SyncCharts/Estérel : un environnement graphique pour la spécification et la programmation d'applications réactives complexes.

C. André
I3S (UNSA)
andre@alto.unice.fr

M. Bourdellès
CMA (ENSMP)
michel.bourdelles@cma.cma.fr

S. Dissoubray
SIMULOG
sylvan.dissoubray@simulog.fr

Résumé

On présente ici l'environnement graphique de conception d'applications réactives complexes SyncCharts/Estérel. SyncCharts est un formalisme graphique de description d'automates parallèles hiérarchiques. Estérel est le langage réactif synchrone que l'on utilise. L'apport de SyncCharts/Estérel réside dans l'efficacité du code cible généré et les performances des techniques de vérification utilisées. Ces gains ont été obtenus grâce aux fondements sémantiques des SyncCharts et aux récents progrès du compilateur Estérel V5.

1. Introduction

Le langage synchrone Estérel [2], conçu par l'école des Mines et l'INRIA, est spécialisé dans la programmation réactive des logiciels temps-réel et embarqués à dominante contrôle. La méthode de compilation procède par traduction en équations booléennes, ce qui permet d'utiliser les algorithmes de vérification et d'optimisation temps/place classiques en CAO électronique. Estérel est maintenant arrivé à maturité pour une exploitation industrielle sur des systèmes complexes, depuis la phase de spécification jusqu'à la production de code embarqué (C/C++, Java, Ada, VHDL et pour micro-contrôleurs).

Ce constat a motivé l'action d'industrialisation d'Estérel V5 que Simulog a engagée en 1997, en partenariat avec Dassault-Aviation et Thomson. Pour faciliter la diffusion et l'utilisation d'Estérel auprès d'automaticiens et d'architectes système, Simulog réalise l'outil graphique SyncCharts [3] (Synchronous Charts) permettant de décrire des automates communicants parallèles et hiérarchiques dans le style des Statecharts [4], avec des possibilités accrues. En effet, les fondements sémantiques des SyncCharts, conçus à l'université de Nice, offrent des possibilités réelles de preuves et permettent de générer du code efficace.

Les atouts de l'environnement SyncCharts/Estérel sont la richesse des mécanismes de préemption, l'approche mixte textuelle et graphique, les mécanismes de réutilisation et de paramétrage des composants et l'intégration des outils (simulation, vérification, optimisation, génération de code). SyncCharts/Estérel contribue ainsi à la productivité et à la sécurité de la programmation des applications réactives.

Le cas test de la turbine à vapeur (« steam-boiler ») illustre l'utilisation de l'environnement SyncCharts/Estérel. Ce problème de taille raisonnable, déjà traité à l'aide de nombreuses méthodes formelles (B, Z, Statecharts, Lustre) [1], permet de mettre en évidence les apports spécifiques et les avantages de SyncCharts/Estérel.

2. L'approche synchrone des systèmes réactifs

2.1 Les systèmes réactifs

Les systèmes réactifs sont en interaction permanente avec leur environnement. Les domaines d'application concernent les systèmes temps-réels et embarqués, le traitement du signal, les contrôleurs de processus, les protocoles de communication, la robotique, les contrôleurs de périphériques, etc. La complexité de ces applications et les contraintes sévères en temps ou sécurité auxquelles elles sont soumises imposent des méthodes de conception adaptées pour atteindre les niveaux de productivité et de sécurité requis.

2.1.1 Les langages synchrones

Les langages et formalismes synchrones (Estérel, Lustre, Signal, Statecharts et Argos) répondent aux besoins spécifiques de la programmation des systèmes réactifs en suivant un modèle synchrone. Essentiellement, cette approche suppose que la réaction du système est synchrone avec l'événement qui l'a déclenchée, ou encore que la réaction a lieu de manière atomique, en un seul cycle d'horloge.

2.1.2 Le langage Estérel

Grâce à des techniques qui s'appuient sur les sémantiques mathématiques du langage, un programme Estérel correct peut être transformé en un circuit électronique séquentiel.

Le compilateur Estérel V5 exploite cette analogie et génère deux formats de code interchangeables. La forme circuit est composée de logique combinatoire et de registres, tandis que sa version logicielle équivalente est un système d'équations booléennes avec variables. Le format logiciel est aisément traduit dans plusieurs langages cibles compte tenu de sa structure simple (C/C++, Java, Ada, VHDL, micro-contrôleurs).

Le modèle synchrone confère au langage Estérel un avantage important : il permet, même en présence de parallélisme, de détecter les programmes qui expriment un comportement non déterministe ou non réactif (qui se bloquent).

La compilation sous forme circuit permet de réutiliser des algorithmes d'optimisation et de vérification issus de la technologie de la CAO électronique qui donnent des résultats sur des systèmes de taille importante. Ces techniques permettent de détecter des dysfonctionnements qui pourraient rester invisibles avec une exploration par simulation.

3. Les SyncCharts

3.1 Origines

Les concepts nouveaux du modèle sémantique et la syntaxe inhabituelle du langage Estérel sont un frein à son utilisation par un public d'architectes système et d'automaticiens. Il fallait donc fournir un formalisme graphique adapté, plus proche des outils couramment utilisés dans l'industrie tels que le Grafcet, les réseaux de Petri, SA/RT ou les Statecharts.

Le formalisme retenu, SyncCharts, conçu à l'I3S [3], est largement inspiré des Statecharts. En s'appuyant sur l'existence d'un prototype recherche d'éditeur/compilateur de SyncCharts, et en partenariat avec Dassault-Aviation et Thomson, la société Simulog réalise un produit commercial de conception de systèmes réactifs (disponible au premier semestre 98).

L'objectif est de mettre à disposition un environnement qui intègre d'une part la flexibilité des SyncCharts pour la conception et la simulation en phase amont, d'autre part les apports d'Estérel pour l'efficacité du code embarquable généré et la vérification de propriétés par exploration exhaustive de l'espace d'états.

3.2 Syntaxe

3.2.1 Automates

Un modèle SyncCharts est composé d'étoiles, représentant les états de l'automate, et de transition entre ces étoiles. Les étoiles peuvent être des états simples (cercles) ou des macro-états (rectangles), définis hiérarchiquement. Une transition entre états est représentée par un arc ; il peut s'agir de préemption faible (arc simple), forte (commençant par un disque rouge) ou de terminaison normale (commençant par un triangle rouge). Les étoiles peuvent être finales, ce qui est indiqué par une enveloppe en trait double.

3.2.2 Parallélisme

Un macro-état peut contenir plusieurs composantes parallèles, séparées par des pointillés. Chaque composante parallèle possède un unique connecteur initial (disque noir plein) qui détermine le point d'entrée dans l'automate lorsque le macro-état englobant devient actif. Plusieurs transitions initiales peuvent partir de ce connecteur.

3.2.3 Communication

Dans l'approche synchrone, toutes les composantes perçoivent simultanément les mêmes informations diffusées par des signaux (ceux-ci sont présents ou non, et peuvent porter des données). On peut toutefois restreindre la portée d'un signal en le rendant local à un macro-état.

3.3 Comportement dynamique

Les transitions sont étiquetées sous la forme déclencheur/effet, où le déclencheur est une combinaison booléenne de signaux, et l'effet consiste par exemple à émettre des signaux. Lorsque les signaux rendent un déclencheur passant, la transition est activée, et l'état courant change. Une émission de signal dans une composante parallèle peut déclencher une évolution dans une autre composante, et ainsi de suite. La sémantique des SyncCharts assure que ces évolutions sont cohérentes et qu'un nouvel état stable est atteint.

La modélisation consiste à décrire le comportement des sous-systèmes individuels comme des automates évoluant en parallèle et synchronisés par les émissions de signaux. Cette méthode permet d'exprimer des comportements complexes de manière modulaire.

4. Un exemple : contrôle d'une turbine à vapeur

4.1 Spécification

Ce cas d'étude décrit le comportement d'un contrôleur de turbine à vapeur. Le fonctionnement de cette turbine a été spécifié par J.R. Abrial, E. Borger et H. Langmaack [1]. La turbine se compose de plusieurs éléments : le corps de la turbine proprement dite, une valve qui permet de vidanger la turbine et quatre pompes. Il s'agit pour le contrôleur d'ouvrir et fermer la valve et les pompes afin d'assurer que le niveau d'eau ne devienne pas critique.

Le contrôleur connaît l'état du système grâce à plusieurs capteurs. Tout d'abord deux capteurs donnent le niveau d'eau et le débit de vapeur d'eau de la turbine. L'état de chaque pompe est connu par deux capteurs. Suivant le statut de ces capteurs le contrôleur envoie des ordres d'ouverture et de fermeture des pompes et de la valve.

Cependant, il se peut (malheureusement) qu'un ou plusieurs des éléments de la turbine tombe en panne, par exemple qu'une pompe ne s'ouvre pas ou bien que le capteur de niveau d'eau retourne une valeur incorrecte. Le contrôleur doit d'une part détecter ces pannes, en référer en envoyant un message d'erreur, et attendre en retour des messages indiquant que le message d'erreur a été reçu, puis que la réparation a été effectuée.

Suivant le type de panne, et en attendant les réparations, le contrôleur se trouve dans des modes différents : mode initial, mode normal (tous les éléments fonctionnent correctement), mode dégradé (une pompe est en panne par exemple), mode de secours (il est nécessaire de calculer une valeur approchée du niveau d'eau de la turbine) et enfin le mode d'urgence qui signifie que le contrôleur ne peut plus assurer que le niveau d'eau ne devient pas critique et s'arrête immédiatement. Le système est représenté sur la figure 1, qui correspond à une image du simulateur graphique de la turbine fourni avec la spécification du cas test. Le code (C) généré par notre spécification SyncCharts a été couplé à ce simulateur. Il régule le niveau d'eau. L'utilisateur déclenche manuellement les pannes des pompes ou de leurs contrôleurs.

Nous allons modéliser une partie du système (celle liée au contrôle/commande) de la pompe. Au travers de cet exemple, nous montrerons quelques possibilités d'expression graphiques de comportements réactifs. Notons que, dans un souci de clarté graphique, nous ne détaillons que certains modules.

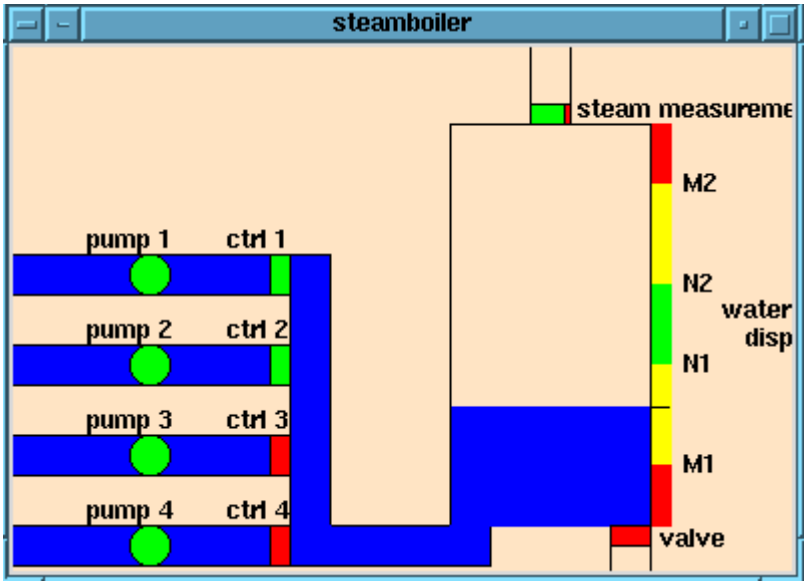

Fig. 1

4.2 Contrôle de la pompe

Le module de contrôle de pompe (représenté Fig. 2), que nous appelons « PumpController » est chargé, en l'absence de pannes, de relayer les demandes d'ouverture et de fermeture d'une pompe aux actionneurs. Le fonctionnement nominal est donc cyclique. Pour tenir compte de l'inertie des actionneurs on temporise les changements d'état par une attente de 5 secondes (signal *five_sec*). La constellation (nom donné à un ensemble d'étoiles modélisant une activité séquentielle) de gauche représente ce comportement.

Des capteurs permettent de détecter les anomalies de fonctionnement. On considère qu'il peut y avoir défaillance de la pompe ou du capteur de débit aval (ce capteur est malheureusement appelé contrôleur de pompe dans la spécification). Les indications des capteurs doivent être analysées par programme : une panne est détectée par une incohérence entre l'état attendu du processus physique et son état observé. La détection des pannes dépend donc du contexte (état courant du système). Nous avons, en conséquence, placé les détecteurs de défaillance dans les états eux-mêmes (raffinement de macro-état).

Le module *CP_Mntr* (Closed Pump Monitor) doit détecter

- une défaillance de pompe dans l'état pompe fermée (module *CP_P_fail* pour pump failure when closed),

- une défaillance du capteur de débit dans le même état (module *CP_C_fail* pour controller failure when closed).

Ces deux détections sont indépendantes et modélisées comme des activités parallèles. Le module *OP_Mntr* (Opened Pump Monitor) a une structure analogue et est actif lorsque la pompe est normalement ouverte.

Un détecteur signale une anomalie en émettant un sigal (e.g., *p-failure* émis par *CP_P_fail*). Ce signal est perçu par un sous-système chargé de suivre les réparations. Les deux constellations à droite dans le module *PumpController* sont chargées respectivement du suivi des réparations sur le capteur de débit (droite haut) et sur la pompe (droite bas). Nous avons détaillé la gestion de la panne du capteur de débit.

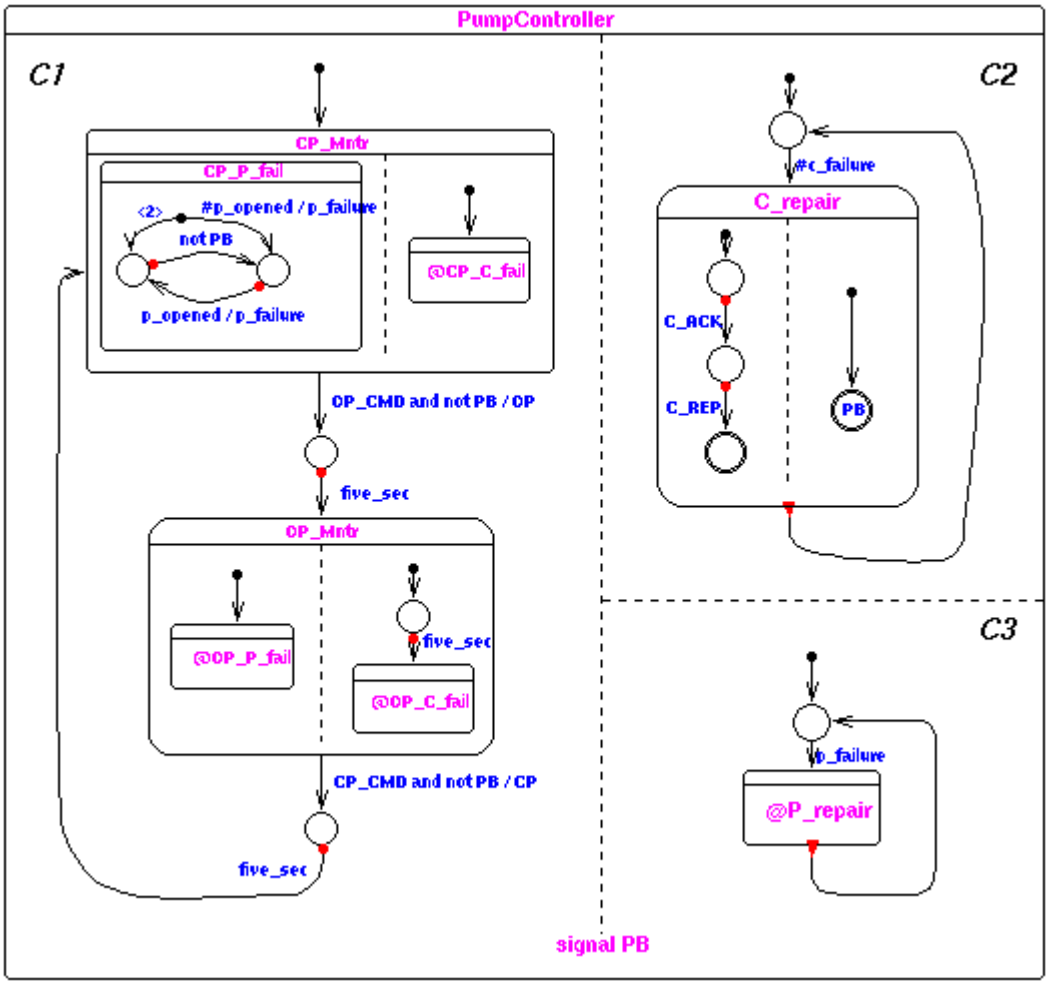

Fig. 2

Le module *C_repair* montre qu'on attend une séquence d'événements (réception d'un accusé de réception, puis fin de réparation). En parallèle avec cette attente, on émet en permanence le signal local *PB* (Problem) pour informer les autres composantes du contrôleur qu'une panne est en cours. Ce signal bloque en fait les changements d'état du cycle normal (préemptions faisant intervenir l'absence de ce signal). Dès la réparation effectuée, le module *C_repair* a ses deux constellations composantes dans un état final (étoile avec double cercle). Il termine donc instantanément (terminaison normale). Le sous-système est à nouveau prêt à prendre en compte une nouvelle demande de réparation. Le symbole # placé avant le signal déclencheur *C_failure* se lit « immédiat », il permet de rentrer immédiatement dans le module *C_repair* si l'anomalie est toujours présente.

4.3 Autres commentaires

Le module *CP_P_fail* montre qu'un module peut avoir plusieurs états initiaux. La sélection de l'état initial correct est déterministe (dans ce cas, si *p_opened* est présent on va dans l'étoile de droite. Sinon on va dans celle de gauche, ce choix étant moins prioritaire comme l'indique la priorité 2 associée à l'arc).

Signalons enfin une propriété non utilisée dans le contrôleur de pompe : les SyncCharts permettent d'exprimer la suspension d'un état ou macro-état lorsqu'un signal est présent. Ce mécanisme puissant permet de conserver un état courant (ainsi que les valeurs des variables) tout en figeant les évolutions momentanément. On évite ainsi les mécanismes d'entrée par histoire tels que l'utilisent les Statecharts.

5. Vérification de propriétés

L'utilisation de l'environnement SyncCharts pour écrire des contrôleurs de systèmes réactifs permet de profiter de l'environnement de vérification et de simulation du langage Estérel. Les outils de vérification `xeve` et `fc2symbmin` permettent de s'assurer de la satisfaction de propriétés de sûreté et de vivacité. Des contre-exemples peuvent être simulés à l'aide de l'outil graphique `xes` ou visualisés sous forme d'automates par `autograph`.

Les outils de vérification utilisent la méthode de vérification par observateur. Celle-ci consiste à associer au programme un observateur écrit en langage Estérel ou SyncCharts. Celui-ci peut, en émettant certains signaux, amener le programme à vérifier dans la configuration de départ souhaitée. Les outils permettent, pour toutes les exécutions possibles à partir de cette configuration, de vérifier que certains comportements du programme peuvent ou ne peuvent pas se produire, ou encore qu'ils se produiront inévitablement.

Ces propriétés nécessitent de construire une représentation de tous les comportements possibles du contrôleur. Cette représentation peut être soit implicite, dans ce cas elle est décrite par un ensemble de fonctions booléennes, soit l'automate est généré explicitement, mais le label des étiquettes est interprété symboliquement lors de son analyse, par exemple réduction, masquage de signaux ou/et composition.

Plusieurs propriétés, énoncées ci-dessous, ont été vérifiées sur une description SyncCharts de la turbine dans une version avec deux pompes :

- Le contrôleur est forcément dans un et un seul des modes normal, dégradé, de secours ou d'urgence à chaque instant.
- Le contrôleur ne peut pas être en mode normal, dégradé ou de secours avec la valve de vidange ouverte.
- Si le système passe en mode d'urgence, il s'arrête dans l'instant.
- Le contrôleur ne peut s'arrêter que dans les cas prévus et pas autrement.
- Il n'est pas possible d'envoyer une demande d'ouverture et une demande de fermeture à une pompe dans le même instant.
- En cas de détection de panne de capteur de niveau d'eau et en l'absence d'autre panne, le contrôleur passe du mode normal au mode de secours.

6. Références

[1] Jean-Raymond Abrial, Egon Borger, Hans Langmaack , *Formal methods for Industrial Applications, Specifying and Programming the Steam Boiler Control* LNCS 1165 , 1996.

[2] F. Boussinot, R de Simone, *The Esterel language*, proceedings of the IEEE, 79(9): 1268-1336, September 1991

[3] C. André, *Representation and Analysis of Reactive Behaviors: A Synchronous Approach*, Computational Engineering in System Application (CESA), IEEE-SMC, p19-29, Lille (F), July 1996.

[4] D. Harel, *Statecharts: A visual formalism for complex systems.* Science of computer programming, 8 :231-274, 1987.

Pour plus d'informations, le lecteur pourra aussi consulter :

- http://www.simulog.fr pour l'environnent SyncCharts/Estérel ;

- http://www.inria.fr/meije/verification/esterel/ pour les outils de vérification pour le langage Estérel .