INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Projet EP-ATR

Environnement de programmation d'applications temps réel

Rennes

THÈME 1C

Table des matières

1 Composition de l'équipe

Responsable scientifique

Paul Le Guernic [DR Inria]

Assistante de projet

Huguette Béchu [TR Inria]

Personnel Inria

Albert Benveniste [DR, projet Sigma 2]
Patricia Bournai [IR, Atelier, à mi-temps dans le projet]
Thierry Gautier [CR]
Hervé Marchand [CR]
Jean-Pierre Talpin [CR]

Personnel CNRS

Loïc Besnard [IR, Atelier]

Personnel Université de Rennes 1

Bernard Houssais [maître de conférences]
Michel Le Borgne [maître de conférences]
Sophie Pinchinat [maître de conférences, détachée en tant que CR Inria à parir du 1er octobre 1999]
Christophe Wolinski [maître de conférences]

Chercheurs doctorants

Fernando Jiménez Fraustro [bourse du gouvernement mexicain]
Mickaël Kerbœuf [bourse MENRT, à partir du 1er octobre 1999]
Jean-Christophe Le Lann [bourse Inria Industrie, jusqu'au 31 octobre 1999]
Pierre Le Maigat [bourse sc menrt]
Mirabelle Nebut [bourse MENRT]
David Nowak [bourse MENRT, jusqu'au 31 août 1999]
Yunming Wang [bourse CIES]

2 Présentation et objectifs généraux

Mots clés : EP-ATR, système enfoui, temps réel, conception synchrone.

Le projet EP-ATR conduit des travaux à la fois théoriques, méthodologiques, algorithmiques et d'expérimentation sur le thème de la conception de systèmes (ou d'applications) enfouis temps réel ; ces travaux sont basés sur une approche synchrone de la description de processus.

2.1 Contexte des études

Mots clés : système enfoui, temps réel, cycle en V, vérification, méthode formelle.

Les produits informatiques *enfouis,* le plus souvent avec des contraintes *temps réel* fortes, dans des systèmes industriels de grande envergure comme les centrales nucléaires, les systèmes de contrôle aérien, les télécommunications, ou dans des systèmes de taille plus modeste (avions, automobiles...), mais aussi de petite taille comme les processeurs ou contrôleurs divers utilisés dans des produits de grande diffusion, ont pour caractéristique commune de devoir fonctionner selon un mode de coopération permanente avec un environnement. Le système peut influencer sérieusement le comportement de cet environnement, non seulement par son activité mais aussi par son inactivité : ainsi le dysfonctionnement d'un système de commande de vol peut entraîner un accident d'avion ; celui d'un système de régulation thermique, la perte d'une entreprise avicole, par exemple. En fonction de la gravité des conséquences prévisibles d'un dysfonctionnement du composant informatique sur les plans économique, humain, ou social, le développement des applications considérées est l'objet de procédures plus ou moins lourdes visant à la réduction des risques d'erreurs ou de défaillances du système.

Traditionnellement le développement de ces applications s'appuie sur un *cycle en V* qui de l'amont vers l'aval, en partant d'un cahier des charges, passe par l'établissement de spécifications aussi complètes et consistantes que possible, sur lesquelles s'appuie la conception globale puis détaillée à partir de laquelle est produit le codage de l'application. La confiance dans le produit réalisé repose sur le respect de procédures codifiées incluant un chemin de *vérification,* inverse du chemin de conception, allant du test unitaire au test système. Les *méthodes formelles,* s'appuyant sur des modèles mathématiques correctement définis, trouvent aujourd'hui une place grandissante dans ces méthodologies et sont d'ores et déjà admises, voire recommandées, comme complément dans les méthodes traditionnelles, y compris dans des documents normatifs.

2.2 Objectif général du projet

Mots clés : temps réel, cycle en V, traitement temps réel du signal, télécommunication, génération de code enfoui, code distribué, architecture hétérogène, programmation synchrone.

C'est dans ce cadre que s'inscrivent les activités conduites par le projet EP-ATR : elles visent à proposer des modèles théoriques, des enrichissements du contexte méthodologique traditionnel (cycle en V) prenant en compte ces modèles, et enfin des logiciels (à l'état de prototype avancé) permettant de mettre en œuvre ces méthodes de conception nouvelles et de conduire des expérimentations des solutions proposées. Ayant débuté dans le domaine du traitement temps réel du signal en télécommunication, nos travaux ont trouvé de nouveaux contextes d'applications, en particulier par des collaborations suivies avec différents projets de l'Inria (robotique, image), avec EDF, avec nos partenaires du projet Esprit Sacres, dans le domaine des télécommunications, là encore en collaboration avec des projets de l'Inria, avec Alcatel et le Cnet, et dans le domaine de l'automobile, avec les industriels impliqués dans le projet AEE, plus là aussi différents projets de l'Inria et d'autres partenaires universitaires.

En même temps que se sont élargis les domaines d'applications, les problèmes que nous

considérons se sont étendus d'une part en amont vers la spécification de systèmes temps réel, d'autre part en aval vers la génération de code enfoui, éventuellement sous la forme de code distribué sur des architectures hétérogènes et au delà, vers la prise en compte de composants matériels, en synthèse et en modélisation.

Ces travaux, qui concernent ainsi aujourd'hui le développement des applications temps réel depuis leur spécification jusqu'à leur mise en œuvre matérielle, reposent sur une modélisation homogène des niveaux de description rencontrés au cours de la conception : cette approche, fondée sur un modèle mathématique de la *programmation synchrone,* permet de mettre formellement en relation différentes versions de l'application, réduisant en cela les risques liés à des ruptures dans le cycle de leur conception.

2.3 Conception synchrone

Mots clés : conception synchrone, Signal, système réactif, non déterminisme.

L'idée assez simple du modèle de *conception synchrone* est de considérer qu'un programme plongé dans un environnement (constitué de procédés, d'opérateurs, d'autres programmes), interagit significativement avec cet environnement au travers d'un ensemble fini de supports de communication à des instants formant un ensemble dénombrable partiellement ordonné. À chaque instant, plusieurs actions (messages reçus, émis, calculs. . .) peuvent être effectuées ; elles sont alors simultanées et possèdent un même indice temporel. L'écoulement du temps résulte des successions de ces communications, mais aussi de changements explicites décrits dans l'algorithme que le programme met en œuvre (par exemple pour une équation $y_t = f(x_{t-1})$, la sortie y sera simultanée à chaque occurrence suivante de l'entrée x). Selon les formalismes, les sorties calculées sont, sauf changement explicite dans le programme, soit simultanées aux entrées qui ont provoqué leur calcul comme dans les langages Eterel, Lustre ou Signal, soit produites à l'instant suivant comme dans Statemate ou VHDL par exemple.

Le langage Signal [7], conçu et mis en œuvre dans le projet, est de plus, comme Lustre, construit selon une approche flot de données faisant d'un programme un système d'équations. À la différence de ce qui se passe en Lustre, le système d'équations d'un programme Signal décrit une *relation* entre les signaux d'entrée et de sortie du système ; ceci permet d'utiliser Signal pour donner des spécifications partielles ou décrire des comportements non déterministes. Cette banalisation (néanmoins partielle) des entrées et des sorties permet en outre la spécification et la programmation de systèmes *réactifs* [HP85] mais aussi «*pro-actifs*».

2.4 Thèmes de recherche

Mots clés : Signal, conception synchrone, vérification, transformation de programme, communication synchrone/asynchrone, architecture hétérogène, composant matériel, format commun, programmation synchrone, DC+, Statecharts, automatismes industriels, système dynamique polynomial, synthèse de contrôleur, BDD.

[HP85] D. Harel, A. Pnueli, « On the development of reactive systems », *in: Logics and models of concurrent systems*, K. Apt (éditeur), NATO Advanced Study Institute on Logics and Models for Verification of Concurrent Systems, Springer Verlag, p. 477–498, New-York, 1985.

S'appuyant sur une expression en Signal des différentes versions d'un système en cours de conception, la méthodologie que nous développons consiste en l'application d'une série de transformations de descriptions respectant le schéma suivant :

- spécification, conception et vérification de l'application indépendamment de l'architecture cible, grâce à l'hypothèse de synchronisme et au non-déterminisme permettant de fournir un modèle du comportement de l'environnement ;
- affinement progressif vers l'implémentation (conception détaillée incluant le partitionnement) guidé par des simulations/vérifications à différents niveaux ; à cette étape, la cible est une architecture logicielle pouvant être vérifiée et évaluée ;
- implantation effective du schéma d'exécution obtenu sur des architectures éventuellement asynchrones et distribuées, en relâchant au besoin l'hypothèse de synchronisme (tout en restant dans le cadre du modèle synchrone), et en garantissant une implémentation correcte ;
- génération de code exécutable, de composants matériels ou de composants hybrides matériels/logiciels.

Nos activités de recherche concernent les différentes étapes de ce schéma de conception. Comme certaines de ces activités concernent plusieurs de ces étapes, nous distinguons ici, pour une meilleure lisibilité, les thèmes suivants : description d'applications, étude des propriétés des processus synchrones, méthodes et outils pour la conception d'architectures de mise en œuvre, développements et expérimentations.

2.4.1 Description d'applications

La description d'une application, tant au niveau de la spécification que de la conception, suppose l'existence d'un langage suffisamment expressif pour les classes d'applications visées ; la réduction des ruptures dans le cycle de conception est favorisée par le support logiciel des traitements dans une sémantique homogène. Les études portant sur l'augmentation du pouvoir d'expression du langage Signal ont pour but d'étendre le domaine d'application des techniques synchrones, soit par la définition de relations sémantiques entre les formalismes synchrones et d'autres formalismes, soit par l'adjonction de nouvelles primitives ou de nouveaux concepts dans le langage lui-même.

- Avec la société TNI (voir section 7.5) qui commercialise Sildex, un environnement de programmation fondé sur Signal, nous avons travaillé à la définition d'une *nouvelle version (V4)* dont l'une des principales extensions porte sur la modularité ; d'autres extensions ont également été définies et de nouvelles techniques de compilation sont implémentées (voir section 6.1) ;
- Sur le plan théorique, nous avons introduit la notion de *structure synchrone,* qui permet de modéliser à la fois les concepts de synchronicité et de causalité (voir section 6.2) ;
- Entreprises initialement dans le cadre d'un ancien projet avec les Laboratoires de Marcoussis, les études sur les liens entre le paradigme objet et le modèle synchrone trouvent un nouvel élan dans les travaux auxquels nous participons sur la conception de BDL ; BDL est un formalisme reposant sur le modèle synchrone pour la spécification de comportements d'objets (voir section 6.3).

Le *format commun* de la programmation synchrone (DC+) résulte de travaux menés depuis plusieurs années, tout d'abord par les équipes françaises dans lesquelles ont été conçus les langages synchrones Esterel, Lustre et Signal au sein du groupement C2A, puis dans le cadre d'un projet européen Eureka (Synchron), et enfin dans le cadre des projets européens LTR Syrf et R&D Sacres [Sac97]. La définition de ce format a été décidée en vue de favoriser le partage de logiciels issus de la communauté synchrone, mais aussi de permettre une large ouverture vers d'autres formalismes en amont, et d'autres outils, latéralement et en aval.

Le projet européen Sacres (voir section 7.2) a eu précisément pour objectif de développer un environnement de conception multi-formalisme synchrone à destination des applications critiques enfouies ; outre les études sur le format lui-même, nous y avons conduit des travaux sur la *traduction de Statemate* en DC+. L'intégration à travers DC+ des langages synchrones Esterel, Lustre et Signal se poursuit dans le cadre du projet Syrf (voir section 6.4). Une étude comparable concerne la modélisation et traduction des langages de programmation d'automatismes industriels de la norme IEC 1131 (voir section 6.4).

2.4.2 Étude des propriétés des processus synchrones

Un programme Signal décrit le comportement d'un système de transitions dont divers formalismes permettent d'étudier les propriétés.

Jusqu'à une période récente, nos travaux sur ce thème ont porté essentiellement sur une modélisation des comportements par des *systèmes dynamiques polynomiaux* sur les entiers modulo 3 (permettant le codage des booléens et de l'absence d'occurrence d'un signal à un instant donné). Un système de calcul symbolique (Sigali) a été développé dans ce cadre (voir section 5.2). C'est dans ce système que sont maintenant étudiées des techniques de *synthèse de contrôleurs,* techniques utilisables pour affiner des spécifications, pour aider à la conduite de postes de commandes, voire à la génération automatique de tests (voir section 6.6).

Les systèmes dynamiques (voir section 6.5) permettent d'analyser (partiellement) les trajectoires des processus dans un ensemble d'états résultant de la combinaison de variables du programme Signal. Cet ensemble lui-même est l'objet, dès la compilation, de calculs qui reposent sur une représentation par hiérarchie de BDD (ceci est appelé *calcul d'horloges* [1]). Ces calculs sont utilisés en particulier pour effectuer des *transformations de programmes* en vue de vérification, de distribution et de génération de code.

Une approche plus classique pour l'étude des propriétés des programmes consiste en la génération d'un automate fini qui peut alors être fourni à des outils de vérification automatique largement développés et utilisés dans la communauté. Nos travaux portent notamment sur la *génération d'automates* prenant en compte la hiérarchie des horloges, et sur des méthodes de réduction et de comparaison (voir section 6.5).

Des approches complémentaires pour la vérification de propriétés sont fournies par les techniques d'*interprétation abstraite,* permettant la prise en compte de signaux numériques, par exemple (voir sections 6.5 et 6.7), et d'autre part par la démonstration interactive de théorèmes (voir section 6.8).

[Sac97] SACRES CONSORTIUM, « The Declarative Code DC+, version 1.4 », novembre 1997, Esprit project EP 20897: Sacres, `http://www.irisa.fr/ep-atr/Publis/Abstract/sacres-dc.html`.

2.4.3 Méthodes et outils pour la conception d'architectures de mise en œuvre

La définition de l'architecture supportant l'application temps réel peut comporter différentes classes de composants comme des processeurs standards, des DSP, des Asics, etc., et c'est donc dans la perspective de mise en œuvre sur de telles cibles hétérogènes que nous inscrivons aussi bien nos études sur la conception détaillée et le codage que des activités relevant de la conception conjointe matériel/logiciel.

Le problème du partitionnement d'un programme flot de données synchrone est abordé en s'attachant aux propriétés structurelles du graphe de l'application ; supposant donnée une répartition du code sur une architecture cible (cette répartition peut résulter d'algorithmes d'optimisation ou être explicitée comme spécification non fonctionnelle, par exemple pour des raisons de traçabilité), nous nous proposons de produire, par des transformations locales à chaque composant de cette architecture, à la fois le code pour ces composants et les protocoles de communication à mettre en œuvre (voir section 6.9). Cette mise en œuvre peut être «Globalement Asynchrone Localement Synchrone» (Gals). Le cadre théorique de nos recherches permet de caractériser des mises en œuvre synchrones ou partiellement désynchronisées en terme de préservations de propriétés de flots, d'ordres partiels, de taille mémoire, par rapport au programme initial.

Pour la génération de composants matériels ou de code exécutable sur un processeur programmable, l'approche que nous adoptons est là encore celle de la transformation de programmes conduisant à une expression en Signal aussi proche que possible de la structure du code cible (C ou VHDL actuellement). Cette approche diminue les risques d'incorrection liés au passage à un autre langage.

Elle est accompagnée d'études sur la modélisation synchrone de composants matériels en vue du prototypage rapide de circuits spécifiques, qui participent à l'activité grandissante du projet sur le thème de la conception conjointe matériel/logiciel (voir section 6.11). Un domaine d'application que nous considérons plus particulièrement est celui de l'automobile, pour lequel il s'agit de décrire dans un même formalisme (AIL) une architecture véhicule depuis les prestations demandées jusqu'à l'architecture opérationnelle finale (voir section 6.12).

Dans l'approche synchrone, les durées d'exécution ne sont pas directement prises en compte. La prise en compte de ces durées, nécessaire à la vérification de la correction de la mise en œuvre en regard des contraintes temps réel, est effectuée en considérant une interprétation qui, à un programme synchrone et une architecture donnée, associe un programme synchrone sur nombres entiers ; l'image du programme initial modélise les durées d'exécution de la mise en œuvre. On peut alors simuler le programme image ou chercher à utiliser des techniques analytiques, comme par exemple celles qu'offre l'algèbre MaxPlus (voir section 6.13).

2.4.4 Développements et expérimentations

Les travaux théoriques conduits dans le projet aboutissent à des prototypes mettant en œuvre les algorithmes conçus au cours de ces études. Nous nous efforçons de conserver et diffuser ce savoir-faire du projet par une intégration dans un environnement de conception construit autour du langage Signal. Une section est consacrée à la présentation de ce logiciel (section 5.1).

Des expérimentations sont conduites pour valider les algorithmes en relation avec des industriels. Elles concernent également la diffusion des principes synchrones par des développements donnant à des formalismes divers une traduction en Signal. Enfin l'étude d'applications en coopération avec des industriels ou des projets académiques est une source importante des problèmes nouveaux que traite le projet EP-ATR.

2.5 Problèmes ouverts et perspectives

Mots clés : système hybride, génération de test.

Le développement d'un système enfoui temps réel, tel que décrit dans ces pages, prend des processus discrets pour modèles des procédés physiques de l'environnement et suppose donc l'étude algorithmique (par exemple de lois de commande) effectuée. Or la validation d'un système complet doit prendre en compte, soit par modèle, soit dans des maquettes matérielles (et souvent les deux), les fonctions continues dirigeant le comportement des procédés. Si des travaux d'interfaçage entre des langages synchrones et des outils dédiés au continu tels que Matlab et Simulink ont été entrepris, il reste beaucoup à faire pour obtenir une réelle *intégration des formalismes discrets et continus* dans des systèmes hybrides.

Un obstacle sérieux à la mise en œuvre correcte par construction des systèmes temps réel reste le lien entre temps physique et temps logique et entre différents grains de temps discret, en particulier pour les points suivants :

- Prise en compte des événements fournis par l'environnement le plus souvent asynchrone au programme synchrone : les études sur les modèles de communication *synchrone/ asynchrone* sont à compléter.
- Si nous savons définir correctement des procédures de raffinement efficaces portant sur l'espace (adjonction/suppression de variables), en ce qui concerne le temps, les études que nous avons entreprises tant par le biais des transformations affines [28], que dans les cadres du (re)timing monocadencé ou de l'utilisation de l'algèbre MaxPlus, restent encore de portée trop limitée pour satisfaire les besoins de transformation visant à éliminer les systèmes d'exploitation dans les systèmes critiques enfouis.

Pour aller vers la vérification complète des propriétés d'un système, il est nécessaire de prendre en compte des domaines plus riches que les booléens. Nous comptons entreprendre des études visant à compléter dans un premier temps le calcul d'horloges par des techniques provenant des résolutions de contraintes sur domaines finis ; pour aller plus loin, il deviendra nécessaire d'adapter des techniques de démonstration automatique. Nous envisageons de conduire ces études en collaboration avec des équipes spécialisées sur ces thèmes de recherche.

Même si les techniques formelles progressent, tant dans les esprits de leurs utilisateurs potentiels que dans les performances des algorithmes mis en œuvre, la simulation et le test resteront des activités nécessaires à la validation d'un système. C'est pourquoi nous envisageons de participer à des études sur la génération automatique de tests, principalement d'intégration, qui posent des problèmes non résolus.

En supposant un système correct vis-à-vis de spécifications de comportements nominaux, l'étude du comportement du système en présence de défaillances reste un sujet majeur pour les entreprises mettant en œuvre des systèmes critiques. Les études théoriques (qui bien sûr ne

peuvent concerner les seuls informaticiens) doivent être développées. Nous comptons aborder ce sujet sans prétendre à des solutions à court terme.

3 Fondements scientifiques

3.1 Spécification et programmation synchrone

Mots clés : conception synchrone, sémantique synchrone, programmation synchrone, Signal, transformation de programme.

Résumé : *Nous définissons la sémantique fonctionnelle d'un programme synchrone comme un ensemble de suites de valuations des variables de ce programme dans un domaine de valeurs complété par une représentation de l'absence d'occurrence d'une variable. Les opérateurs du langage Signal décrivent des relations sur de telles suites. Le compilateur de Signal est un outil formel capable de synthétiser la synchronisation globale d'un programme et l'ordonnancement de ses calculs. De nombreuses phases de compilation s'expriment par des transformations de programmes définies par des homomorphismes de programmes Signal.*

Les usages différents des termes *synchrone* et *asynchrone* selon les contextes dans lesquels ils apparaissent font qu'il nous semble nécessaire de préciser, autant que possible, ce qui constitue l'essence même du paradigme synchrone [BB91,BCLH93,Ber89,Hal93]. Les points suivants apparaissent comme caractéristiques de l'approche synchrone :

- Le comportement des programmes synchrones progresse via une suite infinie d'actions composées.
- Chaque action composée est constituée d'un nombre borné d'actions élémentaires, pour les langages Esterel, Lustre et Signal.
- À l'intérieur d'une action composée, les décisions peuvent être prises sur la base de l'*absence* de certains événements, comme il apparaît sur l'exemple des trois instructions

[BB91] A. Benveniste, G. Berry, « The Synchronous Approach to Reactive and Real-Time Systems », *Proceedings of the IEEE 79*, 9, Septembre 1991, p. 1270–1282.

[BCLH93] A. Benveniste, P. Caspi, P. Le Guernic, N. Halbwachs, « Data-Flow Synchronous Languages », *in : Lecture Notes in Computer Science 803, Proc. of the REX School/Symposium, Noordwijkerhout, Netherlands*, J. W. de Bakker, W. de Roever, G. Rozenberg (éditeurs), 803, Springer–Verlag, p. 1–45, Juin 1993.

[Ber89] G. Berry, « Real-Time Programming: special purpose or general purpose languages », *in : Information processing 89*, G. X. Ritter (éditeur), Elsevier Science Publisher B.V., 1989.

[Hal93] N. Halbwachs, *Synchronous programming of reactive systems*, Kluwer, 1993.

suivantes, issues respectivement de Esterel, Lustre, et Signal :

`present x else 'stat'`	l'action `stat` est exécutée en l'absence du signal `x`.
`y = current x`	en l'absence d'occurrence du signal `x`, `y` prend la valeur de la dernière occurrence de `x`.
`y := x default z`	en l'absence d'occurrence des signaux `x` et `z`, `y` est absent, sinon en l'absence d'occurrence du signal `x`, `y` prend la valeur de `z`.

– *Lorsqu'elle est définie,* la composition parallèle de deux programmes s'exprime toujours par la composition des couples d'actions composées qui leur sont respectivement associées, elle-même obtenue par la conjonction de leurs actions élémentaires respectives.

Pour ce qui concerne la spécification de programmes (ou de propriétés), la règle ci-dessus est clairement la bonne définition de la composition parallèle.

S'il s'agit également de programmation, la nécessité que cette définition soit compatible avec une sémantique opérationnelle complique largement la condition «lorsqu'elle est définie».

3.1.1 Sémantique synchrone

La sémantique fonctionnelle d'un programme Signal est décrite comme l'ensemble des suites admissibles de valuations des variables de ce programme dans un domaine de valeurs complété par la notation d'absence d'occurrence [5, 8].

Considérant :

– A, un ensemble de variables,

– D, un domaine de valeurs incluant les booléens,

– $\perp$, n'appartenant pas à D,

une *trace* T sur $A_1 \subset A$ est une fonction $T : \mathbb{N} \to A_1 \to (D \cup \{\perp\})$.

Pour tout $k \in \mathbb{N}$, un événement sur A_1 est une valuation $T(k)$: une trace est une suite d'événements. On appelle événement nul l'événement dans lequel chaque valeur est égale à $\perp$.

Pour toute trace T, il existe une trace F unique appelée *flot*, notée *flot*(T), dont la sous-suite des événements non nuls est égale à celle de T et initiale dans F. La projection $\pi_{A_2}(F)$ sur un sous-ensemble $A_2 \subset A_1$ d'un flot F, défini sur A_1, est le flot *flot*(T) pour T trace des restrictions des événements de F à A_2.

Un *processus* P sur A_1 est alors défini comme un ensemble de flots sur A_1. L'union de l'ensemble des processus sur $A_i \subset A$ est noté $\mathcal{P}_A$.

Étant donné P_1 et P_2 deux processus définis respectivement sur des ensembles de variables A_1 et A_2, leur composition, notée $P_1|P_2$, est l'ensemble des flots F définis sur $A_1 \cup A_2$ tels que $\pi_{A_1}(F) \in P_1$ et $\pi_{A_2}(F) \in P_2$. La composition de deux processus P_1 et P_2 est ainsi définie par l'ensemble de tous les flots respectant, en particulier sur les variables communes, l'ensemble des contraintes imposées respectivement par P_1 et P_2.

Soit $\mathbf{1}_{\mathcal{P}}$ le processus ayant comme seul élément la trace (unique) sur l'ensemble vide de variables. On montre alors que $(\mathcal{P}_A, \mathbf{1}_{\mathcal{P}}, |)$ est un monoïde commutatif (cette propriété rend

possibles les transformations de programmes mentionnées en 3.1.2) :

$$(P_1|P_2)|P_3 = P_1|(P_2|P_3)$$
$$P_1|P_2 = P_2|P_1$$
$$P|\mathbf{1}_{\mathcal{P}} = P$$

De plus, pour tout $A_1 \subset A$, les processus $\mathbf{0}_{\mathcal{P}}$ définis par l'ensemble vide de flots sur A_1 sont absorbants :

$$P|\mathbf{0}_{\mathcal{P}} = \mathbf{0}_{\mathcal{P}}$$

Enfin, l'opérateur de composition est idempotent (ceci autorise la réplication de processus) :

$$P|P = P$$

Par ailleurs, si P est un processus sur A_1 et Q un processus sur A_2 inclus dans A_1, on a :

$$P|Q = P$$

si et seulement si tout flot de la projection de P sur A_2 est un flot de Q (Q est moins contraint que P).

3.1.2 Langage Signal

Un programme Signal [7] spécifie un système temps réel au moyen d'un système d'équations dynamiques sur des *signaux*. Les systèmes d'équations peuvent être organisés de manière hiérarchique en sous-systèmes (ou *processus*). Un signal est une suite de valeurs à laquelle est associée une *horloge,* qui définit l'ensemble discret des instants auxquels ces valeurs sont présentes (différentes de $\perp$). Les horloges ne sont pas nécessairement reliées entre elles par des fréquences d'échantillonnage fixes : elles peuvent avoir des occurrences dépendant de données locales ou d'événements externes (comme des interruptions, par exemple).

Le langage Signal est construit sur un petit nombre de primitives, dont la sémantique est donnée en termes de processus tels que décrits ci-dessus. Les autres opérateurs de Signal sont définis en terme de ces primitives, et le langage complet fournit les constructions adéquates pour une programmation modulaire.

Pour un flot F, une variable X et un entier t on note, lorsqu'il n'y a pas de confusion possible, X_t la valeur $F(t)(X)$ portée par X en t dans le flot F. On note par le même symbole une variable ou une fonction dans les domaines syntaxique et sémantique. Dans le tableau ci-dessous, on omet les événements nuls (qui, rappelons-le, terminent les flots ayant un nombre fini d'événements non nuls).

Le noyau de Signal se compose des primitives suivantes :

– Fonctions ou relations étendues aux suites :

$$\texttt{Y := f(X1,...,Xn)} \ : \begin{cases} \forall k, \texttt{Y}_k = \perp \ \Rightarrow \ \texttt{X1}_k = ... = \texttt{Xn}_k = \perp \\ \forall k, \texttt{Y}_k \neq \perp \ \Rightarrow \ \texttt{Y}_k = \texttt{f}(\texttt{X1}_k,...,\texttt{Xn}_k) \end{cases}$$

– Retard (registre à décalage) :

$$\texttt{Y := X \$1 init v0} : \begin{cases} \forall k, \texttt{Y}_k = \perp \Rightarrow \texttt{X}_k = \perp \\ \texttt{Y}_0 \neq \perp \Rightarrow \texttt{Y}_0 = \texttt{v0} \\ \forall k > 0, \texttt{Y}_k \neq \perp \Rightarrow \texttt{Y}_k = \texttt{X}_{k-1} \end{cases}$$

– Extraction sur condition booléenne :

$$\texttt{Y := X when B} : \forall k, \begin{cases} \texttt{B}_k \neq \mathit{true} \Rightarrow \texttt{Y}_k = \perp \\ \texttt{B}_k = \mathit{true} \Rightarrow \texttt{Y}_k = \texttt{X}_k \end{cases}$$

– Mélange avec priorité :

$$\texttt{Y := U default V} : \forall k, \begin{cases} \texttt{U}_k \neq \perp \Rightarrow \texttt{Y}_k = \texttt{U}_k \\ \texttt{U}_k = \perp \Rightarrow \texttt{Y}_k = \texttt{V}_k \end{cases}$$

La composition de deux processus $P|Q$ se traduit directement en la composition des ensembles de flots associés à chacun d'eux.

La restriction de visibilité de `X`, P `/ X` est la projection de l'ensemble des flots associés à P sur l'ensemble des variables obtenu en enlevant `X` à celles de P.

Comme on peut le voir pour les primitives, chaque signal a sa propre référence temporelle (son «horloge», ou ensemble des instants où il est différent de $\perp$). Par exemple, les deux premières primitives sont monocadencées : elles imposent que tous les signaux impliqués aient la même horloge. En revanche, dans la troisième et la quatrième primitives, les différents signaux peuvent avoir des horloges différentes. L'horloge d'un programme Signal est alors la *borne supérieure* de toutes les horloges des différents signaux du programme (les instants du programme sont les instants de l'un au moins de ces signaux).

Le compilateur de Signal consiste principalement en un système formel capable de raisonner sur les horloges des signaux, la logique, et les graphes de dépendance. En particulier, le *calcul d'horloges* [1] et le calcul de dépendances fournissent une synthèse de la synchronisation globale du programme à partir de la spécification des synchronisations locales (qui sont données par les équations Signal), ainsi qu'une synthèse de l'ordonnancement global des calculs spécifiés. Des contradictions et des inconsistances peuvent être détectées au cours de ces calculs.

On peut toujours ramener un programme P comportant des variables locales à un programme, égal à P, de la forme Q`/A1/.../An` où Q est une composition de processus élémentaires sans restriction de visibilité (i.e., sans R`/A`). Un principe général de transformation de programmes que nous appliquons (dans un but de vérification, pour aller vers la mise en œuvre, pour calculer des abstractions de comportement) est alors de définir des homomorphismes $\mathcal{T}_i$ sur les programmes Signal, tels que Q est égal à la composition de ses transformés par $\mathcal{T}_i$. Grâce aux propriétés du monoïde commutatif, la transformation qui à Q associe cette composition est elle-même un homorphisme. On sépare ainsi un programme en différentes parties sur lesquelles seront alors appliqués des traitements spécifiques.

3.2 Vérification et synthèse

Mots clés : Signal, transformation de programme, système dynamique polynomial, vérification, synthèse de contrôleur, BDD.

Résumé : *Le principe de transformation des programmes Signal permet de décomposer un programme en une partie décrivant le contrôle booléen et une partie contenant les calculs. Le contrôle lui-même définit un système dynamique qui peut être étudié sous plusieurs aspects à des fins de vérification et de synthèse : étude de l'ensemble des états admissibles (partie statique), pour laquelle une forme canonique arborescente utilisant des* BDD *a été définie ; calcul dynamique s'appuyant sur la représentation équationnelle d'un automate.*

En appliquant le principe de transformation, on décompose un programme P en une partie $Q(P)$ contenant le contrôle booléen et une partie $C(P)$ contenant les calculs non booléens, telles que $P = Q(P)|C(P)$.

Toute propriété de sûreté, qui peut s'exprimer sous la forme d'un programme Signal R, satisfaite par $Q(P)$, ce qui s'exprime sous la forme $R|Q(P) = Q(P)$, est également satisfaite par P, puisqu'il résulte de $P = Q(P)|C(P)$ que $P = Q(P)|P$ (voir 3.1.1).

À une équation purement booléenne I correspond $Q(I) = I$; $C(I)$ est alors l'élément neutre du monoïde.
D'une équation monocadencée, est extraite par Q la partie synchronisation des signaux ; on obtient par exemple pour `x := y+z` l'expression :
`x ^= y ^= z | x := y+z`
(`x ^= y ^= z` spécifie l'égalité des horloges de `x`, `y` et `z`).
Une équation de la forme `x := y when b`, dans laquelle `x` est non booléen, est décomposée en : `x ^= (y when b) | x := y when b`.
Une équation de la forme `x := y default z`, dans laquelle `x` est non booléen, est décomposée en : `x ^= (y default z) | x := y default z`.

Cette interprétation permet donc d'extraire, de façon automatique, par $Q(P)$, l'aspect système à événements discrets, du système hybride spécifié par le programme. En raison de l'opérateur de retard qui introduit des indices temporels différents, le système est dynamique.

L'étude de ces systèmes dynamiques repose sur l'utilisation de techniques algébriques sur les corps de Galois. Elle vise à exprimer les propriétés des systèmes dynamiques et à donner une solution algorithmique pour leur vérification et pour la synthèse de systèmes satisfaisant certaines spécifications.

$Q(P)$ est défini sur trois valeurs : `{ vrai, faux, absent }`. La sémantique des opérateurs de Signal et l'approche flot de données équationnelle conduisent naturellement à un codage de $Q(P)$ en équations polynomiales sur le corps $\mathbb{Z}/3\mathbb{Z}$ (ou $\mathcal{F}_3$), `vrai`, `faux`, `absent` étant représentés respectivement par 1, -1, 0 ($+$ est l'addition modulo 3, $\times$ est la multiplication usuelle).

L'étude de la sémantique abstraite d'un programme Signal se ramène alors à l'étude des systèmes dynamiques de la forme :

$$\left\{ \begin{array}{lcl} X_{n+1} & = & P(X_n,Y_n) \\ Q(X_n,Y_n) & = & 0 \\ Q_0(X_0) & = & 0 \end{array} \right.$$

où X est un vecteur d'état dans $(\mathbb{Z}/3\mathbb{Z})^n$ et Y un vecteur d'événements (interprétations abstraites de signaux) qui font évoluer le système.

Un tel système dynamique n'est qu'une forme particulière de système de transitions à espace d'états finis. C'est donc un modèle de système à événements discrets sur lequel il est possible de vérifier des propriétés [2] ou bien de faire du contrôle.

L'étude d'un programme consiste alors en :

– l'étude de sa partie *statique,* c'est-à-dire l'ensemble de contraintes

$$Q(X_n,Y_n) \quad = \quad 0$$

– l'étude de sa partie *dynamique,* c'est-à-dire le système de transitions

$$\begin{array}{rcl} X_{n+1} & = & P(X_n,Y_n) \\ Q_0(X_0) & = & 0 \end{array}$$

et l'ensemble de ses états atteignables, etc.

Différentes techniques ont été développées pour ces deux problèmes. Les contraintes statiques sont essentielles pour la *compilation* des programmes Signal, et des techniques très efficaces ont été développées pour cela. La partie dynamique demande plus de calculs, et est utilisée principalement pour la vérification de propriétés ; une technique plus générale — mais en retour moins efficace — a été développée pour la prendre en compte.

Le calcul d'horloges statique est au cœur du compilateur Signal,; il en détermine largement ses performances. Ce calcul s'appuie sur l'ordre partiel des horloges, qui correspond à l'inclusion des ensembles d'instants (une horloge pouvant être plus fréquente qu'une autre).

La situation suivante doit être considérée : `H` est l'horloge d'un signal, par exemple un signal à valeurs réelles `X`, et `K` est l'ensemble des instants où le signal `X` dépasse un seuil : `K := when (X > X_MAX)`. Alors 1/ chaque instant de `K` est un instant de `H`, et 2/ pour calculer le statut de `K`, il faut d'abord connaître le statut de `H`. Il y a donc à la fois le fait que `K` est moins fréquent que `H` et qu'il existe une contrainte de causalité de `H` vers `K`. Ceci est dénoté par `H` $\rightarrow$ `K`. De tels *sous-échantillonnages* successifs organisent les horloges en plusieurs *arbres,* l'ensemble de ces arbres constituent la *forêt* d'horloges du programme considéré. Si un seul arbre est obtenu, la synchronisation du programme et son exécution s'en déduisent aisément.

La forêt associée à un programme donné n'est pas unique, la question de l'équivalence de forêts d'horloges se pose donc. Une *forme canonique* de forêt a été définie [1]. Un algorithme efficace pour trouver cette forme canonique a été développé. Il repose sur des manipulations préservant l'équivalence, prenant en compte l'ordre des variables résultant de la causalité, et combinées à des techniques BDD (*Binary Decision Diagrams* introduits par Bryant en 1986 [Bry86]).

Le calcul dynamique s'appuie sur la représentation équationnelle d'un automate : les automates, leurs états, événements et trajectoires sont manipulés au travers des *équations* qui les représentent. Calculer des trajectoires d'états ou d'événements, des états atteignables, des projections de trajectoires, des états de *deadlock,* etc., s'effectue alors sur les coefficients des équations polynomiales. De manière similaire, des techniques de *synthèse de contrôle* ont été développées pour un système dynamique donné pour différents types d'objectifs de contrôle

[Bry86] R. Bryant, « Graph-Based Algorithms for Boolean Function Manipulations », *IEEE Transaction on Computers C-45*, 8, Août 1986, p. 677–691.

proposés par Manna et Pnueli [MP92,MP95], mais aussi pour des objectifs de contrôle portant sur la qualité de service.

La manipulation d'équations dans $\mathcal{F}_3$ est tout à fait similaire à la manipulation d'équations booléennes. Une variante de la technique BDD, appelée TDD (Ternary Decision Diagrams), a été développée pour réaliser ces calculs. Des expériences ont montré que le système formel Sigali qui en résulte peut effectuer en un temps raisonnable des preuves (ou de la synthèse de contrôleurs) sur des automates comportant plusieurs millions d'états atteignables.

Synthèse d'automatismes discrets

Partant d'un modèle global du système, contrôler un système dynamique polynomial consiste à se donner un objectif de commande (propriétés des trajectoires) et à synthétiser un contrôleur répondant à cet objectif [9]. Dans notre approche, le contrôleur synthétisé est une équation

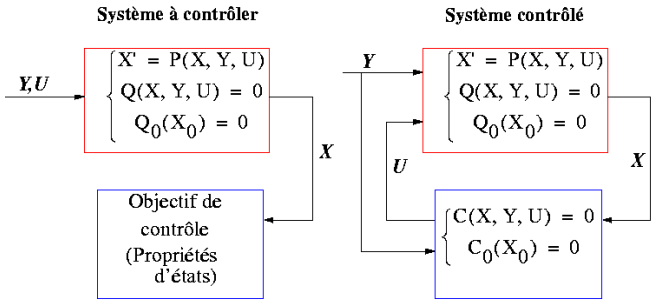

FIG. 1 – *Principe du contrôle*

polynomiale, C(X,Y,U), dépendant de l'état courant du système, X, des événements incontrôlables, Y, et des commandes, U (figure 1). Le rôle de cette équation, ajoutée au système initial, consiste à forcer la valeur de ceux-ci en restreignant, pour un état donné, le choix possible des valeurs des commandes admissibles. Les événements contrôlables peuvent alors être vus comme des événements de sortie du contrôleur (respectivement des événements d'entrée du système initial).

Différents types d'objectifs de contrôle peuvent être considérés : assurer l'invariance, l'atteignabilité ou l'attractivité d'un ensemble d'états, etc. Les objectifs de contrôle peuvent également traduire un critère qualitatif et non plus logique. Ils s'expriment alors comme des relations d'ordre ou comme un critère de minimisation sur une trajectoire bornée du système.

Les différentes mises en œuvre réalisées sont surtout axées sur l'intégration des différents algorithmes induits par les objectifs de contrôle dans le système de calcul formel Sigali, mais sont également axés sur l'intégration de Sigali dans l'environnement Signal de manière à faciliter les preuves de programmes et la synthèse d'objectifs de commande.

[MP92] Z. MANNA, A. PNUELI, *The Temporal Logic of Reactive and Concurrent Systems: Specification*, Springer-Verlag, 1992.

[MP95] Z. MANNA, A. PNUELI, *The Temporal Logic of Reactive and Concurrent Systems: Safety*, Springer-Verlag, 1995.

4 Domaines d'applications

4.1 Panorama

Mots clés : conception synchrone, télécommunication, traitement temps réel du signal, énergie, transport, avionique, automobile.

Résumé : *Les recherches sur les méthodes de développement d'applications ne sauraient se concevoir sans une confrontation avec des applications, pour identifier en amont les problèmes rencontrés par les concepteurs (utilisateurs potentiels de nos techniques) et pour valider en aval les solutions proposées. C'est ainsi que le projet* EP-ATR *s'est impliqué très tôt dans des travaux liés aux télécommunications. En outre, il est depuis plusieurs années engagé dans une coopération avec* EDF *sur l'utilisation des techniques synchrones dans le domaine de l'énergie. D'autres domaines ont été abordés, en particulier une application en traitement radar infrarouge a été traitée. Dans le cadre du projet Sacres, les applications considérées relèvent du domaine de l'avionique. Enfin, le domaine des applications automobiles est abordé à travers le projet* AEE *(voir section 7.4).*

4.2 Télécommunications

Participants : Albert Benveniste, Paul Le Guernic, Jean-Pierre Talpin, Yunming Wang.

Mots clés : Signal, télécommunication, traitement temps réel du signal, communication synchrone/asynchrone, conception objet.

Résumé : *Nos activités dans le domaine des télécommunications sont issues d'une longue collaboration avec le Cnet, d'où provient le développement initial du langage Signal. Elles se sont poursuivies dans le cadre du projet Cairn autour des langages Signal et* ALP*, ainsi que dans de nouvelles collaborations dans lesquelles l'utilisation mixte des modèles synchrone et asynchrone est étudiée.*

L'industrie des télécommunications est, de plus en plus, soumise à de fortes contraintes qui demandent un effort important visant à maximiser la généricité des solutions proposées et à raccourcir les délais de mise sur le marché des produits. La diversité que l'on rencontre dans les applications développées nécessite la mise en œuvre de techniques variées pour répondre aux problèmes rencontrés. Les techniques synchrones peuvent fournir des solutions partielles qu'il convient d'intégrer dans des méthodes de conception plus globales.

Issu d'une longue collaboration avec le Cnet, le langage Signal a d'abord été développé dans le cadre d'applications en traitement du signal. Le projet Cairn («Codesign d'Applications Irrégulières et Régulières par Niveaux») nous a permis, en collaboration avec le projet Api de l'Irisa, d'étendre les thèmes abordés à la conception de composants supportant des algorithmes qui comportent du calcul numérique intensif (image ou signal) et du contrôle complexe.

La taille et la complexité des applications mises en œuvre, la nécessité d'obtenir des spécifications et des programmes génériques, destinés à des configurations diverses fonctionnant

dans des contextes hétérogènes, conduisent à mettre en œuvre des outils de conception fondés sur l'approche objet. Dans le cadre de cette approche, la complexité des interactions et les contraintes temps réel sont traitées à l'aide de descriptions de comportements faisant appel à des modèles d'automates. L'étude de l'utilisation de l'approche synchrone dans ce contexte a d'abord été entreprise en collaboration avec les Laboratoires de Marcoussis par la définition d'un modèle d'interaction entre des objets (décrits dans le langage Spoke) et des processus (décrits dans le langage Signal). Elle se poursuit maintenant avec Alcatel, en collaboration avec les projets Pampa, ADP, Compose et Meije, dans le cadre de l'action Reutel-2000 visant notamment à maîtriser la conception objet d'applications mises en œuvre selon des techniques mêlant les modèles synchrone et asynchrone.

4.3 Énergie

Participants : Fernando Jiménez Fraustro, Hervé Marchand, Sophie Pinchinat.

Mots clés : énergie, méthode formelle, programmation synchrone, automatismes industriels.

Résumé : *Nos activités dans le domaine de l'énergie se situent depuis plusieurs années dans le cadre de coopérations avec* EDF *qui se prolongent à travers le projet Syrf (voir section 7.3). Elles concernent notamment la vérification et la synthèse de contrôleurs, mais aussi la simulation de systèmes hybrides.*

Dans le domaine de la production et de la distribution d'énergie, en particulier électrique, on est en présence de systèmes dont :

- La sécurité est un caractère essentiel ; elle concerne divers aspects des systèmes de contrôle :
  - le service fourni peut avoir des aspects critiques dans la nature de ceux à qui il est destiné (par exemple les hôpitaux),
  - le matériel lui-même est soumis à des conditions de fonctionnement dont le dérèglement peut entraîner des dommages fatals,
  - enfin les techniques utilisées peuvent comporter un risque pour l'environnement dans lequel s'effectue l'activité.

  Dans ces conditions, les systèmes qui contrôlent la production et la distribution d'énergie posent des problèmes qui sont du domaine d'application de nos techniques d'analyse et de vérification de comportement.
- La complexité est grande :
  - que ce soit dans le cas de réseaux de distribution, constitués de contrôleurs de transformateurs interconnectés,
  - ou bien de contrôleurs de centrale électrique, où les capteurs à prendre en compte et les actionneurs auxquels fournir une commande se comptent par centaines, voire milliers, et où les architectures sur lesquelles le contrôle s'exécute comprennent des réseaux d'automates programmables en parallèle.

La conception et l'analyse de ces systèmes requièrent le support d'outils automatisés pour la construction de modèles et le calcul de mises en œuvre correctes vis-à-vis de la spécification. Elles doivent exploiter la particularité d'architectures à base d'automates programmables, comme il en est construit par Siemens, avec qui nous coopérons dans le projet Esprit Sacres.

– L'héritage des spécifications de contrôleurs obéit à une culture particulière : les langages de spécification des contrôleurs ont été utilisés pour la conception de systèmes de taille importante, et récrire ces systèmes dans un langage nouveau n'est guère envisageable. Il s'agit de langages à base d'automates communicants, ou d'autres de type *blocs-diagrammes* et circuits d'opérateurs. Dans un but de meilleure acceptation des méthodes formelles par les utilisateurs, et de réutilisation du fonds des développements antérieurs, il est intéressant de travailler à l'intégration de ces langages aux techniques que nous proposons, sous la forme de leur encodage, et d'une traduction automatique dans un format synchrone.

5 Logiciels

5.1 Environnement de programmation Signal

Mots clés : Signal, programmation synchrone, format commun, DC+, transformation de programme, Sildex.

Contact : P. Bournai.

Résumé : *Le développement d'un environnement de programmation Signal, construit à l'Irisa selon des techniques de conception modulaires, répond à trois objectifs :*

a) il nous permet d'étudier des extensions sémantiques ou algorithmiques du modèle synchrone ;

b) il nous permet de mieux comprendre les applications et de dégager ainsi des problématiques nouvelles ;

c) il est diffusé à des fins d'expérimentation et d'enseignement dans des laboratoires pour lesquels la version commerciale Sildex ne convient pas.

L'environnement de programmation Signal se compose d'un compilateur et d'un éditeur graphique orienté blocs-diagrammes. *L'éditeur de Signal permet à l'utilisateur de construire ses programmes sous forme à la fois textuelle et graphique. L'existence de cet éditeur est un vecteur majeur pour la diffusion.*

Le compilateur et l'éditeur graphique sont écrits en C Ansi et sont disponibles sous les systèmes SunOS 4.1 et Solaris 2.5 et Linux. La même version du compilateur est également disponible sous Windows 95 et Windows NT. *Une nouvelle version de l'éditeur graphique écrite en C Ansi et Java est en cours de développement. Elle permettra d'avoir une version commune de l'environnement graphique de Signal dans les mondes Windows et Unix.*

Environnement de compilation

Un programme Signal est représenté de façon interne par un GHDC qui constitue donc la structure principale de l'environnement. On peut distinguer :

- un ensemble de *traitements frontaux* qui produisent un graphe hiérarchisé à partir d'un source Signal (ou DC),
- un ensemble de *transformations* qui restituent un graphe hiérarchisé ; ceci constitue le cœur du compilateur,
- un ensemble de *post-traitements* qui produisent les sources d'autres outils.

Traitements frontaux. L'ensemble de traitements pour la production du GHDC est constitué :

- de l'analyse syntaxique et contextuelle, qui fournit la représentation interne d'un programme source, Signal ou DC+, sous forme d'un arbre de syntaxe abstraite ;
- de la production de graphe, qui associe à tout programme un graphe caractérisé par un système d'équations d'horloges. À ce stade, aucune vérification sur le système d'équations ni sur le graphe (cycles...) n'est effectuée.

Transformations du graphe. L'ensemble de traitements qui transforment le graphe hiérarchisé est constitué :

- de la compilation, dont le rôle principal est de trianguliariser le système d'équations d'horloges et de détecter la présence de cycles de dépendances de données. Cette transformation permet la vérification partielle de la correction du programme vis-à-vis de ses synchronisations. La synthèse partielle d'expressions explicites du contrôle se traduit en une forêt d'arbres d'horloges, dont les racines sont éventuellement arguments de contraintes non résolues, et les nœuds internes des expressions explicites. Pour ce *calcul d'horloges,* nous avons développé une structure hiérarchique de BDD qui s'avère très performante ; nous utilisons actuellement le package BDD de Berkeley.
- des opérations de partitionnement de graphe, qui produisent un graphe constitué de nœuds représentant eux-mêmes des graphes. On peut citer :
  - la séparation du contrôle et des calculs, qui consiste à séparer la partie contrôle de l'application de la partie calcul.
  - le calcul de lignées sur entrées, qui consiste à partitionner un graphe selon le critère qualitatif suivant : deux nœuds sont éléments de la même lignée sur entrée si et seulement s'ils sont précédés du même sous-ensemble d'entrées. Ce partitionnement est à la base d'un nouveau schéma de génération de code séquentiel : une lignée peut être exécutée de manière atomique dès que ses entrées sont disponibles.
  - la répartition de programmes, qui se base sur l'utilisation de *pragmas* pour l'affectation des nœuds à des unités de calcul.
- des calculs systèmes suivants :
  - la synthèse d'interface, dont le but est l'extraction d'éléments de la représentation interne en vue de la compilation séparée de programmes Signal. Cette opération

consiste en le calcul de la fermeture transitive du graphe réduite aux entrées/sorties du processus compilé.
- le *retiming*, qui consiste en la réécriture de toute fonction synchrone construite sur les expressions de retard afin d'une part, de faire apparaître des variables d'état booléennes et d'autre part, de réduire le nombre des variables d'état.

Post-traitements. L'ensemble de post-traitements est constitué :
- de la génération de code séquentiel, qui passe par un tri topologique du graphe et qui produit du code C ;
- de la restitution de source, qui fournit à l'utilisateur le GHDC sous la forme d'un nouveau programme Signal faisant apparaître la hiérarchie obtenue et les synchronisations calculées. La restitution du source peut également être effectuée en partant d'une représentation sous forme d'arbre de syntaxe abstraite.
- de l'interfaçage avec des systèmes de preuves ; actuellement la connexion avec l'outil Sigali est réalisée dans le but d'étudier les propriétés dynamiques des programmes (décompilateur $\mathbb{Z}/3\mathbb{Z}$).
- les calculs d'architectures ; actuellement la connexion avec l'outil Syndex est réalisée. L'outil Syndex permet d'effectuer une implantation optimisée sous contraintes temps réel sur une architecture multi-processeur.

Diffusion du logiciel

La version commerciale de Signal est vendue par TNI sous la forme de l'environnement Sildex. La version Inria de Signal peut être obtenue dans le cadre d'une convention de mise à disposition gratuite signée pour un an renouvelable.

Signal est actuellement mis à disposition dans des écoles ou universités (Ubo, IUP de Lorient, université de Nantes Irin, Supelec, Oil & Gas University of Ploiesti — Pologne, University of Victoria — Canada, University of Reading — Grande-Bretagne, University of Michigan — USA, Mecaprom — Mexique), et chez certains industriels pour des études ou évaluations particulières (EDF).

Des mises à disposition communes Signal–Syndex peuvent également être effectuées (Y. Sorel, projet Sosso à Rocquencourt).

5.2 Sigali

Mots clés : Signal, DC+, Sigali, système dynamique polynomial, vérification, synthèse de contrôleur.

Contact : M. Le Borgne.

Résumé : *Sigali est un système de calcul formel permettant la vérification de propriétés de programmes Signal ou* DC+.

Sigali est un système de calcul formel interactif spécialisé dans les calculs algébriques sur l'anneau $\mathbb{Z}/3\mathbb{Z}[X]$. Il est destiné à la vérification des propriétés statiques et dynamiques de

programmes Signal ou DC+ [1, 2] et plus généralement de tout système dynamique polynomial dans $\mathbb{Z}/3\mathbb{Z}[X]$. Il permet également la synthèse de contrôleurs de systèmes à événements discrets. L'adjonction de primitives de création et de manipulation de fonctions à valeurs entières autorise les calculs de commande optimale.

Sigali est un logiciel déposé à l'APP sous le numéro IDDN.FR.001.370006.S.P.1999.000.10600. Comme pour l'environnement Signal, Sigali peut être obtenu après signature d'une convention de mise à disposition.

6 Résultats nouveaux

Nos thèmes de recherche introduits en 2.4 ont donné lieu aux résultats décrits dans les sections suivantes :

- les sections 6.1 à 6.4 relèvent de la «Description d'applications» (2.4.1) ;
- les sections 6.5 à 6.8 concernent l'«Étude des propriétés des processus synchrones» (2.4.2) ;
- les sections 6.9 à 6.13 portent sur des «Méthodes et outils pour la conception d'architectures de mise en œuvre» (2.4.3).

6.1 Évolutions de Signal et de son environnement

Mots clés : programmation synchrone, Signal, format commun, DC+, transformation de programme.

Participants : Loïc Besnard, Patricia Bournai, Thierry Gautier, Paul Le Guernic

.

Résumé : *La version actuellement diffusée de Signal, Signal V4, a été définie en coopération avec la société* TNI *(François Dupont), qui développe et commercialise l'environnement Sildex issu des travaux sur Signal. Elle est progressivement enrichie dans le sens d'une meilleure expressivité, et son environnement est amélioré de manière à en assurer une diffusion plus large.*

Évolutions du langage

En tant que langage à flots de données, Signal respecte le principe traditionnel de l'«assignation unique» : tout signal qui n'est pas une entrée du programme comporte une et une seule équation de définition. Si l'on souhaite cependant utiliser Signal pour décrire des automates à la Statecharts par exemple, ou des «automates de mode», on devra souvent définir différemment un même signal selon l'état ou le mode considéré. Afin d'alléger l'expression, il est alors pratique de disposer d'une notion de définition partielle d'un signal. Nous avons donc assoupli le principe précédent en introduisant la notation `Y ::= E` pour représenter cette notion en Signal : une telle expression est équivalente à une équation `Y := E default Y`. Ces expressions ne suffisent pas à fixer l'horloge de `Y`, qui doit être déterminée par ailleurs.

La sémantique de traces de Signal définit une sémantique dénotationnelle en termes de relations, ne nécessitant pas en général une orientation des équations. Mais ceci est perdu en

partie dans la syntaxe du langage puisque l'emploi du `:=` définit explicitement les entrées et les sorties. Nous avons donc ajouté un nouveau symbole, noté `:==:`, permettant d'exprimer des contraintes d'égalité sur les suites, sans imposer a priori un ordre d'évaluation. Charge est alors au compilateur de calculer un tel ordre et d'expliciter les calculs.

Enfin, nous avons introduit deux autres notions permettant de simplifier l'écriture de certains programmes Signal :

- la notion de *nodes* à la Lustre, permettant de déclarer des processus *endochrones,* sans accélération interne, intermédiaires donc entre les fonctions (sans mémoire) et les processus généraux ;
- les notions de types processus et de processus passés en paramètres (limités ici aux paramètres statiques).

Environnement Signal–DC+

L'environnement de Signal V4 intègre désormais le format DC+ et permet donc d'appliquer les mêmes fonctionnalités aux programmes représentés en DC+ (ou en l'un de ses sous-formats, par exemple, DC) et aux programmes Signal.

Les sous-formats de DC+ (bDC+, STS et DC) sont décrits dans la section 7.2. Les transformations inter-formats, DC+ → bDC+ et bDC+ → STS avaient été définies les années précédentes. Elles ont été complétées cette année par la mise en œuvre de la transformation STS → DC, ce qui permet ainsi d'appliquer aux programmes Signal les divers outils disponibles autour de DC.

La transformation STS → DC consiste en :

- la mémorisation à l'horloge *tick* de tous les signaux retardés (horloge d'état) ;
- la suppression des opérateurs polychrones lors de la production du texte DC ; tout nœud de définition est caractérisé par une condition d'activation qui est l'horloge booléenne STS ;
- les entrées du programme sont supprimées et deviennent des signaux locaux ; un signal d'entrée est remplacé par un appel de fonction conditionnée par le booléen qui fixe ses instants de lecture.
- les sorties du programme sont également supprimées et deviennent des signaux locaux ; un appel de procédure conditionnée par le booléen qui fixe les instants d'écriture est ajouté au programme.

À l'ensemble des sous-formats de DC+ listés ci-dessus, nous avons ajouté un niveau supplémentaire appelé sbDC+ (pour «sequentialized boolean DC+») et avons mis en œuvre la transformation bDC+ → sbDC+. Ce niveau est le format d'entrée effectif des générateurs de code. Il est produit pour les programmes sans cycles et sans contraintes. Un code sbDC+ est une liste de nœuds ordonnés selon les dépendances implicites et explicites du programme Signal–DC+. Ceci permet d'écrire de nouveaux générateurs de code sans avoir à parcourir le graphe du programme.

Afin d'améliorer le calcul d'horloges et la génération de code, nous avons cette année enrichi l'environnement des nouvelles fonctionnalités suivantes :

- la suppression des renommages (équations de définition triviales) ;
- l'unification des signaux définis par la même expression ;

– la substitution des signaux référencés au plus N fois dans le programme (N étant un paramètre de compilation), par leur expression de définition.

L'application de ces réécritures a permis de réduire de 35% la taille des fichiers C engendrés.

6.2 Structures synchrones

Participants : Paul Le Guernic, David Nowak, Jean-Pierre Talpin.

Résumé : *Nous avons formalisé un modèle de la programmation synchrone en introduisant la notion de structures synchrones, construite uniquement autour des concepts de synchronicité (modélisé par une relation d'équivalence), et de causalité (modélisé par un ordre partiel).*

Nous avons formalisé l'essence de l'hypothèse synchrone dans le modèle des structures synchrones construit uniquement autour des concepts de synchronicité, modélisé par une relation d'équivalence, et de causalité, modélisé par un ordre partiel [12, 27]. Nous avons défini un signal comme étant un ensemble d'événements non synchrones entre eux. Cette définition autorise des signaux dont les événements ne sont que partiellement ordonnés et que nous disons imaginaires, en contraste avec des signaux réels qui sont forcément totalement ordonnés. Nous avons alors défini les horloges, réelles ou imaginaires, comme les classes d'équivalence des signaux synchrones, et les instants comme les classes d'équivalence des événements synchrones. Puis nous avons montré que l'ensemble des horloges est isomorphe à l'ensemble des ensembles d'instants et a donc une structure de treillis booléen. Cette structure garantit que deux horloges ont toujours une borne supérieure. Nous avons ensuite relié les signaux à une sémantique plus classique de traces où celles-ci sont des fonctions totales définies sur l'ensemble des instants et de codomaine l'ensemble des événements du signal considéré plus une valeur spéciale $\perp$ dénotant l'absence d'un signal à un instant donné. Nous avons montré que les signaux munis de morphismes définissent une catégorie de préordre cartésienne fermée que l'on peut relier à la catégorie des structures d'événements premières. Une logique temporelle arborescente similaire à ISTL («Interleaving Set Temporal Logic») a été définie sur les structures synchrones. L'ordre partiel des structures synchrones modélisant uniquement la causalité temporelle a été étendu par un ordre partiel modélisant la dépendance des données, ce qui a donné lieu à la définition des structures synchrones dépendantes. Cette relation de dépendance est un ordre partiel inclus dans le préordre de la structure synchrone. Il a la propriété que toutes les dépendances viennent du passé ou du présent, i.e., la valeur d'un événement ne peut pas dépendre de la valeur d'un événement de son futur. L'hypothèse synchrone est une bonne abstraction pour la conception et la vérification des systèmes réactifs. Pour la réalisation effective de ces systèmes réactifs, il faut découper les instants logiques de la spécification en sous-instants tout en respectant les dépendances temporelles et de données. Une telle transformation s'appelle un *raffinement temporel*. Ce concept décrit la recherche par le compilateur Signal d'un ordre d'exécution des événements synchrones respectant à la fois les dépendances temporelles et de données. Les dépendances de signaux conditionnées par une horloge ont été définies et des théorèmes permettant l'abstraction des dépendances conditionnées d'un processus par rapport à ses signaux d'entrée et de sortie ont été prouvés. Finalement, les structures synchrones valuées

ont été définies par l'association d'une valeur à chaque événement et ont permis de donner une sémantique dénotationnelle complète de Signal pouvant modéliser le non-déterminisme ainsi que les dépendances conditionnées induites par les programmes Signal.

6.3 BDL, un formalisme synchrone déclaratif pour UML

Participants : Albert Benveniste, Paul Le Guernic, Jean-Pierre Talpin, Yunming Wang.

Mots clés : systèmes de transitions synchrones, UML.

Résumé : *En collaboration avec le projet Pampa, et dans le cadre du contrat Reutel avec Alcatel, nous avons élaboré un formalisme synchrone de préordres étiquetés, appelé* BDL *(Behavioral Description Language).* BDL *est destiné à permettre la manipulation abstraite d'objets, de composants, et d'architectures, avec une sémantique double synchrone/asynchrone, s'appuyant sur les résultats décrits dans la section 6.9. Nous étudions une syntaxe graphique pour* BDL*, par adaptation du formalisme des Statecharts. Notre objectif est de pouvoir transcoder en* BDL *à la fois des Statecharts* UML*, et des diagrammes de séquence* UML.

Nous avons introduit le modèle des Spots, ou systèmes de transitions synchrones préordonnés, permettant de modéliser indifféremment les systèmes réactifs synchrones et les systèmes distribués asynchrones [34]. Outre des spécifications de contrôle selon la philosophie synchrone, ce formalisme autorise la description de causalités et de séquencements au moyen de relations de pré-ordre. Par rapport aux langages synchrones existants, ce modèle offre une nouvelle loi de composition : le choix non déterministe. Avec la composition parallèle synchrone et le choix non déterministe, on dispose des opérations adéquates pour faire de l'héritage sur le plan comportemental (par addition de contraintes, ou par enrichissement de comportements).

BDL est une syntaxe concrète qui met en œuvre ce modèle. BDL bénéficie complètement de l'ensemble des résultats sur la désynchronisation (endo- et isochronie). BDL est destiné à être connecté à la plateforme Umlaut dévloppée dans le projet Pampa, ainsi qu'à l'environnement Signal pour bénéficier des outils qui y sont associés.

6.4 Format commun DC+ et multi-formalisme

Participants : Loïc Besnard, Patricia Bournai, Thierry Gautier, Fernando Jiménez Fraustro, Paul Le Guernic.

Mots clés : format commun, DC+, programmation synchrone, Signal, Statecharts, automatismes industriels, norme IEC 1131, Grafcet.

Résumé : *Le format* DC+*, issu des travaux des projets européens Synchron et Sacres, permet de représenter, par delà un langage particulier, la paradigme «flots de données synchronisés». Il constitue aussi un format concret, servant de vecteur commun de représentation, pour des programmes ou des propriétés sur lesquels on souhaite appliquer des transformations définies dans le cadre du modèle synchrone.*

Une approche multi-formalisme est également considérée dans le contexte des langages de la norme IEC 1131 *concernant les automatismes industriels (comprenant le Grafcet ainsi que différents langages graphiques et textuels).*

Multi-formalisme Signal–Lustre–Esterel via DC+

Dans le cadre du projet Syrf (voir section 7.3), nous avons expérimenté sur une étude de cas une approche multi-formalisme utilisant les langages synchrones Signal, Lustre et Esterel. Il s'agit d'une partie du contrôle d'une chambre de régulation climatique (température et ventilation) utilisée dans l'avionique [17]. Des parties de l'application ont donc été programmées en Lustre et Esterel et le superviseur a été écrit en Signal en faisant appel à des processus externes pour les parties Lustre et Esterel. Celles-ci ont été compilées à part via les compilateurs Lustre et Esterel, et les représentations de ces parties en format commun DC ont ainsi été produites. Chacun de ces codes DC a été compilé séparément dans le compilateur Signal–DC+ de manière à en obtenir l'*abstraction horloges et dépendances.* L'inclusion de ces abstractions dans le programme Signal du superviseur permet alors de compiler celui-ci en s'appuyant sur les résultats de la compilation séparée de certains de ses composants.

Cette même expérimentation a été poursuivie en appliquant au système multi-formalisme ainsi défini la méthodologie de distribution de programmes offerte dans l'environnement Signal–DC+, soit, après la définition du graphe logiciel représenté ici par le programme Signal du superviseur :

- définition par l'utilisateur de son architecture matérielle, les boîtes correspondant alors à des processeurs ;
- *mapping* du graphe logiciel sur le graphe matériel, avec réplication de code autorisée (une boîte représentant du logiciel peut être dupliquée dans plusieurs boîtes représentant des processeurs) ;
- compilation globale du programme représentant la composition des processeurs ; chaque processeur est attribué d'un pragma particulier qui sera exploité par le compilateur pour le partionnement ;
- remplacement automatique (au niveau graphique) des processeurs par le résultat de leur compilation ; le résultat de la compilation est un programme Signal dont l'interface est modifiée afin de rendre endochrone chaque programme sur chaque processeur ;
- ajout des processus de communication avec leurs connexions ; chaque processus de communication est composé d'un processus de réception des données, d'un processus de description de la ligne de communication, et d'un processus d'émission des données ;
- compilation globale en vue d'une simulation de l'ensemble de l'architecture ou d'une évaluation de son coût ;
- compilation partielle pour chaque processeur en vue d'une génération de code embarqué ; les fonctions d'entrées-sorties des processeurs se font via les fonctions d'émission ou de réception des processus de communication concernés.

Si l'expérimentation réalisée permet de valider l'approche proposée, il faut cependant remarquer que dans le cas général (multi-horloge), l'intégration via le format commun DC ne permet pas de retrouver directement les horloges qui caractérisent l'application. Elle est donc insuffisante lorsque l'on veut appliquer dans sa pleine mesure la théorie de la distribution des

programmes synchrones développée autour de Signal, qui s'appuie fortement sur la structure des horloges. Cela appelerait donc une intégration à un niveau plus élevé que celui de DC.

Une réponse partielle à cette observation a été fournie par la réalisation (sous forme de maquette) d'un traducteur de Lustre en Signal (les programmes sont traduits ici au niveau syntaxique). Pour les programmes Lustre traduits, il est alors possible d'appliquer toutes les fonctionnalités du compilateur Signal.

Modélisation synchrone de la norme IEC 1131 de programmation des systèmes de contrôle

Ce travail se fait en coopération avec Éric Rutten (projet Bip, Inria Rhône-Alpes).

Les systèmes de contrôle industriels sont des systèmes à la fois complexes et à la sécurité critique. Leur conception repose sur des standards, tels que la norme de l'International Electrotechnical Commission IEC 1131 [IEC93], et en particulier sa partie IEC 1131-3 concernant les langages de programmation. Nous travaillons à l'intégration de ce standard de spécification et de l'environnement de programmation synchrone Signal : il s'agit d'établir un modèle de ces langages, visant à offrir à de telles spécifications l'accès aux outils d'analyse et de mise en œuvre des techniques synchrones.

Les langages concernés sont deux langages textuels : *Instruction List,* IL, proche de l'assembleur, et le texte structuré *Structured Text,* ST, un langage impératif séquentiel, de la famille de Pascal, Ada, ou C ; et deux langages graphiques : les schémas à relais (*Ladder Diagrams,* LD), et les diagrammes de blocs-fonctions (*Functional Block Diagrams,* FBD). En plus de ces langages, la norme utilise une variante de Grafcet, SFC (*Sequential Function Charts*), comme langage graphique pour la modélisation des aspects des fonctions de commandes séquentielles des systèmes de contrôle. Le tout fonctionne dans le contexte d'un schéma d'exécution cyclique, faisant intervenir une notion de pas de réaction ou *scan.*

On a construit un modèle en Signal du langage ST [20, 21], réutilisant des résultats de l'étude sur l'interopérabilité entre Statemate et Signal [11, 15]. Les affectations en séquence sont codées dans un graphe de dépendances de données, les conditionnelles par des sélections (sous-échantillonnages) : on reste donc dans le même instant synchrone. Pour les itérations, dans le cas borné on peut développer en séquence la répétition du corps de boucle, mais cette solution paraît coûteuse. Pour les itérations non bornées, elle n'est pas suffisante : pour les représenter, on utilise le mécanisme de sur-échantillonnage de Signal, de façon à insérer entre le début et la fin de la boucle les instants synchrones nécessaires au calcul de chaque itération.

Le travail en cours comprend la modélisation en Signal des éléments du langage SFC. Nous associons à chaque étape une équation Signal qui définit les conditions d'activation et désactivation de l'étape considérée. On peut alors vérifier à l'aide de Sigali des propriétés d'un diagramme SFC, telles que : la stabilité du diagramme SFC (au cas où on applique un algorithme d'interprétation avec recherche de stabilité), la détection de la possibilité de réactivation d'une étape (qui est interdite par la norme) ou la détection de possibilités de blocage.

[IEC93] IEC, « International Standard for Programmable Controllers », *rapport de recherche n° IEC 1131 parts 1–5*, IEC (International Electrotechnical Commission), 1993.

6.5 Vérification et validation

Participants : Albert Benveniste, Michel Le Borgne, Hervé Marchand, Mirabelle Nebut, Sophie Pinchinat.

Mots clés : vérification, Sigali, système dynamique polynomial, Signal, génération d'automate, bisimulation, méthodes symboliques, système hybride, co-simulation Signal/Simulink.

Résumé : *Dans le cadre de la vérification des programmes, nous avons défini différents types d'équivalences comportementales pour les systèmes dynamiques polynomiaux. Des algorithmes symboliques pour le calcul de bisimulations ont été proposés et implémentés dans Sigali. Dans le cadre de la compilation des programmes, nous visons à améliorer significativement la synthèse du contrôle des programmes Signal. Pour cela, une exploitation de l'interprétation plus fine des définitions d'horloges est à l'étude. Dans le cadre général de la validation des modèles, des travaux portant sur la combinaison de systèmes discrets et de systèmes continus en utilisant Signal et Simulink on été menés à bien en collaboration avec l'université de Linköping.*

Sigali : plate-forme de vérification basée sur les automates
Nous avons poursuivi le développement des méthodes de vérification basées sur les automates. L'approche est basée sur des modèles comportementaux *intensionnels* (encore appelés symboliques ou implicites), obtenus par abstraction booléenne à partir des spécifications en langage Signal ou en DC+. Nous nous sommes particulièrement intéressés à la notion d'abstraction d'automates [32]. D'après [Ouv99], on retrouve en fait trois grands types d'abstractions :

- *Abstraction par fusion d'états.* Pour cette catégorie, nous avons défini un algorithme qui calcule symboliquement le système quotient modulo une relation d'équivalence d'états. Cette relation pouvant par exemple résulter du calcul d'une plus grande auto-bisimulation (forte, faible, ou encore *branching*). D'autres types d'équivalences moins «contraignantes» que la bisimulation ont également été considérés.

  Les algorithmes de calcul d'équivalence par bisimulations symboliques ont été implémentés dans l'outil de vérification Sigali. Concernant le calcul du modèle implicite quotient, un premier algorithme a été proposé et également implémenté dans Sigali. Ceci nous a notamment amené à intégrer dans Sigali de nouveaux types de réordonnancements de variables des TDD de manière à optimiser la taille des structures de données et la complexité des algorithmes.
- *Abstraction par restriction.* Elle consiste à restreindre le comportement du système en «coupant» soit des transitions soit des paquets d'états (e.g., états ne vérifiant pas une propriété). Dans les deux cas, cette abstraction entraîne, en règle générale, une perte de vivacité du système. C'est à ce niveau que la synthèse de contrôleur peut jouer un rôle

[Ouv99] Ouvrage Collectif (Coordination Philippe Schnoebelen), *Vérification de logiciels : Techniques et outils du model-checking*, Vuibert, 1999.

en calculant par exemple un contrôleur forçant le système obtenu à être vivace. Notons qu'il est alors possible d'utiliser les techniques d'abstraction par fusion pour réduire le nombre de variables qui sont nécessaires pour représenter les états accessibles du nouveau système.

– *Abstraction sur les variables.* Toujours dans un souci de diminuer la taille des systèmes dynamiques obtenus et des calculs ultérieurs de vérification, nous avons implémenté dans le décompilateur $\mathbb{Z}/3\mathbb{Z}$ la notion de *cône d'influence* qui permet de n'engendrer que la partie du système dynamique nécessaire à la vérification d'un ensemble de propriétés. Cette amélioration peut également être utilisée dans le cadre de la synthèse de contrôleurs.

Amélioration du calcul d'horloges

Traditionnellement l'horloge d'un signal (l'ensemble de ses instants de présence) est interprétée par une variable booléenne. En particulier la compilation d'un programme consiste à résoudre un système d'équations booléennes induites par les synchronisations de ce programme. L'expérience montre que cette interprétation est suffisante pour traiter des programmes dont le contrôle est essentiellement booléen mais est limitée dès qu'on aborde des conditions numériques.

Il est possible d'enrichir l'interprétation des horloges en prenant en compte les relations entre les valeurs des signaux présents à cette horloge. L'information apportée par une telle interprétation permet par exemple de traiter des conditions arithmétiques. Cette approche peut s'apparenter aux techniques d'interprétation abstraite présentées en 6.7 mais elle s'en distingue essentiellement en s'appuyant sur la notion de hiérarchie d'horloges.

Dans un premier temps, nous utilisons de telles interprétations pour améliorer la synthèse du contrôle de programmes Signal. En particulier, nous étudions des méthodes qui permettent de prouver l'égalité de deux horloges par analyse statique, là où la technique actuellement implantée trouve ses limitations.

Nous nous intéresserons ensuite au développement de méthodes incluant la sémantique comportementale des systèmes (i.e., leur dynamique) afin d'améliorer les méthodes précédentes, éventuellement par extension de l'outil Sigali.

Simulation de systèmes hybrides

Dans le cadre de la validation des modèles, nous avons étudié la possibilité de combiner des spécifications de systèmes discrets et de systèmes continus, en vue de permettre la simulation d'un programme de contrôle dans un environnement «réaliste», c'est-à-dire décrit par des spécifications analogiques.

L'approche utilisée consiste à connecter des composants discrets Signal (pour la partie contrôle) avec des composants continus Simulink (pour la partie analogique). Pour cela, les composants Signal sont compilés en fonctions C qui sont ensuite «embarquées» dans le code C compilé à partir du modèle Simulink. La taille des pas de calcul dans le modèle continu est fixée statiquement, puisque imposée par l'outil de compilation *Real-Time-Workshop* proposé dans Simulink. Toutefois, différents protocoles de communication entre le système continu et le contrôleur ont été implémentés. Les pas d'exécution du contrôleur Signal peuvent ainsi s'opérer

selon les protocoles suivants :

Protocole 1 : soit à chaque pas de calcul dans le modèle continu (activation périodique) ;

Protocole 2 : soit lors de certains pas de calcul dans le modèle continu (activation apériodique), i.e., lorsqu'un passage de seuil dans le modèle continu induit un changement de l'état au niveau du contrôleur ;

Protocole 3 : soit de façon asynchrone, i.e., suivant une horloge différente de celle des calculs dans le modèle continu.

Nous avons débuté l'étude par la mise en œuvre du co-simulateur d'un contrôleur discret d'une pompe non triviale proposée par C. Polhem. Les parties discrètes de cette co-simulation ont été spécifiées en Signal et embarquées dans un environnement continu spécifié à l'aide de Simulink. La simulation a notamment permis de mettre en évidence le manque de robustesse du contrôleur proposé par C. Polhem pour commander la pompe[1]. Les protocoles 1 et 2 ont été appliqués sur cet exemple.

Actuellement, nous testons cette approche dans le cadre d'un cas d'étude plus complexe : un générateur de vapeur [30]. Ce cas d'étude décrit un contrôleur de pompes, garantissant un niveau minimal de remplissage d'un réservoir. Ce problème nécessite un contrôleur relativement complexe. Dans ce cadre, la co-simulation fournit alors une aide importante dans la phase de débogage de celui-ci.

Ces travaux ont été conduits par Stéphane Tudoret dans le cadre du projet européen Syrf (voir section 7.3) en partenariat avec l'université de Linköping.

6.6 Synthèse automatique de contrôleurs

Participants : Hervé Marchand, Sophie Pinchinat.

Mots clés : système dynamique polynomial, synthèse de contrôleur, bisimulation, Sigali.

Résumé : *Sur la base des résultats de la Section 6.5 concernant le calcul de bisimulation symbolique et sur les travaux de* [BL98]*, nous avons défini une nouvelle méthode permettant la synthèse de contrôleurs pour des systèmes dynamiques polynomiaux. Le système contrôlé est obtenu par calcul d'une plus grande bisimulation symbolique entre le système et la spécification attendue du système. Des travaux visant à minimiser les contrôleurs obtenus sont également en cours.*

Bisimulation versus contrôle

Nous proposons cette année une alternative à la synthèse de contrôleurs par rapport à celle présentée en Section 3.2 [26, 31]. Nous nous plaçons dans le cadre *Ramadge et Wonham* [RW89]

1. Il faut noter que ce manque de robustesse est dû à l'algorithme de contrôle proposé et non aux protocoles de communication utilisés.

[BL98] G. Barret, S. Lafortune, « Bisimulation, the Supervisory Control Problem and Strong Model Matching for Finite State Machines », *Journal of Discrete Event Dynamic Systems 8*, 4, December 1998, p. 337–429.

[RW89] P. J. Ramadge, W. M. Wonham, « The Control of Discrete Event Systems », *Proceedings of the IEEE; Special issue on Dynamics of Discrete Event Systems 77*, 1, 1989, p. 81–98.

(les événements sont exclusifs et sont soit contrôlables (Σ_c) soit incontrôlables (Σ_{uc})). À partir de [BL98], nous avons développé une méthode efficace permettant de résoudre le problème de la synthèse de contrôleur en utilisant des techniques de bisimulation symbolique présentées en Section 6.5.

Schématiquement, le principe est le suivant : on se donne un système dynamique polynomial P modélisant le système à contrôler et un autre système dynamique polynomial S correspondant au comportement attendu du système contrôlé. Le calcul du système contrôlé est alors obtenu par le biais du calcul d'une plus grande auto-Σ_{uc}-bisimulation du système dynamique polynomial produit $P \times S$ (la Σ_{uc}-bisimulation est une variante de la bisimulation ne portant que sur les événements incontrôlables Σ_{uc}). Cette nouvelle approche nous permet d'appréhender des objectifs de contrôle plus généraux car fournis par un système dynamique polynomial et non plus simplement comme un ensemble d'états et une propriété sur celui-ci.

L'étude de cette relation entre la notion de contrôlabilité d'un système et la bisimulation se poursuit et nous essayons actuellement d'étendre ces résultats à des notions d'événements partiellement contrôlables (ayant une composante contrôlable et une autre incontrôlable (cf Section 3.2), ou encore à des événements dont le statut de contrôlabilité va dépendre de l'état courant du système. Nous regardons également comment introduire directement lors du calcul du contrôleur les techniques de réduction de modèle présentées dans la section 6.5 de manière à réduire la complexité du résultat.

Observation partielle

Toujours sur la base théorique proposée par Ramadge et Wonham, nous continuons à regarder le problème de la synthèse de contrôleur optimal[2] appliquée à un système partiellement observé (seul un sous-ensemble des événements est observable). À cet effet, un nouveau modèle intermédiaire prenant en compte le coût des trajectoires non observables a été introduit et un premier algorithme donnant lieu à un contrôleur optimal a été développé. Ces travaux sont conduits en collaboration avec Stéphane Lafortune de l'université du Michigan, Ann Arbor, MI, USA.

Minimisation des contrôleurs

Dans l'approche présentée en Section 3.2, le système à contrôler est donné par un système dynamique polynomial de la forme

$$\begin{cases} X' = P(X,Y,U) \\ Q(X,Y,U) = 0 \\ Q_0(X_0) = 0 \end{cases}$$

Pour un objectif de contrôle donné, le contrôleur est donné par une paire de polynômes ($C_0(X)$, $C(X,Y,U)$) qui sur-contraignent le système d'équations initial. Une des propriétés du contrôleur est que Q et C partagent des contraintes redondantes. On aimerait «retirer» de C les contraintes déjà imposées par le polynôme Q. Pour cela, nous utilisons les techniques de décomposition d'une équation polynomiale par cofacteur généralisé. Intuitivement, pour une

2. Le terme optimal signifie ici minimisation du coût le long d'une trajectoire bornée du système.

fonction polynomiale f et une contrainte g, le cofacteur de f par rapport à g est un polynôme $f_{|g}$ qui lorsque g s'annule est identique à f [3]. Appliquée à la synthèse, l'idée est de calculer un contrôleur $C'(X,Y,U) = C_{|Q}(X,Y,U)$, où $C_{|Q}$ est le résultat de la fonction cofacteur appliquée à C et Q. On obtient alors un contrôleur C' tel que $Q(X,Y,U) = 0$ implique $C'(X,Y,U) = 0 \Leftrightarrow C(X,Y,U) = 0$. C' est un polynôme avec moins de contraintes que C ; on peut de plus prouver que la taille du TDD de C' est inférieure à celle du TDD de C.
systèmes réactifs

6.7 Interprétation abstraite des systèmes réactifs

Participant : Jean-Pierre Talpin.

Mots clés : Signal, interprétation abstraite, vérification.

Résumé :
Afin d'essayer de pallier les limites des techniques de vérification appliquées actuellement, nous avons réalisé un outil d'interprétation abstraite de relations linéaires entre signaux entiers pour le langage Signal.

Dans le cadre de l'action incitative Presysa (voir section 8.2), et en collaboration avec le projet Lande (Frédéric Besson et Thomas Jensen), nous avons réalisé le prototype d'un outil d'interprétation abstraite pour le langage Signal [18]. Cet outil permet d'analyser les équations linéaires entre signaux entiers de programmes Signal. L'interprétation abstraite est utilisée afin de donner une approximation de l'espace des états accessibles par ces signaux au moyen de polyèdres. Le prototype a été mise en œuvre au moyen d'outils développés par les projets Lande et Api.

6.8 Preuves co-inductives de systèmes réactifs

Participants : Michaël Kerbœuf, David Nowak, Jean-Pierre Talpin.

Mots clés : Signal, sémantique synchrone, Coq, co-induction, vérification, preuve de théorème.

Résumé : *Nous avons réalisé une étude de cas mettant en œuvre conjointement un langage de programmation synchrone, Signal, et un assistant de preuve de théorèmes, Coq, en considérant le problème de la chaudière à vapeur (steam boiler).*

L'intérêt de l'utilisation combinée de Signal et de l'assistant de preuve Coq pour la conception des systèmes réactifs a pu être illustré sur un exemple concret et non trivial : le problème du *steam boiler* [12, 30]. Cette étude de cas a permis de révéler certains avantages de l'approche formelle Signal-Coq. En particulier, la spécification est naturelle, puisqu'indépendante de toute considération de vérification. Un assistant de preuve comme Coq permet en effet de s'affranchir

3. Dans $\mathcal{F}_2$, le résultat est canonique, par contre, dans $\mathcal{F}_3$, il existe plusieurs cofacteurs possibles selon le choix d'implémentation.

des contraintes portant sur les propriétés paramétrées ou portant sur des valeurs numériques non linéaires, auxquelles la vérification par model-checking est confrontée. En outre, l'utilisation de Coq permet d'acquérir une connaissance profonde de la spécification Signal et permet, à l'instar des contre-exemples fournis par un model-checker, de cibler précisément l'erreur en cas d'échec de la vérification. L'utilisation naturelle et intuitive de la co-induction confirme la pertinence du choix de cet outil pour modéliser les signaux du langage.

6.9 Mises en œuvre distribuées

Participants : Albert Benveniste, Loïc Besnard, Patricia Bournai, Thierry Gautier, Paul Le Guernic, Jean-Pierre Talpin.

Mots clés : Signal, DC+, programmation synchrone, génération de code enfoui, code distribué, architecture hétérogène, compilation séparée, causalité, communication synchrone/asynchrone.

Résumé : *La génération de code distribué pour les programmes synchrones pose deux grands types de difficultés. 1/ La compilation directe de code C (ou, plus généralement, de code séquentiel) n'est pas compatible avec une recomposition ultérieure avec d'autres modules. En d'autres termes, on ne peut pas compiler séparément (du moins de façon brutale) des programmes synchrones. 2/ Une architecture distribuée ne s'accomode pas, en général, de l'hypothèse de synchronisme parfait qui correspond au modèle de la programmation synchrone. Pour le premier problème, nous avons proposé une notion de* «tâche atomique» *qui constitue le grain maximal autorisant une compilation séparée avec réutilisation possible dans tout contexte [19]; ces résultats ont été exposés dans le rapport d'activité 1998. Pour le second problème, nous avons proposé les propriétés* d'endochronie *pour un programme synchrone, et* d'isochronie *pour un réseau de programmes synchrones. Ces deux propriétés garantissent une distribution correcte par construction, en s'appuyant uniquement sur les services d'une communication de type «send/receive» fiable [14, 16].*

Sur ces bases, une méthodologie a été développée pour la génération de code distribué, qui implique : 1/ la spécification par l'utilisateur, au niveau du programme source, de la répartition du programme, 2/ l'application de transformations automatisées du code intermédiaire, qui contrôle la correction du partitionnement, 3/ la génération du code distribué correspondant aux différents processus et à leurs communications. Nous renvoyons le lecteur à la section 6.4 pour la description de cette méthodologie sur un exemple particulier, et nous détaillons le point précédent.

Critères pour la distribution de programmes synchrones

La particularité du modèle synchrone (qui fait aussi sa souplesse) est la possibilité pour le contrôle, lors d'une réaction, de prendre des décisions sur la base de l'absence de tel ou tel événement. Ceci est évidemment impossible en univers distribué de nature asynchrone, où la notion de réaction n'existe pas (à moins de distribuer une horloge globale). Nous avons mis

en évidence une classe de programmes, dits *endochrones,* dont aucune décision de contrôle ne nécessite de tester l'absence de signaux dans l'environnement. Ces programmes peuvent être aisément utilisés en tant que modules dans une exécution répartie. Lorsqu'un module n'est pas endochrone, on sait le rendre endochrone en lui rajoutant un petit moniteur, spécifié en Signal. Ceci revient à attacher au module le protocole dont il a besoin, préalablement à toute répartition. Dans les cas standards, ceci est pris en charge par le compilateur Signal.

En collaboration avec B. Caillaud, du projet Pampa, nous avons également dégagé la notion d'*isochronie* pour une paire de programmes synchrones : deux programmes forment une paire isochrone si le remplacement du protocole de communication synchrone par une communication asynchrone par file non bornée ne change pas les comportements de chaque composant (bien sûr, le timing global du système résultant n'est pas préservé). Répartir une paire isochrone sur une architecture asynchrone est immédiat. Rendre une paire isochrone se fait, là encore, par ajout de protocoles synthétisés.

Cette année, nous avons approfondi nos résultats concernant l'isochronie, l'endochronie, et leurs propriétés de compositionnalité. En particulier, nous avons montré que l'endochronie n'est pas une propriété compositionnelle, mais que l'isochronie l'est. Nous avons également montré que l'isochronie est une propriété locale, en ce sens que, dans un réseau de processus synchrones, il suffit de tester, localement, l'isochronie entre composants en interaction pour obtenir l'isochronie de tout le réseau. Ces résultats ouvrent la voie vers l'utilisation de technologies synchrones avec instanciation dynamique et communications désynchronisées, donc vers les applications en programmation objet.

Distribution sur une architecture de type «Time Triggered»
Il existe des architectures ne fournissant pas les services de type send/receive fiable, requis pour que les techniques précédentes de déploiement sur architecture asynchrone s'appliquent. On en rencontre par exemple dans les bus de terrain utilisés en automatique embarquée. L'exemple le plus connu est l'architecture CAN/Osek utilisée en automobile. Selon ce type d'architecture, les composants capteurs, actionneurs, calculateurs, et bus, ont leur horloge périodique propre, ces horloges n'étant pas synchronisées. Les écritures et les lectures sur le bus sont non bloquantes. Dans un petit travail pour le projet Syrf (voir section 7.3), nous avons montré comment on pouvait adapter notre méthode ci-dessus à ce cas [29].

6.10 Structures de données pour le déploiement asynchrone de systèmes de transition synchrones

Participants : Albert Benveniste, Paul Le Guernic, Jean-Pierre Talpin.

Mots clés : systèmes de transitions synchrones, hiérarchisation.

Résumé : *Nous avons élaboré une structure de données fondée sur la notion de système de transitions synchrone permettant de décider les propriétés d'endochronie et d'isochronie au moyen d'algorithmes de hiérarchisation. Applicable à la fois au modèle abstrait de Signal et aux Spots introduits en section 6.3, cette structure de données systématise les travaux de P. Amagbegnon, L. Besnard et P. Le Guernic*

sur les hiérarchies d'horloges pour Signal. Ces structures de données sont appelées HNF *(Formes Normales Hiérarchiques).*

Nous avons défini les HNF, et nous avons donné des algorithmes incrémentaux pour les construire [33]. Ces structures de données sont différentes des BDD, en particulier elles ne nécessitent pas d'ordonnancement préalable des variables afin de devenir canoniques : elles fournissent un ordre partiel sur les variables en même temps qu'une structure d'inclusion d'horloges.

Nous avons montré que la HNF permet de définir des procédures de décision répondant aux problèmes essentiels posés par le déploiement de programmes synchrones sur des architectures distribuées (*désynchronisation*) :

- la propriété d'*endochronie* d'un système, c'est-à-dire l'équivalence entre son observation synchrone (interne) et son observation asynchrone (externe) ;
- la propriété d'*isochronie,* c'est-à-dire l'équivalence comportementale entre les compositions synchrones ou asynchrones de deux composants d'un système, propriété cruciale pour déployer correctement les composants d'un système spécifié de manière synchrone sur un réseau asynchrone.

6.11 Synthèse de circuits et conception de systèmes matériels/logiciels

Participants : Jean-Christophe Le Lann, Christophe Wolinski.

Mots clés : Signal, programmation synchrone, conception conjointe matériel/logiciel, composant matériel, synthèse de circuits, consommation des circuits.

Résumé : *Nous poursuivons notre étude dans le domaine de la synthèse comportementale de circuits digitaux à partir de Signal. Cette synthèse suppose le découpage d'un cycle Signal en sous-instants pendant lesquels on cherche à partager au mieux les ressources.*

Notre attention a porté cette année sur plusieurs aspects :

- pré-synthèse et ordonnancement sous contraintes ;
- génération de chemins de données sous contraintes de ressources ;
- génération d'automates de contrôle.

Pré-synthèse

Notre chaîne de conception se base sur la représentation interne du graphe hiérarchisé aux dépendances conditionnées (GHDC), format interne du compilateur Signal. La première étape consiste en des pré-traitements classiques en synthèse comportementale, mais également en une utilisation optimale des informations présentes dans la hiérarchie des gardes, par une restructuration du graphe. Dans l'optique d'une exécution spéculative, nous pouvons ainsi augmenter la fréquence d'exécution de certains nœuds de calcul lors du processus d'ordonnancement sous contraintes (réalisé en aval), ce qui permet d'optimiser l'utilisation des ressources. Cette pré-synthèse se termine par la génération de code VHDL comportemental. Il incorpore les résultats

effectivement utiles d'ordonnancement. Pendant la génération du code nous effectuons également les opérations d'optimisation de bas niveau comme l'initialisation des variables par une valeur *don't care* et la réduction de la taille des opérateurs et des bus de données. Les résultats de ce travail ont été présentés dans différents articles [22, 23, 24].

Génération de chemins de données sous contraintes de ressources
La pré-synthèse suppose l'utilisation d'outils de synthèse de haut niveau, comme le *Behavioral compiler* de Synopsys. Une autre possibilité est celle qui consiste à construire un outil de synthèse, afin d'étudier l'application de nouveaux algorithmes sur le GHDC. Ainsi, un prototype a été développé, qui permet notamment de prendre en compte différents ordonnancements et de distribuer statiquement les calculs sur des unités fonctionnelles de façon équilibrée [25]. Cet équilibrage peut par exemple avoir des conséquences non négligeables sur le gradient thermique du circuit. Un algorithme génétique a été élaboré afin de trouver la meilleure assignation. Les populations en présence correspondent à différentes assignations possibles. Chaque assignation est elle-même caractérisée par un nombre qui restitue une probabilité d'activation de telle ou telle unité fonctionnelle du système, probabilité calculée à partir des gardes associées aux opérations qu'elle est susceptible de réaliser à chaque cycle. La hiérarchisation des gardes est à nouveau d'un grand secours. L'insertion de matériel redondant (verrous transparents) dans le chemin de données reste à l'étude en ce qui concerne l'optimisation de la consommation. L'allocation des registres et l'assignation des variables profitent également du calcul d'horloges : nous avons étendu l'algorithme de Tseng-Sieworeck en y incluant les informations d'exclusion mutuelle trouvées par le compilateur. Enfin, en construisant judicieusement les arbres de multiplexeurs entre les registres et les unités fonctionnelles, on peut réduire le nombre d'étages de logique traversés et diminuer la consommation, sans ajout de matériel redondant.

Génération d'automates de contrôle
Dès lors que le chemin de données est réalisé, nous pouvons générer l'automate de contrôle, orchestrant l'ensemble. Les premiers résultats à nouveau que le calcul d'horloges permet d'accéder à des optimisations qu'un synthétiseur classique peut difficilement détecter.

Dans l'ensemble de ces activités, notons que l'utilisation du code C généré par le compilateur Signal V4 nous a permis d'accéder à des simulations rapides, que ce soit au niveau purement fonctionnel, mais aussi à des fins d'instrumentation : par exemple l'instrumentation qui vise à étudier l'impact des différentes assignations des unités fonctionnelles est vue comme un ensemble de nouvelles équations (générées automatiquement par notre prototype) dans le programme Signal d'origine. En ce sens, on retrouve une certaine affinité avec nos travaux précédents (thèse d'Apostolos Kountouris), qui visaient à mesurer les délais associés aux programmes Signal.

6.12 Architectures embarquées pour l'automobile

Participant : Thierry Gautier.

Mots clés : automobile, AIL, OIL, OSEK.

Résumé : *Dans le cadre du projet* AEE *(voir section 7.4), en collaboration avec les autres partenaires du projet, nous avons commencé à définir un «langage»,* AIL *(*Architecture Implementation Language*), dont l'objectif est de décrire une architecture véhicule depuis les prestations demandées jusqu'à l'implémentation, concrétisée par une ou plusieurs architectures opérationnelles résultantes.*

Le langage AIL doit permettre d'assurer la portabilité et la réutilisation d'éléments constitutifs d'une application automobile (composants logiciels et/ou matériels) et d'optimiser et valider l'architecture de l'application dans toutes les phases du processus de développement ; il doit permettre aussi d'assurer la génération du paramétrage OIL (*Osek Implementation Language*) qui décrit la mise en œuvre aux niveaux bas via le système Osek.

Les objets du langage se déclinent suivant différents points de vue :

- niveau «prestation», où la notion de «fonction-véhicule» est l'objet permettant de décrire tout ou partie de la réalisation d'une prestation ;
- niveau «fonctionnel», où l'on décrit une architecture fonctionnelle comme un assemblage de fonctions élémentaires échangeant des données ;
- niveau «implémentation», qui comprend la description de l'architecture matérielle, l'architecture opérationnelle, représentant la projection d'une architecture fonctionnelle sur une architecture matérielle, et l'implémentation des fonctions élémentaires via leurs entités logicielles applicatives et logiciels de base servant d'interface entre les applications et le support d'exécution (la fonction élémentaire est une entité fonctionnelle allouée à un seul calculateur lors de l'opération de placement).

Dans un premier temps, il a été défini une liste des objets à manipuler dans AIL sous forme de classes «à la UML» décrivant les attributs caractéristiques de ces objets et leurs liens. Reste à formaliser cette définition, à prévoir l'outillage à définir dans le projet AEE et les passerelles à assurer avec des outils existants.

6.13 Techniques d'algèbre MaxPlus pour l'évaluation de performances

Participants : Albert Benveniste, Pierre Le Maigat.

Mots clés : Signal, évaluation de performance, algèbre MaxPlus.

Glossaire :

évaluation de performance : technique permettant l'analyse du comportement temporel (au sens du temps réel) d'un programme sur une architecture.

automate temporisé : modèle consistant à affecter des spécifications de comportement temporel aux actions situées sur les transitions d'un automate.

algèbre MaxPlus : algèbre linéaire construite sur le semi-anneau $\{\mathbb{Z}, max, +\}$, elle est adaptée à la manipulation formelle pour les modèles d'évaluation de performance.

Résumé : *Nous avons développé une nouvelle direction de recherche visant à utiliser les approches d'algèbre MaxPlus, promues et étudiées au sein du groupe du même nom (G. Cohen, J-P. Quadrat, F. Baccelli, S. Gaubert), pour étudier le comportement temporel de divers modèles de programmation, dont Signal et les* MSC

(Message Sequence Charts, notation de type scénario). Ce travail est réalisé avec Claude Jard et Loïc Hélouët (projet Pampa) et Stéphane Gaubert (projet Meta2, Rocquencourt).

Grâce au travail conduit au sein du projet EP-ATR par A. Kountouris et P. Le Guernic [6], nous disposions d'une technique permettant d'associer, à tout couple constitué d'un programme Signal et d'une architecture support (également spécifiée en Signal), un autre programme Signal modélisant la consommation de temps par ce programme s'exécutant sur l'architecture considérée. Restait à développer une technique symbolique permettant le raisonnement sur ce type de modèle. Nous avons développé une extension des techniques d'algèbre MaxPlus, promues et étudiées au sein du groupe du même nom, permettant de couvrir le cas qui nous intéressait. Dans cette algèbre, la manipulation symbolique de modèles d'évaluation de performance de programmes Signal se présente comme un problème d'algèbre linéaire, avec toutefois des spécificités dues aux particularités du semi-anneau «MaxPlus». Nous avons appelé *diagrammes temporels* les représentations graphiques que l'on peut associer à de tels modèles.

Cette année nous avons poursuivi l'étude de ce modèle introduit dans le rapport d'activité 1998. En particulier, P. Le Maigat a approfondi les relations algébriques entre automates d'ordre, automates Max-Plus, et diagrammes temporels tels qu'introduits dans notre étude. L. Hélouët (du projet Pampa) et P. Le Maigat ont entamé une étude pour la traduction de hMSC temporisés (high-Level Message Sequence Charts) vers le modèle des diagrammes temporels. La prise en compte des compteurs des MSC a demandé l'introduction d'extensions de notre précédent modèle où l'on ne raisonne plus seulement sur des dates au plus tôt, mais aussi sur des contraintes de ponctualité (dates au plus tard).

7 Contrats industriels (nationaux, européens et internationaux)

convention Alcatel Alsthom Recherche/Inria n° 197A93600000MC012 (09/1997–08/2000)

7.1 Projet Reutel-2000, convention Alcatel Alsthom Recherche/Inria n° 197A93600000MC012 (09/1997–08/2000)

Participants : Albert Benveniste, Paul Le Guernic, Jean-Pierre Talpin, Yunming Wang.

Mots clés : télécommunication, communication synchrone/asynchrone, conception objet, BDL.

Résumé : *Le projet Reutel-2000 a pour but la réalisation d'outils pour l'aide au développement d'applications distribuées temps-réel en télécommunication. Les autres partenaires de cette action sont les projets Inria Pampa,* ADP *et Compose.*

L'objectif général du projet est la maîtrise du développement logiciel d'applications de télécommunication par la conception d'outils de manipulations formelles à l'intérieur d'une chaîne de développement définie par Alcatel. Il est pour cela nécessaire d'apporter à l'environnement de programmation en question des outils pour l'analyse, la vérification et l'optimisation d'applications. L'interface de ces outils avec la chaîne de développement d'Alcatel est le langage de

spécification BDL. Son rôle est d'accompagner l'utilisateur depuis les tâches préliminaires de spécification jusqu'aux travaux de codage et de test. Il doit aussi servir d'interface à l'intégration des outils de l'Inria dans la chaîne de développement d'Alcatel.

Dans le cadre de ce projet, nous avons réalisé le prototype d'un traducteur de Statecharts vers BDL.

7.2 Projet Esprit Sacres, convention n°195C4170031307006 (11/1995–01/1999)

Participants : Albert Benveniste, Loïc Besnard, Thierry Gautier, Paul Le Guernic, Éric Rutten [(maintenant dans le projet Bip, Inria Rhône-Alpes)].

Mots clés : système enfoui, méthode formelle, format commun, DC+, Signal, Statecharts, Sildex, avionique, génération de code enfoui, code distribué, compilation séparée, vérification, validation de code.

Résumé : *Le projet Esprit Sacres («Safety Critical Embedded Systems: From Requirements to System Architecture») a pour objectif, dans une perspective de commercialisation de ses résultats, de fournir aux concepteurs de systèmes critiques embarqués une nouvelle méthodologie permettant de réduire significativement le risque d'erreurs et le temps de conception. L'approche proposée est multi-formalisme et s'appuie sur des outils industriels existants, Statemate et Sildex notamment, pour lesquels des outils de vérification formelle et de génération de code réparti doivent être intégrés. Le vecteur de cette intégration est le format commun* DC+.

`http://www.tni.fr/sacres`

Présentation générale

Le projet Esprit 20897 IT (R&D) Sacres s'est achevé en janvier 1999. Il regroupait les organismes suivants : Siemens (RFA), British Aerospace (Grande-Bretagne), i-Logix (Grande-Bretagne), Inria (France), Offis (RFA), Snecma (France), TNI (France) et le Weizmann Institute (Israël).

Le but du projet était de fournir aux concepteurs de systèmes embarqués, en particulier de systèmes critiques sûrs de fonctionnement, une meilleure méthodologie de conception permettant de réduire significativement tant le risque d'erreurs que le temps de conception. Pour cela, la validation des spécifications initiales doit se faire à l'aide d'outils de vérification formelle intégrés, et les phases de génération de code, réparti notamment, doivent être automatisées.

L'approche proposée dans Sacres est multi-formalisme. Elle s'appuie sur un certain nombre d'outils existants : Statemate,/Statecharts, Sildex,/Signal, Sildex /Grafcet, Timing Diagrams. Les résultats du projet sont commercialisés par les partenaires industriels vendeurs (i-Logix et TNI) et ont été utilisés avec succès par les partenaires industriels utilisateurs (British Aerospace, Siemens et Snecma).

L'architecture de l'environnement Sacres, illustrée dans la figure 2, montre qu'entre les langages à la disposition des utilisateurs, les outils de vérification, de validation et ceux de génération de code, le format d'échange DC+ joue un rôle central.

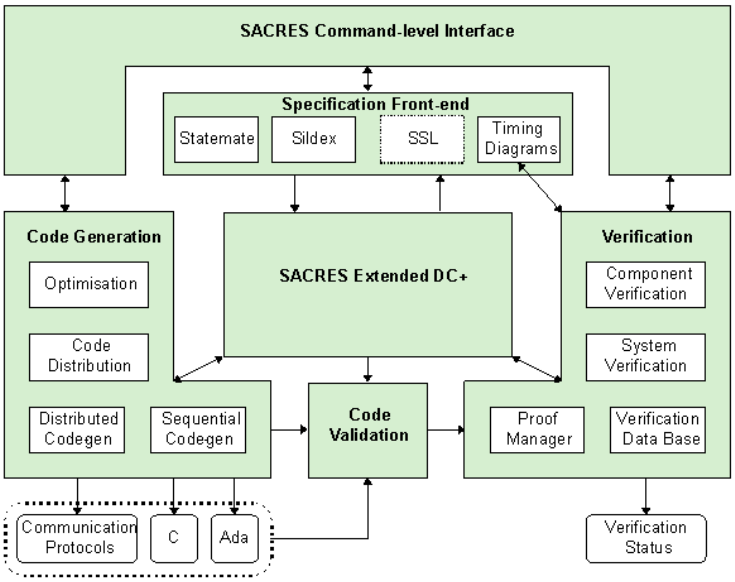

FIG. 2 – *Architecture globale de l'environnement Sacres*

Le format d'échange DC+

Le format DC+, issu des travaux du projet européen Synchron, sert de vecteur commun de représentation, pour des programmes (et des propriétés) décrits complètement ou partiellement à l'aide de Signal ou de Statecharts, et à destination d'outils de vérification et de génération de code.

Différents niveaux de DC+, ou sous-formats, ont été identifiés (voir figure 3). La caractérisation de ces différents niveaux et des transformations inter-niveaux (ou inter-formats), a pour objectif d'adapter une représentation aux fonctions qui peuvent lui être appliquées.

Ainsi, le sous-format bDC+ (pour «boolean DC+»), dans lequel les horloges, représentées comme des flots booléens, sont organisées en une hiérarchie pour laquelle il existe une horloge

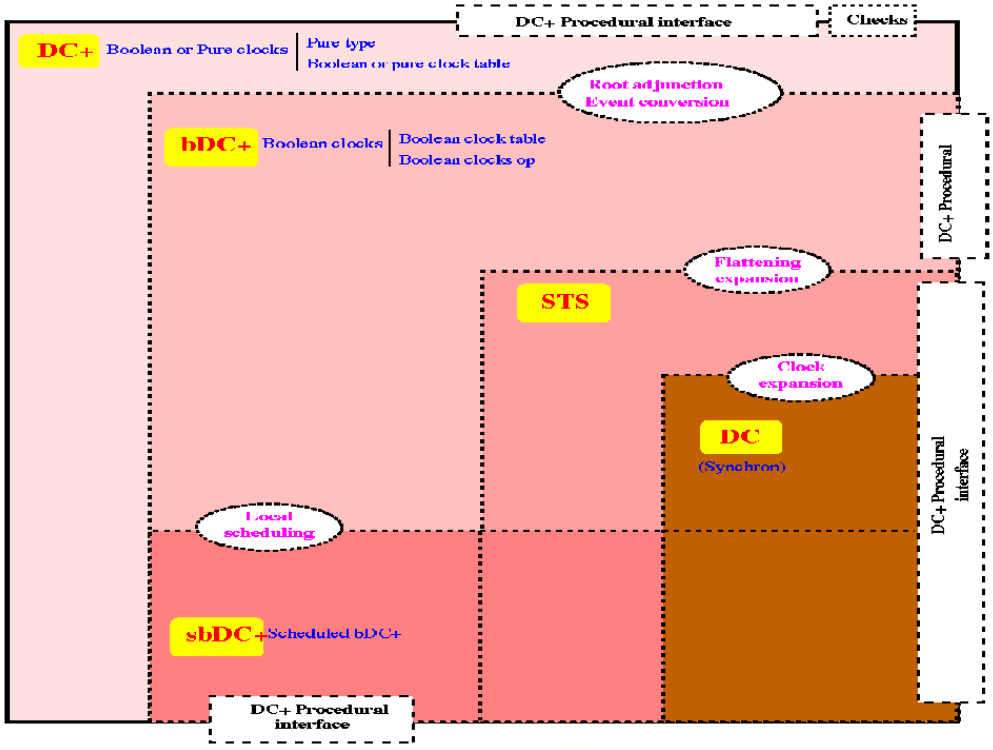

FIG. 3 – L'architecture DC+

maîtresse, est le point d'entrée adéquat pour des outils s'appuyant sur la hiérarchie des horloges, comme par exemple des générateurs de code.

Le sous-format STS (pour «Symbolic Transition Systems») de bDC+, dans lequel la hiérarchie des horloges est plate (le statut présent/absent d'un signal est défini à tout instant par un booléen), est utilisé en entrée d'outils de vérification.

Les transformations inter-formats, DC+ → bDC+, bDC+ → STS et STS → DC ont été définies et mises en œuvre dans le cadre, désormais commun, de la *machine virtuelle* DC+ et du compilateur Signal (voir section 6.1).

Activités de EP-ATR dans le cadre du projet Sacres

Le projet Sacres s'est conclu en début d'année, avec :

– l'obtention d'un ensemble de résultats sur la modélisation synchrone de Statemate [11, 15], avec une version de l'intégration des langages de Statemate à l'environnement DC+, et sa mise en œuvre sous la forme d'un prototype de traducteur en format commun DC+, connecté à l'Api de l'outil Statemate de iLogix. Ces résultats peuvent avoir des répercussions dans l'étude des variantes directes des Statecharts, comme celle de UML, ou des langages différents mais présentant des similarités de structure et de comportement, comme les langages de la norme IEC 1131-3 (voir section 6.4). L'exploration des notions

de granularités de temps peut aussi trouver écho dans la modélisation des mises en œuvre de programmes synchrones.

- l'obtention d'une *machine virtuelle* DC+, réorganisation et généralisation de l'outil de compilation Signal, suite aux travaux communs avec TNI de définition et de mise en œuvre autour du format commun DC+ (voir section 6.4). Ceci comprend le développement d'algorithmes de génération de code sur la machine DC+ (booléanisation des événements, séquentialisation/ordonnancement des calculs, production de code C ou Ada). Il s'y trouve aussi une version d'outil de préparation à la vérification de niveau système, par la traduction des aspects structurels d'une application décrite dans le format DC+ vers le formalisme de spécification au niveau système SSL, point d'entrée des outils de vérification développés par Offis.
- l'obtention de résultats concernant les méthodes de distribution de programmes à partir de leur modèle DC+ : fonction d'allocation, marquage du graphe DC+, extraction de sous-graphes, génération de code pour les communications. La modélisation d'architectures au moyen de DC+ a aussi été étudiée. Ces résultats se sont accompagnés de contacts, qui se sont poursuivis, avec le Weizmann Institute et iLogix concernant la génération de mises en œuvre distribuées de programmes synchrones et réactifs.
- l'acquisition d'une connaissance dans les domaines applicatifs représentés par nos partenaires industriels, de par nos contacts à l'occasion des applications dans le projet. Notons aussi l'apport dû à notre participation à l'effort de transfert de technologie et de diffusion des résultats. Il s'agit notamment de notre participation à l'organisation et à la réalisation de rencontres avec un groupe d'industriels sensibilisés à l'approche Sacres, en vue d'établir des coopérations techniques autour de l'utilisation de l'environnement. Nous avons aussi participé à la diffusion par la présentation d'exposés dans des conférences.

7.3 Projet Esprit Syrf, convention n°197G04900MPG211 (01/1997–12/1999)

Participants : Albert Benveniste, Thierry Gautier, Paul Le Guernic, Hervé Marchand, Sophie Pinchinat.

Mots clés : format commun, DC+, Signal, programmation synchrone, système hybride, code distribué, compilation séparée, génération d'automate, vérification, génération de test, conception conjointe matériel/logiciel.

Résumé : *Le projet Esprit Syrf («Synchronous Reactive Formalisms») a pour objet de développer des études autour des formalismes synchrones dans un cadre coopératif industriel et de recherche initié dans le projet Eureka Synchron.*

`http://www-verimag.imag.fr/SYNCHRONE/SYRF/syrf.html`

Le projet Esprit Syrf 22703 complète les travaux des projets européens Synchron et Sacres.

Les participants sont l'Inria (ex-projet Spectre de Grenoble, projets Meije de Sophia-Antipolis et EP-ATR de Rennes), le German National I.T. Research Institute, la compagnie suédoise Logikkonsult, l'institut de recherche suédois de l'Université de Linköping, la compagnie Schneider-Électrique France, la compagnie suédoise Saab Military Aircraft et Électricité de France.

Il a pour objectifs :

- de combiner les différents formalismes synchrones,
- d'étendre les techniques de vérification automatique,
- de développer la génération de code vers des architectures distribuées,
- d'étudier les modèles qui combinent des aspects synchrones et asynchrones,
- d'établir un lien avec la conception matériel/logiciel,
- la conception du synchronisme analogique ou discret.

Le projet EP-ATR est plus particulièrement impliqué dans les thèmes suivants :

Techniques symboliques pour la manipulation d'automates. Ce thème général recouvre l'utilisation de moteurs de technologie BDD ou autre pour la résolution de problèmes tels que : vérification, test, calculs d'abstractions, synthèse de contrôleurs.

Intégration du synchrone et de l'asynchrone. L'objectif est de comprendre comment basculer d'un point de vue synchrone vers un point de vue radicalement asynchrone, où le temps est vu comme un ordre partiel, et vice-versa. La génération automatique de code réparti est l'un des principaux résultats attendus. Il s'agit évidemment d'un thème important pour le projet, c'est également un des thèmes majeurs pour EP-ATR dans le projet Esprit Sacres.

Hardware et co-design. L'effort porte sur les liens entre formalismes synchrones et VHDL ou Verilog.

Études de cas proposées par EDF. Il s'agit soit d'études de type architectural, soit de l'utilisation des techniques de synthèse de contrôleur dans le cadre de la conduite d'une centrale. Un papier de synthèse portant sur la synthèse automatique de contrôleur d'un poste de transformation électrique a été publié en [26].

7.4 Projet AEE, convention n° 99.2.93.0122 (01/1999–04/2002)

Participant : Thierry Gautier.

Mots clés : automobile, conception conjointe matériel/logiciel, composant matériel.

Résumé : *Le projet AEE («Architecture Electronique Embarquée») vise les applications de transport, et notamment l'automobile. Il a pour objet de concevoir et valider un processus rapide et sûr pour la définition de l'architecture système et le développement des logiciels associés embarqués, en assurant l'indépendance entre matériel et logiciel.*

http://aee.inria.fr/

Les partenaires du projet AEE sont : PSA Peugeot Citroën, Renault, Aérospatiale Matra Lanceurs, Sagem, Siemens, Valeo, l'Ircyn – École Centrale de Nantes, le Loria et l'Inria. Le projet est soutenu par le Secrétariat d'État à l'Industrie. À l'Inria, les projets impliqués sont Sosso (Yves Sorel), EP-ATR, Meije, Pampa et Reflecs.

Les objectifs généraux sont les suivants :

- assurer une offre produit «innovante, économique et compétitive» pour le logiciel embarqué (maîtrise du compromis coût/innovation et prestations client améliorées) ;
- assurer la maîtrise du développement (processus de développement, validation, standardisation des échanges entre constructeurs et équipementiers, sûreté de l'électronique auto embarquée) ;
- permettre l'indépendance du logiciel et du matériel (constituants interopérables, compatibilité OSEK/VDX) ;
- régler les aspects contractuels.

Le projet EP-ATR est plus particulièrement impliqué dans la partie du projet consistant à permettre la conception et la maîtrise d'une architecture système flexible, avec d'une part, le développement des outils de définition de l'architecture, et d'autre part, le développement d'un environnement d'allocation.

7.5 TNI

Mots clés : Signal, Sildex.

Résumé : *La société TNI, qui développe et commercialise l'environnement Sildex pour Signal, est un partenaire associé à nombre de nos activités.*

http://www.tni.fr/produits/genielo/sildex/indexgb.html

Nous collaborons étroitement avec la société TNI, qui assure l'industrialisation de Signal à travers l'environnement Sildex. Un axe essentiel de cette collaboration concerne la diffusion du synchrone en général, et en particulier des outils développés d'un côté et de l'autre autour de Signal.

Notre collaboration avec TNI s'effectue également au sein du projet européen Sacres.

8 Actions régionales, nationales et internationales

8.1 Actions régionales

Mots clés : Signal, Grafcet, Statecharts.

Groupe Télé-Productique Bretagne

`http://doelan-gw.univ-brest.fr:8080/teleproductique/teleprod.html`

Un groupe de travail sur la productique se réunit régulièrement, rassemblant des partenaires intéressés dans l'Ouest, à Brest (Ubo, ENIB, IUT, ENSIETA) et à Rennes (Insa, Supelec, ENS-Cachan à Bruz). Des intérêts communs avec l'Ubo et Supelec se sont dégagés, concernant la modélisation de contrôleurs de cellule de production en Signal, Grafcet ou Statecharts, ainsi que les architectures d'automates programmables.

8.2 Actions nationales

Mots clés : vérification, preuve de théorème, interprétation abstraite.

Groupe Grafcet du club EEA

`http://www.lurpa.ens-cachan.fr/grafcet/`

Un groupe de travail de l'EEA (association d'enseignants et chercheurs en Electronique, Electrotechnique et Automatique) est consacré au Grafcet, et plus largement aux formalismes liés aux automatismes industriels. Il rassemble des industriels et enseignants-chercheurs francophones actifs sur le sujet, notamment dans l'établissement des normes de l'IEC, et dans la diffusion des résultats dans l'enseignement. Nous y contribuons en y présentant nos résultats et les spécificités de l'approche synchrone.

Action coopérative Presysa

`http://www.loria.fr/\~rusi/action`

L'action coopérative Inria Presysa se situe dans l'axe intitulé «développement de logiciels certifiés». Elle concerne la vérification de systèmes synchrones et asynchrones ayant un nombre d'états infinis en se concentrant sur les propriétés de sûreté et d'invariance. Les participants sont les suivants : CMI (Marseille), Verimag (Grenoble), Loria (Nancy) et Irisa (Rennes)

Actuellement, les méthodes algorithmiques ne peuvent être utilisées que sur des systèmes infinis particuliers. Par ailleurs, les méthodes déductives sont difficiles à mettre en œuvre et demandent un fort investissement. Pour cette raison, l'automatisation et l'intégration des méthodes de vérification sont des facteurs clés dans le succès d'un assistant de preuve.

Les techniques d'automatisation de la preuve que nous proposons d'utiliser sont basées sur la réécriture. L'intégration des méthodes de vérification se fera en particulier par combinaison de preuve déductive et de *model-checking,* en se basant sur des techniques d'interprétation abstraite.

9 Diffusion de résultats

9.1 Animation de la communauté scientifique

A. Benveniste a fait partie du comité de programme de «FEMSYS 1999 : Workshop on Formal Design of Safety Critical Embedded Systems» (Munich, RFA, 15–17 mars 1999), dont il était aussi «co-chair».

Th. Gautier a fait partie du comité de programme de «ISLIP'99 : 12th International Symposium on Languages for Intensional Programming» (Athènes, Grèce, 28–30 juin 1999).

9.2 Enseignement universitaire

Les membres de l'équipe participent à divers titres à la formation d'étudiants à l'université et à l'Insa.

P. Le Guernic, S. Pinchinat, Th. Gautier, B. Houssais et L. Besnard ont donné des cours de DEA, 5^{e} année Insa, DESS-Isa, Diic 2^{e} année et 3^{e} année d'IUP (UBS) consacrés à la programmation temps réel. Ils ont aussi participé à la formation Futé de l'Ifsic destinée à des ingénieurs de Thomson Multimedia.

Dans le cadre des formations de troisième cycle, nous avons assuré l'encadrement d'étudiants stagiaires de DEA informatique : Mickaël Kerbœuf, et Stéphane Tudoret (en co-encadrement avec Simin Nadjm-Tehrani, de l'Université de Linköping, Suède).

9.3 Participation à des colloques, séminaires, invitations

On pourra se reporter à la bibliographie pour la liste des colloques et congrès auxquels les membres de l'équipe ont participé.

10 Bibliographie

Ouvrages et articles de référence de l'équipe

[1] T. P. AMAGBEGNON, L. BESNARD, P. LE GUERNIC, « Implementation of the Data-flow Synchronous Language Signal», *in : Proceedings of the ACM Symposium on Programming Languages Design and Implementation (PLDI'95)*, ACM, p. 163–173, 1995, `http://www.irisa.fr/ep-atr/Publis/Abstract/amabegnon95d.htm`.

[2] T. P. AMAGBEGNON, P. LE GUERNIC, H. MARCHAND, E. RUTTEN, « Signal- the specification of a generic, verified production cell controller », *in : Formal Development of Reactive Systems - Case Study Production Cell*, C. Lewerentz, T. Lindner (éditeurs), *Lecture Notes in Computer Science*, 891, Springer Verlag, p. 115–129, janvier 1995, `http://www.irisa.fr/ep-atr/Publis/Abstract/amabegnon95a.htm`.

[3] P. AUBRY, P. LE GUERNIC, S. MACHARD, « Synchronous distribution of Signal programs », *in : Proc. of the 29th Hawaii International Conference on System Sciences, 1*, IEEE Computer Society Press, p. 656–665, Janvier 1996, `http://www.irisa.fr/ep-atr/Publis/Abstract/aubry96a.htm`.

[4] M. BELHADJ, « VHDL & Signal: A Cooperative Approach », *in : International Conference on Simulation and Hardware Description Languages, Western Simulation Multi-Conference*, Society for Computer Simulation, p. 76–81, Tempe, Arizona (USA), 1994.

[5] A. Benveniste, P. Le Guernic, C. Jacquemot, « Synchronous programming with events and relations: the Signal language and its semantics », *Science of Computer Programming 16*, 1991, p. 103–149.

[6] A. Kountouris, P. Le Guernic, « Profiling of Signal Programs and its Application in the Timing Evaluation of Design Implementations », *in : Proc. of the IEE Colloq. on HW-SW Co-synthesis for Reconfigurable Systems*, IEE, p. 6/1–6/9, Février 1996, `http://www.irisa.fr/ep-atr/Publis/Abstract/kountouris96a.htm`.

[7] P. Le Guernic, T. Gautier, M. Le Borgne, C. Le Maire, « Programming Real-Time Applications with Signal », *in : Proceedings of the IEEE, 79*, 9, p. 1321–1336, Septembre 1991, `http://www.irisa.fr/ep-atr/Publis/Abstract/leguernic91a.htm`.

[8] P. Le Guernic, T. Gautier, « Data-Flow to von Neumann: the Signal approach », *in : Advanced Topics in Data-Flow Computing*, J. L. Gaudiot, L. Bic (éditeurs), p. 413–438, 1991, `http://www.irisa.fr/ep-atr/Publis/Abstract/leguernic91c.htm`.

[9] H. Marchand, M. Le Borgne, « Partial Order Control of Discrete Event Systems modeled as Polynomial Dynamical Systems », *in : 1998 IEEE International Conference On Control Applications*, Trieste, Italie, 1998, `http://www.irisa.fr/ep-atr/Publis/Abstract/marchand98a.htm`.

[10] E. Rutten, P. Le Guernic, « Sequencing data flow tasks in Signal », *in : Proceedings of the ACM SIGPLAN Workshop on Language, Compiler and Tool Support for Real-Time Systems*, Orlando, Florida (USA), Juin 1994, `http://www.irisa.fr/ep-atr/Publis/Abstract/rutten94b.htm`.

Thèses et habilitations à diriger des recherches

[11] J. R. Beauvais, *Modélisation de StateCharts en Signal pour la conception de systèmes critiques temps-réel*, thèse de doctorat, Ifsic, Université de Rennes 1, janvier 1999, `http://www.irisa.fr/ep-atr/Publis/Abstract/beauvais99.htm`.

[12] D. Nowak, *Spécification et preuve de systèmes réactifs*, thèse de doctorat, Ifsic, Université de Rennes 1, octobre 1999, `http://www.irisa.fr/ep-atr/Publis/Abstract/nowak99b.htm`.

[13] E. Rutten, *Programmation sûre des systèmes de contrôle/commande : le séquencement de tâches flot de données dans les langages réactifs*, Habilitation à diriger des recherches, Ifsic, Université de Rennes 1, décembre 1999.

Articles et chapitres de livre

[14] A. Benveniste, B. Caillaud, P. Le Guernic, « Compositionality in dataflow synchronous languages: specification & distributed code generation », *Information and Computation*, 1999, à paraître.

Communications à des congrès, colloques, etc.

[15] J. R. Beauvais, R. Houdebine, Y. M. Tang, P. Le Guernic, E. Rutten, T. Gautier, « Une modélisation de StateCharts et ActivityCharts en Signal », *in : Actes du 2ème Congrès sur la Modélisation des Systèmes Réactifs, MSR'99*, p. 213–222, Cachan, mars 1999, `http://www.irisa.fr/ep-atr/Publis/Abstract/beauvais99a.htm`.

[16] A. Benveniste, B. Caillaud, P. Le Guernic, « From synchrony to asynchrony », *in : CONCUR'99, Concurrency Theory, 10th International Conference*, J. Baeten, S. Mauw (éditeurs), *Lecture Notes in Computer Science, 1664*, Springer, p. 162–177, août 1999.

[17] L. Besnard, P. Bournai, T. Gautier, N. Halbwachs, S. Nadjm-Tehrani, A. Ressouche, « Design of a multi-formalism application and distribution in a data-flow context: an example », *in : Proceedings of The 12th International Symposium on Languages for Intensional Programming, ISLIP' 99*, M. Gergatsoulis, P. Rondogiannis (éditeurs), p. 8–30, NCSR Demokritos, Athens, Greece, juin 1999, `http://www.irisa.fr/ep-atr/Publis/Abstract/gautier99b.htm`.

[18] F. Besson, T. Jensen, J. P. Talpin, « Timed polyhedra analysis for synchronous languages », *in : Proceedings of the 10th International Conference on Concurrency Theory (CONCUR'99)*, *Lecture Notes in Computer Science*, *1664*, Springer, août 1999.

[19] T. Gautier, P. Le Guernic, « Code generation in the SACRES project », *in : Towards System Safety, Proceedings of the Safety-critical Systems Symposium, SSS'99*, F. Redmill, T. Anderson (éditeurs), Springer, p. 127–149, Huntingdon, UK, février 1999, `http://www.irisa.fr/ep-atr/Publis/Abstract/gautier99a.htm`.

[20] F. Jiménez Fraustro, E. Rutten, « Modélisation synchrone de standards de programmation de systèmes de contrôle », *in : Actes de la* Journée d'études sur les Nouvelles Percées dans les Langages pour l'Automatique, Amiens, novembre 1999. à paraître.

[21] F. Jiménez Fraustro, E. Rutten, « A synchronous model of the PLC programming language ST », *in : Proceedings of the* Work In Progress session, *1st Euromicro Conference on Real Time Systems, ERTS'99*, p. 21–24, York, England, juin 1999.

[22] A. Kountouris, C. Wolinski, « Combining Speculative Execution and Conditional Resource Sharing to Efficiently Schedule Conditional Behaviors », *in : ASP-DAC'99*, Hong Kong, janvier 1999.

[23] A. Kountouris, C. Wolinski, « Hierarchical Conditional Dependency Graphs for Mutual Exclusiveness Identification », *in : 12th International Conference on VLSI Design*, Goa, India, janvier 1999.

[24] A. Kountouris, C. Wolinski, « High-level Pre-synthesis Optimization Steps using Hierarchical Conditional Dependency Graphs », *in : Proceedings of the EUROMICRO'99*, IEEE Computer Society Press, Milan, Italie, août 1999.

[25] J. C. Le Lann, C. Wolinski, « Load Balancing and Functional Unit Assignment in High-Level Synthesis », *in : Proceedings of the SCI'99/ISAS'99*, Orlando, Florida, USA, août 1999.

[26] H. Marchand, M. Samaan, « On the Incremental Design of a Power Transformer Station Controller using Controller Synthesis Methodology », *in : FM'99 — Formal Methods*, J. M. Wing, J. Woodcock, J. Davies (éditeurs), *Lecture Notes in Computer Science*, *1709*, Springer, p. 1605–1624, Toulouse, France, septembre 1999, `http://www.irisa.fr/ep-atr/Publis/Abstract/marchand99a.htm`.

[27] D. Nowak, J. P. Talpin, P. Le Guernic, « Synchronous Structures », *in : Proceedings of the 10th International Conference on Concurrency Theory (CONCUR'99)*, *Lecture Notes in Computer Science*, *1664*, Springer, août 1999, `http://www.irisa.fr/ep-atr/Publis/Abstract/nowak99a.htm`.

[28] I. M. Smarandache, T. Gautier, P. Le Guernic, « Validation of Mixed Signal-Alpha Real-Time Systems through Affine Calculus on Clock Synchronisation Constraints », *in : FM'99 – Formal Methods*, J. M. Wing, J. Woodcock, J. Davies (éditeurs), *Lecture Notes in Computer Science*, *1709*, Springer, p. 1364–1383, Toulouse, France, septembre 1999, `http://www.irisa.fr/ep-atr/Publis/Abstract/smarandache99a.htm`.

Rapports de recherche et publications internes

[29] A. Benveniste, P. Caspi, « Distributing synchronous programs on a *loosely synchronous,* distributed architecture », *rapport de recherche n° 1289*, Irisa, décembre 1999, `http://www.irisa.fr/EXTERNE/bibli/pi/1289/1289.html`.

[30] M. Kerbœuf, D. Nowak, J. P. Talpin, « The steam-boiler problem in Signal–Coq », *rapport de recherche n° 3773*, Irisa / Inria-Rennes, octobre 1999, `http://www.irisa.fr/ep-atr/Publis/Abstract/kerboeuf99a.htm`.

[31] H. Marchand, M. Le Borgne, « The Supervisory Control Problem of Discrete Event Systems using polynomial Methods », *rapport de recherche n° 1271*, Irisa, octobre 1999, `http://www.irisa.fr/ep-atr/Publis/Abstract/marchand99b.htm`.

[32] S. Pinchinat, H. Marchand, M. Le Borgne, « Symbolic Abstractions of Automata and their application to the Supervisory Control Problem », *rapport de recherche n° 1279*, Irisa, novembre 1999, `http://www.irisa.fr/ep-atr/Publis/Abstract/pinchinat99.htm`.

[33] J. P. Talpin, A. Benveniste, B. Caillaud, P. Le Guernic, « Hierarchic Normal Forms for desynchronization », *rapport de recherche n° 1288*, Irisa, décembre 1999, `http://www.irisa.fr/EXTERNE/bibli/pi/1288/1288.html`.

[34] J. P. Talpin, A. Benveniste, P. Le Guernic, « Asynchronous deployment of synchronous transition systems », *rapport de recherche n° 1269*, Irisa, octobre 1999, `http://www.irisa.fr/ep-atr/Publis/Abstract/talpin99a.htm`.