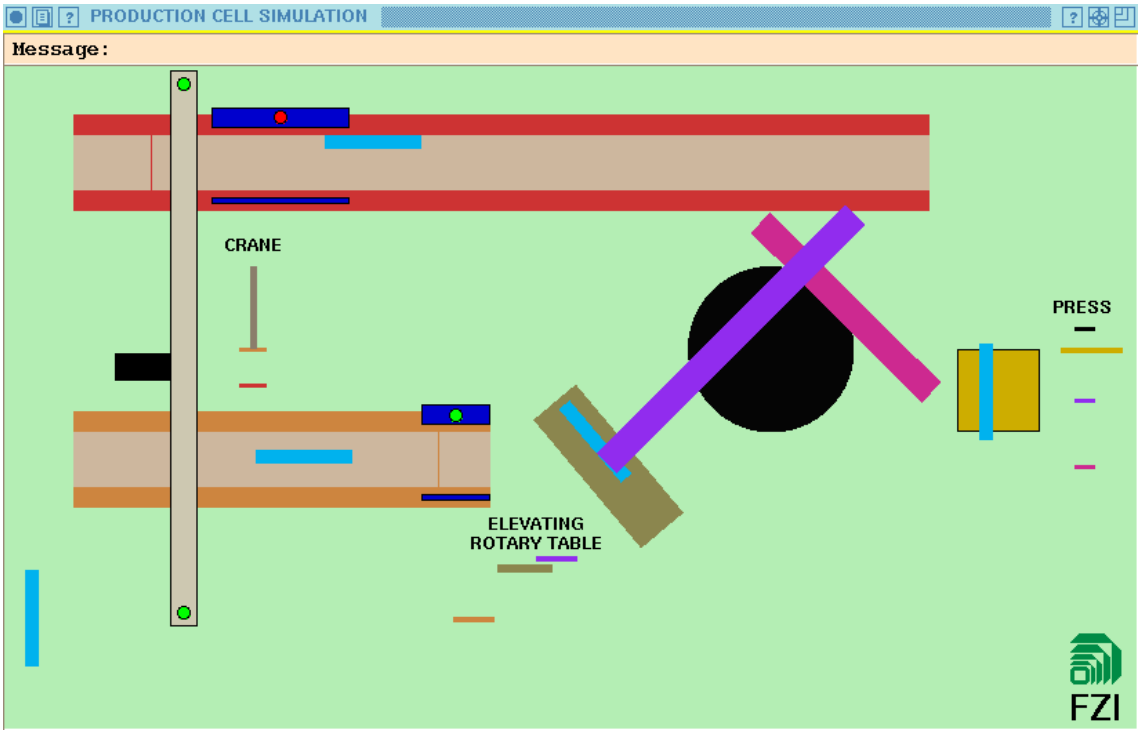


Figure 8: Représentation graphique de l'atelier

Le processus d'usinage se déroule de la manière suivante : chaque pièce est acheminée par le premier convoyeur vers la table rotative. Le robot est ensuite chargé de la transférer vers la presse. Une fois la pièce pressée, le robot effectue le transfert vers le deuxième convoyeur. Cet atelier étant conçu pour l'expérimentation, le processus doit pouvoir évoluer sans l'intervention d'un opérateur extérieur. Ainsi, la presse ne modifie pas réellement la forme des pièces et celles-ci sont réintroduites dans la chaîne en fin de parcours : la grue mobile déplace les pièces usinées du convoyeur de dépôt vers le convoyeur d'alimentation. La tâche du contrôleur est de commander les différents dispositifs et d'enchaîner les différentes opérations :

- Convoyeurs : ils peuvent être mis en marche ou arrêtés. Une cellule photo-électrique installée à leur extrémité indique si une pièce est entrée ou sortie de la partie terminale.
- Table rotative-élévatrice : son rôle est de faire pivoter les pièces de 45^o et de les porter à une hauteur où elles peuvent être saisies par le premier bras du robot.
- Robot : il comprend deux bras orthogonaux. Pour des raisons techniques, les deux bras ne sont pas à la même hauteur. Chaque bras peut s'étendre ou se rétracter horizontalement. Ils sont solidaires du même axe autour duquel ils peuvent pivoter. Les extrémités des deux bras sont pourvues d'électro-aimants qui leur permettent de saisir les pièces métalliques. La tâche du robot est de transférer les pièces de la table rotative vers la presse et de la presse vers le convoyeur de dépôt.

- Presse : elle est constituée de deux plateaux horizontaux, le plateau inférieur pouvant se déplacer verticalement pour forger les pièces.
- Grue mobile : son rôle est de ramasser les pièces du convoyeur de dépôt et de les décharger sur le convoyeur d'alimentation. Un électro-aimant permet la saisie des pièces. Il peut se déplacer horizontalement (d'un convoyeur vers l'autre) et verticalement (les convoyeurs ne sont pas à la même hauteur).

Le contrôleur connaît l'état du système grâce à un ensemble de capteurs (14 au total). Ce sont soit des capteurs *tout-ou-rien* (contacts ou cellules photo-électriques), soit, pour certains dispositifs, des potentiomètres (angle de rotation de la table, extension des bras du robot,...).
Les mouvements des dispositifs sont commandés par 13 actionneurs.

5.2 Conception du contrôleur en syncCharts

Le contrôleur

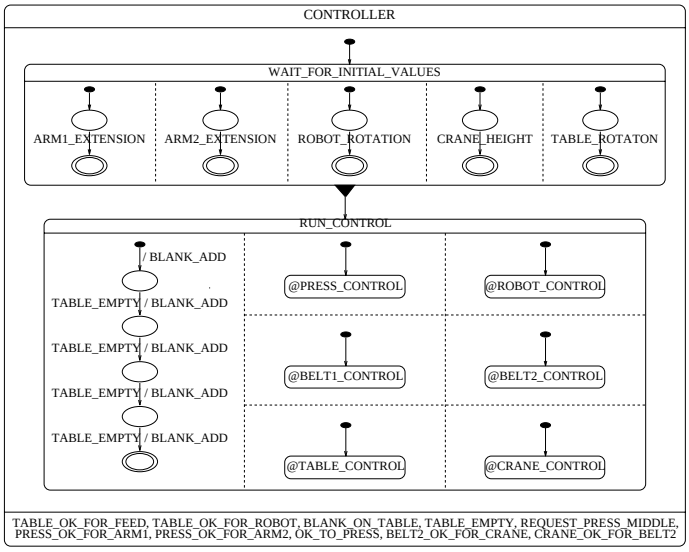


Figure 9: SyncChart du contrôleur

Les particularités de cet atelier font qu'une décomposition fonctionnelle peut se calquer sur une décomposition structurelle. On distingue différentes unités comme le robot, la grue, ... À chaque unité est associé son propre contrôleur. La coordination de l'ensemble se fait par échange de signaux entre les contrôleurs. Ainsi l'interface d'un contrôleur comprend les signaux échangés avec les capteurs et les actionneurs du dispositif, plus des signaux de synchronisation. On a donc 6 contrôleurs : `belt1_control`, `belt2_control`, `table_control`, `robot_control`, `press_control` et `crane_control`.
La figure 9 représente le syncChart de niveau 0 (*toplevel*) du contrôleur global. Il contient 2 macro-états en séquence : `WAIT_FOR_INITIAL_VALUES` spécifie la phase d'initialisation

et RUN_CONTROL correspond au contrôle proprement dit. Le passage de la phase d'initialisation à la phase de contrôle se fait spontanément, dès que la première phase est terminée (utilisation d'un arc de terminaison normale). Le cahier des charges du simulateur ne prévoit pas de phase d'arrêt. Celle-ci pourrait se faire en ajoutant un arc d'avortement menant de RUN_CONTROL à un macro-état d'arrêt. Grâce à l'orthogonalité de la préemption, cette modification ne remettrait pas en cause le corps du macro-état RUN_CONTROL.

La phase d'initialisation consiste à attendre que tous les capteurs numériques aient communiqué leur valeur initiale. Chacune des 5 branches parallèles attend l'occurrence d'un signal (ARM1_EXTENSION,...) puis passe dans un état final. Quand les 5 constellations sont dans leur état final, le macro-état WAIT_FOR_INITIAL_VALUES termine normalement.

Le macro-état RUN_CONTROL comprend une constellation pour chaque contrôleur. La constellation supplémentaire, à gauche sur la figure, se charge d'introduire 5 pièces dans la chaîne (BLANK_ADD). Les signaux de synchronisation entre les différents contrôleurs sont déclarés comme des signaux locaux et mentionnés dans la partie inférieure du macro-état. Ces signaux ne proviennent pas de capteurs ; ils ont été introduits par le concepteur.

Chaque macro-état qui apparaît dans le macro-état RUNCONTROL, fait l'objet d'une description séparée. À titre d'exemple, nous allons détailler le contrôleur de la table rotative.

La table rotative-élévatrice

Le syncChart du contrôleur de la table est représenté figure 10. La table transfère des pièces du convoyeur d'alimentation vers le robot. Elle alterne entre 2 positions : une position basse dans laquelle elle est prête à recevoir une pièce provenant du convoyeur et une position haute qui permet au bras du robot de saisir la pièce. La table doit d'abord se mettre en position de chargement (macro-état GoToLoadPos) puis signaler sa disponibilité au convoyeur. Cette signalisation est faite en SYNCCHARTS par le maintien du signal de synchronisation TABLE_OK_FOR_FEED. Une fois la pièce chargée (occurrence du signal BLANK_ON_TABLE), la table doit se mettre en position de déchargement (GoToUnloadPos) puis signaler sa disponibilité au robot (signal TABLE_OK_FOR_ROBOT). Une fois la pièce enlevée (occurrence du signal TABLE_EMPTY), la table retourne en position initiale et ainsi de suite.

Les macro-états GoToLoadPos et GoToUnloadPos ont des structures et des fonctionnalités analogues. Les SYNCCHARTS permettent de définir des modules génériques et de les instancier.

Macro-état GoToLoadPos

Le macro-état GoToLoadPos (figure 11) comporte 2 branches parallèles : une pour le déplacement vertical et une pour la rotation. On doit sortir de ce macro-état dès que les deux branches sont terminées. Pour spécifier ce comportement, les SYNCCHARTS utilisent les étoiles finales.

Comportement de chaque branche :

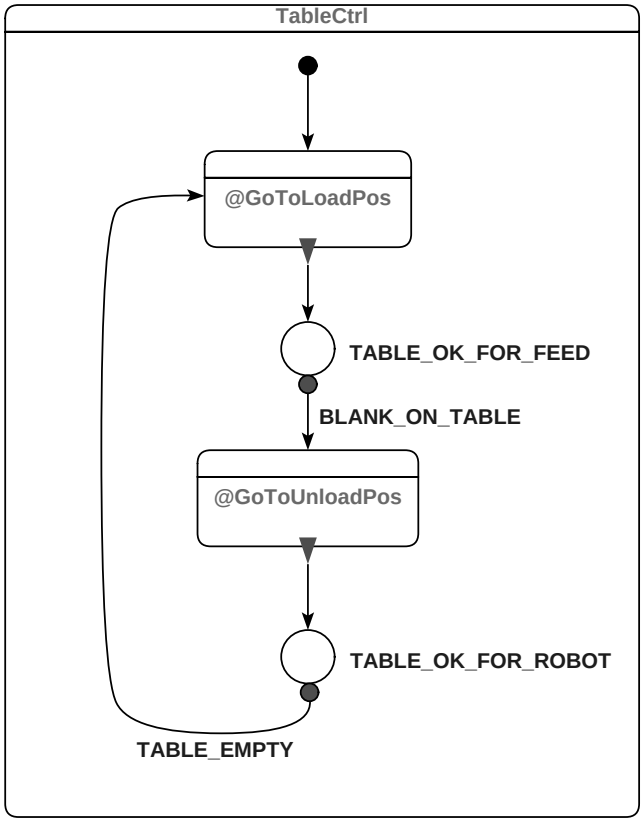


Figure 10: SyncChart du contrôleur de la table

- *La position verticale* de la table est connue grâce à 2 contacts : `TABLE_AT_TOP` et `TABLE_AT_BOTTOM`. Les signaux `TABLE_UPWARD`, `TABLE_DOWNWARD` et `TABLE_STOP_V` commandent les déplacements verticaux de la table.

  Positionner un dispositif physique en un point donné est une tâche qui revient souvent dans les automatismes. Pour spécifier ce comportement, nous avons créé un macro-état générique appelé `basicCtrl` (voir figure 12).

  Le principe est de déclencher le mouvement d'un dispositif (signal de commande `START`) et de l'arrêter (signal de commande `STOP`) quand la position désirée a été atteinte (occurrence du signal `REACHED`). Si le dispositif est initialement dans la position voulue, le signal `REACHED` est immédiatement pris en compte (signal déclencheur précédé du # sur le syncChart) et l'état final est atteint sans émission de `START`.

- Pour *la rotation*, le macro-état `rotateToBelt` est aussi une instance d'un module générique "`servoCtrl`". Ce module a été conçu en SYNCCHARTS séparément et traduit automatiquement en ESTEREL. Nous le réutilisons dans cette application en faisant des "`run`" du module ESTEREL.

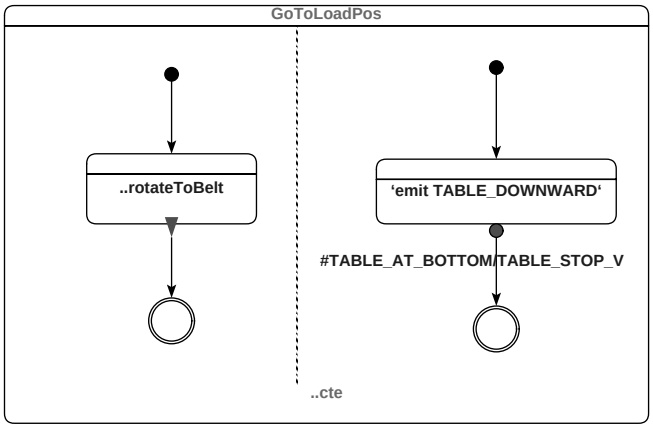


Figure 11: macro-état `GoToLoadPos`

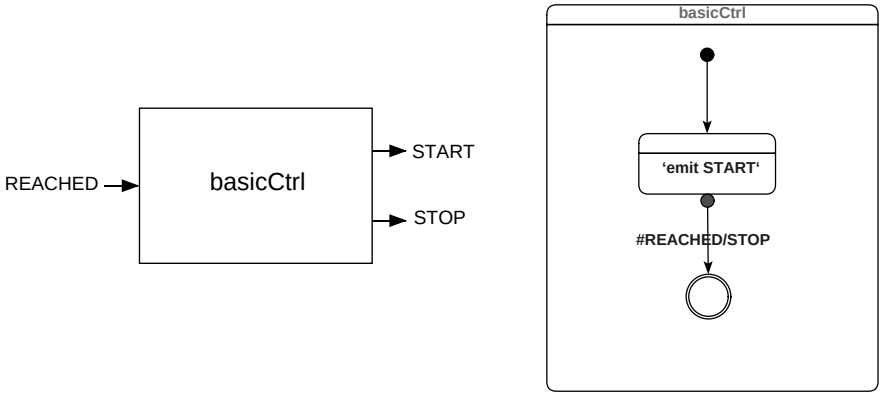


Figure 12: Module générique basicCtrl

Ce module a été défini pour commander les appareils dont la position est repérée par un capteur numérique. Il peut être instancié avec 3 signaux de commande purs, un capteur numérique d'entrée et une valeur numérique *cible*. Il est utilisé pour amener un dispositif dans la position cible à l'aide des 3 commandes d'actionneurs (typiquement avancer, reculer et arrêter).

Il est instancié ici avec les signaux `TABLE_ROTATION`, `TABLE_RIGHT`, `TABLE_LEFT`, `TABLE_STOP_H` et la constante `TABLE_ANGLE_FOR_BELT` pour asservir la position angulaire de la table. Ce module est aussi utilisé dans les contrôleurs du robot et de la grue.

Remarque : certains modules génériques peuvent être directement exprimés en code ESTEREL, ce qui permet de réutiliser des conceptions antérieures.

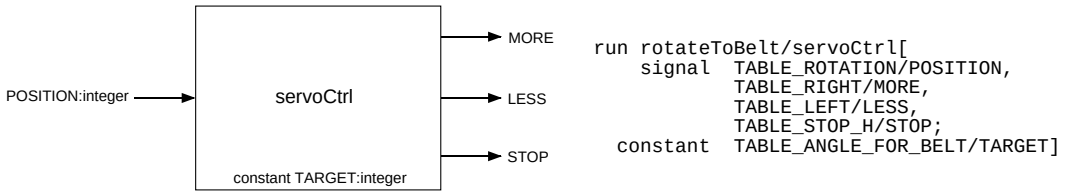


Figure 13: Module générique `servoCtrl`

5.3 Couplage avec la simulation

Le simulateur graphique de l'atelier est un processus Unix qui lit les commandes à destination des actionneurs sur `stdin` et écrit les valeurs des capteurs sur `stdout`. Le format textuel de ces informations est clairement spécifié par les concepteurs du simulateur. Le couplage entre le contrôleur et la simulation nécessite donc d'établir la correspondance entre ces informations textuelles et les signaux ESTEREL du contrôleur.
Ce problème entre dans un cadre plus large qui est celui des *machines d'exécution* dont le rôle est d'exécuter des modules synchrones dans des environnements asynchrones. Une architecture générique de machine d'exécution et une bibliothèque de *handlers* qui permettent la connexion des signaux ESTEREL à l'environnement ont été développées. Un générateur produit le code exécutable de la machine à partir d'un format textuel de configuration.
La modélisation SYNCCHARTS du contrôleur est d'abord compilée en ESTEREL puis en C++. Pour le couplage à la simulation, la machine du contrôleur est configurée avec des handlers sur *pipe* Unix en entrée et en sortie. Des modules de codage/décodage sont chargés de multiplexer et de démultiplexer les signaux sur le pipe.

Une fois le contrôle mis au point par simulation interactive, le couplage avec l'atelier réel se fait en modifiant le type des "handlers". Les handlers pour entrées/sorties physiques remplacent les handlers Unix. Ce sont les seules informations à modifier. Il suffit ensuite de recompiler la machine d'exécution.

6 Conclusion

Les domaines d'application temps-réel et embarquées évoluent rapidement. Les développeurs sont demandeurs d'environnement de développement qui apportent

- des moyens de description conviviaux et puissants, en particulier pour exprimer des comportements complexes,
- des générateurs de code sûrs et efficaces,
- des outils de mise au point qui permettent de tracer les exécutions directement dans le formalisme source,
- ainsi que l'accès à des systèmes de preuves formelles avec possibilité d'expression des propriétés dans le formalisme d'entrée lui-même.

Les SYNCCHARTS, qui font une synthèse entre les formalismes graphiques de haut niveau largement utilisés comme les StateCharts et la rigueur du langage synchrone Esterel, apportent une réponse dans le domaine des applications où les aspects *contrôle* sont dominants.

Le problème que l'on rencontre souvent avec ces modèles mathématiquement bien-fondés est qu'ils restent avant tout "académiques". Ce n'est pas le cas pour les SYNCCHARTS qui s'appuient sur l'environnement ESTEREL et font l'objet d'une action d'industrialisation. Ils devraient ainsi contribuer à accroître la productivité de conception des applications réactives ainsi que leur sûreté.

References

[1] G. Berry. Preemption in concurrent systems. *Proc FSTTCS, Lecture notes in Computer Science*, 761:72–93, 1992.

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

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

[4] IEC, Genève (CH). *Preparation of Function Charts for control systems*, december 1988. International standard IEC 848.

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

[6] B. Selic, G. Gullekson, and P.T. Ward. *Real-Time Object-Oriented Modeling*. J. Wiley Publ., 1994.

[7] C. André et D. Gaffé. Evénements et Conditions en GRAFCET. *APII*, 28(4):331–352, 1994.

[8] M. von der Beeck. A comparison of STATECHARTS Variants. Proc. of Formal Techniques in Real Time and Fault Tolerant Systems, LNCS 863, pp 128–148, Springer-Verlag, 1994.

[9] F. Maraninchi. ARGOS*: un langage graphique pour la conception, la description et la validation des systèmes réactifs.* PhD thesis, Université Joseph Fourier, Grenoble I, Janvier 1990.

[10] G. Berry. *The Esterel v5 Language Primer*. not yet published, available on the web, `www.inria.fr/equipes/meije/esterel`, Sophia Antipolis (F), 1997.

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

[12] C. André, M. Bourdellès, and S. Dissoubray. Un environnement graphique pour la spécification et la programmation d'applications réactives complexes. In *Actes GL97*, number 46 in GL97, Paris (F), décembre 1997. Génie Logiciel.

[13] C. Lewerentz and T. Lindner. *Formal Development of Reactive Systems*, volume 891 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 1995.