INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Projet ESPRESSO

Environnement de Spécification de Programmes RÉactifS SynchrOnes

Rennes

THÈME 1C

Table des matières

1. Composition de l'équipe

Responsable scientifique
Jean-Pierre Talpin [CR Inria]

Assistante de projet
Maryse Auffray [AA Inria, à partir de novembre 2002]
Huguette Béchu [TR Inria, jusqu'à novembre 2002]

Personnel Inria
Thierry Gautier [CR]
Paul Le Guernic [DR]

Personnel CNRS
Loïc Besnard [IR, Atelier]

Personnel Université de Rennes 1
Bernard Houssais [maître de conférences]

Ingénieurs experts et ingénieurs associés
Bruno Le Dez [ingénieur expert]
Luc-Michel Sévère [ingénieur associé]

Chercheur post-doctorant
Qiuling Pan [bourse Inria, à partir du 1er avril 2002]

Chercheurs doctorants
David Berner [bourse Inria, à partir du 1er novembre 2002]
Abdoulaye Elhadji Gamatié [bourse Inria]
Mickaël Kerbœuf [bourse MENRT jusqu'au 30 septembre 2002, Ater à l'Insa de Rennes à partir du 1er octobre 2002]
Sylvain Kerjean [bourse MENRT jusqu'au 22 février 2002]
Mirabelle Nebut [Ater à l'Université de Rennes 1 jusqu'au 31 août 2002, Ater à l'IUT de La Rochelle à partir du 1er septembre 2002]
Laurent Vibert [normalien]

2. Présentation et objectifs généraux

2.1. Introduction

Mots clés : *Espresso*, *système enfoui*, *système embarqué*, *temps réel*, *conception synchrone*, *composant réactif*.

Le projet Espresso propose des modèles, méthodes et outils permettant la conception de composants logiciels répondant au plus haut niveau de fiabilité pour l'ingénierie des systèmes embarqués. Pour cela, le projet promouvois l'approche polychrone (synchrone multi-horloge) comme favorisant la réutilisation de composants logiciels réactifs et comme permettant de les déployer en temps et coûts optimisés, sur un spectre d'architectures allant des circuits aux systèmes distribués, et dans un cadre mathématique offrant les meilleures garanties de fiabilité.

2.2. Contexte des études

Mots clés : *système enfoui*, *système embarqué*, *système temps réel*, *système critique*, *certification*, *cycle en V*, *approche orientée objets*, *composants*, *architecture hétérogène*, *vérification*, *méthode formelle*.

Une très grande part des systèmes informatiques fabriqués et utilisés dans le monde sont des systèmes embarqués. Un tel constat pourrait à première vue surprendre, mais il suffit de considérer combien les télécommunications et les transports (automobile, avion, ferroviaire) sont présents dans notre vie quotidienne pour s'en convaincre définitivement.

Les systèmes enfouis sont également présents dans de nombreuses applications d'utilisation plus discrète, mais d'une importance tout aussi grande, voire critique vis-à-vis du grand public. Évoquons par exemple les systèmes de contrôle aérien, les systèmes de régulation et d'alarme des centrales nucléaires, les serveurs de télécommunication. Ces systèmes sont de très grandes applications, d'une ingénierie complexe, faisant appel à de multiples compétences, soumis à des contraintes temps réel fortes.

La caractéristique commune à tout système enfoui est de fonctionner en permanence et en constante interaction avec l'environnement : le système capte et analyse un flux d'information continu provenant de l'environnement et décide quasi simultanément de commander une réponse appropriée à ce flux. La panne d'un système enfoui peut avoir des conséquences catastrophiques sur son environnement. Par exemple, le dysfonctionnement d'un système de commande de vol peut entraîner un accident d'avion, celui d'un système de régulation thermique, la fermeture d'une entreprise agro-alimentaire. Afin de se prémunir des conséquences de tels accidents, le développement des applications enfouies est l'objet de procédures plus ou moins lourdes, telles que la certification, visant à éliminer tout risque d'erreur ou de défaillance du système.

Traditionnellement, le développement intégré de ces applications s'appuie sur un cycle en V, de l'amont (la spécification) vers l'aval (la mise en œuvre), puis de la validation des composants à la vérification du système. Dans une approche « orientée objets », ce qui change fondamentalement est la place, certes toujours centrale, des composants réactifs dans l'architecture du système considéré, qui elle devient hétérogène. La méthode de développement est principalement axée sur l'intégration de composants réactifs synchrones avec des composants extérieurs dont il faut valider les propriétés décrites au moyen d'interfaces, puis sur la vérification globale de l'assemblage fonctionnel obtenu sur l'architecture cible considérée pour le déploiement du système.

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 à la vérification complète du 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 des méthodes traditionnelles, y compris dans des documents normatifs.

2.3. Objectif général du projet

Mots clés : *modèles théoriques*, *méthodologie de conception*, *prototypes logiciels*, *applications*, *traitement temps réel du signal*, *télécommunication*, *avionique*, *automobile*, *temps réel*, *conception synchrone*, *composant réactif*, *génération de code enfoui*, *génération de code distribué*, *architecture hétérogène*.

Les activités conduites dans le projet visent à proposer des modèles théoriques et des enrichissements au contexte méthodologique traditionnel 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 d'expérimenter les solutions proposées. Les travaux relatifs à l'approche synchrone à l'Irisa ont débuté dans le domaine du traitement temps réel du signal en télécommunication. Ils ont ensuite trouvé de nouveaux contextes d'application, en particulier par des collaborations suivies avec différents projets de l'Inria :

- en robotique et en image avec EDF ;
- dans le domaine de l'avionique avec nos partenaires des projets Esprit Sacres, Syrf et Safeair ;
- dans le domaine des télécommunications, là encore en collaboration avec des projets de l'Inria, avec Alcatel et le Cnet ;
- dans le domaine de l'automobile, avec les industriels impliqués dans le projet AEE, et là aussi différents projets de l'Inria et d'autres partenaires universitaires.

En même temps que se sont élargis les domaines d'application, 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.

2.4. Conception synchrone

Mots clés : *conception synchrone*, *Signal*, *contraintes*, *non-déterminisme*, *spécification partielle*, *système réactif*, *approche orientée objets*, *BDL.*

Le principe de base du modèle synchrone [27][29][35] est fondée sur l'abstraction du temps : il s'agit de considérer qu'un automatisme réactif, plongé dans un environnement donné, interagit significativement avec cet environnement (au moyen d'un ensemble de supports de communication) et à des instants précis (qui forment un ensemble dénombrable et ordonné). À chaque instant, plusieurs actions (messages reçus, calculés et émis) peuvent être effectuées. Ces événements sont par hypothèse simultanés. Ils possèdent donc un même indice temporel.

La perception synchrone de 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. 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 synchrones Esterel [30], Lustre [34] et SIGNAL [8][2][11][10], soit produites à l'instant suivant, comme par exemple dans la variante Statemate des « Statecharts » [36].

Les langages supports du projet Espresso sont le langage SIGNAL et la notation BDL.

Les langages Lustre et SIGNAL sont construits selon une approche flot de données [28] 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 par une relation des contraintes entre les signaux d'entrée et de sortie du système ; ceci permet d'utiliser ce langage 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 mais aussi « pro-actifs ».

La notation BDL [46] reprend quant à elle l'essentiel du modèle de programmation des systèmes de transition synchrones pour lui donner une représentation graphique dans un style inspiré des diagrammes de séquence et des diagrammes d'état de UML. On retrouve dans la notation BDL la forte potentialité de SIGNAL à décrire des spécifications partielles, mais cette fois dans un style de programmation orienté objet avec une connexion plus immédiate avec les autres notations de UML.

2.5. Thématiques de recherche

Mots clés : *système réactif*, *SIGNAL*, *Polychrony*, *conception synchrone*, *programmation synchrone*, *validation*, *vérification*, *model-checking*, *interprétation abstraite*, *système dynamique polynomial*, *transformation de programme*, *déploiement*, *architecture hétérogène*, *composant synchrone*, *codesign*, *conception conjointe matériel-logiciel*, *architecture GALS*, *modularité*, *relation de synchronisation*, *causalité*, *distribution*, *désynchronisation*, *protocole de synchronisation*, *protocole de communication*, *système reconfigurable*.

Une méthodologie de conception synchrone consiste en la mise en œuvre de transformations successives de spécifications, partant de la description partielle des composants d'un système et de ses interfaces (au moyen de propriétés ou d'abstractions) avec l'environnement (les composants externes), jusqu'à l'assemblage et le déploiement de ces composants. Elle intègre la vérification des propriétés de chaque composant du système, la validation de chacune des interfaces par rapport au comportement de l'environnement, puis l'assemblage et le déploiement des composants du système sur l'architecture cible. On retrouve les différents aspects de cette réflexion dans la thématique de notre projet :

- spécification, conception et vérification modulaire de composants réactifs fiables indépendamment de l'architecture cible, grâce à l'hypothèse synchrone ;
- description et intégration de composants externes au moyen de spécifications partielles ;
- vérification des modules réactifs et validation de leurs interfaces avec le monde extérieur ;
- assemblage de la spécification occasionnant la génération de code cible modulaire ;
- déploiement de la spécification occasionnant la génération de protocoles de synchronisation ;

- modélisation des composantes dynamiques et mobiles d'une architecture distribuée de système réactif, offrant les mêmes garanties de fiabilité aux systèmes reconfigurables qu'aux systèmes temps réels.

Ces différents points suscitent une thématique scientifique orientée sur trois axes interdépendants. Ces thèmes introduisent des sujets d'étude et de recherche qui vont au-delà de l'activité particulière d'une année, mais constituent une déclinaison des objectifs pluri-annuels du projet.

2.5.1. Techniques de vérification et de validation

La vérification de systèmes réactifs conçus selon l'approche synchrone était jusqu'à présent envisagée essentiellement sous l'angle de techniques rigoureuses et éprouvées de « model-checking » [33], mises en œuvre en SIGNAL au moyen de la notion de systèmes dynamiques polynomiaux et de l'outil SIGALI [6]. Cette technique est des plus efficaces dans le cas où le système réactif met en œuvre un automatisme discret (dont l'espace de calcul est borné aux booléens ou à $\mathbb{Z}/n\mathbb{Z}$).

Elle ne l'est plus lorsque l'on considère la vérification des propriétés d'un système opérant sur les entiers ou les réels. Dans ce cas, il faut avoir recours à des techniques d'interprétation abstraite, un cadre général d'analyse de programmes qui consiste à ramener le calcul des propriétés abstraites de systèmes réactifs à des domaines aux objets calculables : ainsi le treillis des polyèdres [31][37] ou l'arithmétique de Presburger [45].

2.5.2. Programmation modulaire de composants synchrones

La mise en œuvre d'une approche orientée comosants pour la conception de logiciels réactifs fiables s'accompagne d'une réflexion sur les notions de théorie des types nécessaires à la réalisation d'un système de modules permettant de rendre compte des différentes propriétés comportementales d'un composant synchrone [43].

2.5.2.1. Programmation modulaire et compilation séparée

Tout d'abord, une réflexion sur la modularité, perçue comme unité de spécification, doit aller de pair avec une autre sur la compilation séparée, afin de définir l'unité d'exécution correspondante. De fait, un module forme une unité de compilation s'il est robuste au déploiement sur une architecture donnée. Plus précisément, un module endochrone - pour lequel il existe un temps local interne - peut être utilisé (ou instancié) sur une architecture (fonctionnelle) donnée s'il est placé dans un contexte où la propriété d'invariance de flots est respectée.

Le schéma général de compilation et d'optimisation de la platef-roem POLYCHRONY consiste à opérer des transformations successives des programmes jusqu'aux étapes ultimes de production de code cible. Cette approche diminue les risques d'incorrection liés, par exemple, aux erreurs de traduction d'un langage à un autre.

Dans le cas où la génération de code intervient dans le cadre de la réalisation d'une application distribuée, ou bien d'une architecture matérielle comprenant plusieurs composants préexistants, elle doit s'accompagner d'une modélisation synchrone des propriétés des modules externes (composants matériels ou composants logiciels).

2.5.2.2. Techniques d'assemblage et de déploiement

Le problème de partitionnement, de distribution ou de déploiement d'une spécification synchrone sur une architecture cible donnée doit être abordé en s'attachant aux propriétés structurelles de l'application et notamment à ses composantes prédéfinies.

Étant donné une répartition ou un déploiement de la spécification sur une architecture cible, la désynchronisation consiste tout d'abord à vérifier que le déploiement considéré peut être effectué en préservant le sens de la spécification synchrone initiale. Cela est réalisé en s'assurant que cette spécification satisfait aux critères de l'endo-isochronie ou d'invariance de flots [16].

L'outil de distribution produit non seulement le code exécutable correspondant à chaque composant de l'architecture, mais synthétise également les protocoles de communication mettant en œuvre les resynchronisations des composants ainsi distribués.

La mise en œuvre distribuée d'une spécification synchrone est fondée sur le principe des architectures GALS (*« Globalement Asynchrone Localement Synchrone »*). Le cadre théorique de nos recherches permet de caractériser ces types de mise en œuvre synchrones partiellement désynchronisées en terme de préservation de propriétés de flots, d'ordres partiels, de taille mémoire, par rapport au programme initial.

2.5.3. Conception conjointe d'architectures matériel-logiciel

La conception et la simulation conjointe d'architectures matériel-logiciel (« codesign ») est devenue une spécialité importante au regard de la complexité toujours plus grande et du renouvellement toujours plus rapide des systèmes embarqués. De nombreux modèles, langages et outils de programmation spécifiques existent aujourd'hui pour répondre à ces besoins.

L'industrie se trouve aujourd'hui face à un parc de composants et de circuits (IP) très important, aux spécifications souvent décrites en VHDL ou Verilog. Aussi, afin de répondre rapidement aux attentes d'un marché toujours plus ouvert des systèmes embarqués, la méthode de développement qui en découle consiste à réutiliser au mieux ce capital en mettant en œuvre l'intégration de ces composants (IP) dans des architectures plus complexes : les systèmes sur puce (SOC ou *« systems on chip »*).

Pour la conception d'un SOC au moyen de composants existants, il est nécessaire de considérer des modèles permettant d'assurer la modélisation, la simulation et la vérification de systèmes en considérant le système à un niveau de description d'un grain plus fort, avec des niveaux d'abstraction plus ou moins fins.

Dans ce but, l'attention de l'industrie se focalise aujourd'hui sur l'utilisation de langages de modélisation de haut niveau d'une syntaxe dérivée de celle de langages de programmation « grand-public » (SYSTEMC, SPECC, etc.). L'objectif est de faciliter l'intégration de composants élémentaires dans une architecture complexe, et ceci afin d'accroître significativement la réutilisabilité (des composants) et la productivité (des concepteurs de systèmes). Plusieurs verrous et défis technologiques doivent être levés afin de remplir correctement les objectifs visés.

2.5.3.1. Co-simulation rapide de systèmes

Le plus important est la co-simulation rapide de systèmes. En effet, la modélisation de haut niveau est d'un grand intérêt en simulation. Elle permet d'accélérer considérablement l'évaluation des performances d'architectures complexes, constituées de multiples composants.

Le modèle formel et les techniques d'analyse et de transformation de programmes de la plate-forme expérimentale POLYCHRONY, qui supporte le langage de programmation SIGNAL (voir section 5.1), offrent les services nécessaires à la réalisation des optimisations permettant d'accélérer notablement la simulation d'architectures matériel-logiciel.

- **Résolution formelle** permettant d'anticiper des résultats dès la compilation (techniques d'évaluation partielle, de *model-checking* abstrait).
- **Techniques d'abstraction** (ou d'interprétation abstraite) permettant de décompiler une description VHDL synthétisable sous la forme d'un code de simulation de haut niveau (en C ou en SYSTEMC) offrant des gains de performance de plusieurs ordres de grandeurs[1]. Afin d'en modéliser et d'en automatiser les concepts, on étudiera :
  - des méthodes d'abstraction de données : simulation des opérations au niveau de vecteurs de bits par des opérations sur des entiers et des opérateurs de masquage ;
  - des méthodes d'abstraction temporelle : prendre un pas ou une horloge de simulation plus grande que celle de l'IP d'origine.

[1] À titre de comparaison, l'expérience montre que l'écriture *à la main* d'un modèle de haut niveau en langage C d'une IP VHDL permet de gagner un facteur 10^6 en vitesse de simulation

2.5.3.2. Test de conformité

La capture de modèles de haut niveau et de niveau comportemental d'IP au moyen de la plate-forme POLYCHRONY offre un support formel pour l'utilisation de techniques d'analyse, de transformation, de synthèse et de vérification permettant de garantir la conformité d'un modèle de haut niveau, destiné à la simulation, par rapport à un modèle comportemental, destiné à la synthèse.

L'utilisation d'une plate-forme de modélisation formelle permet d'assurer aux abstractions et transformations de modèles des garanties de conformité par construction, et notamment :

- **Conformité des modélisations de haut niveau par rapport aux spécifications comportementales de composants.** Des critères spécifiques (l'équivalence d'horloges, l'équivalence de flot) permettent de vérifier qu'un modèle de haut niveau a un comportement equivalent à celui de l'IP qu'il abstrait.
- **Conformité des composants modélisés par rapport au modèle de l'architecture.** L'abstraction d'un composant par un modèle de haut niveau ne doit pas perturber le comportement global du système, c'est-à-dire des autres composants de l'architecture. Pour s'en assurer, un critère d'équivalence de flots permet par exemple d'assurer une équivalence par bisimulation, en assurant que le flot de valeurs du modèle de haut niveau (donc polychrone) de l'architecture est identique à celui de son modèle d'implémentation (globalement asynchrone).

2.5.3.3. Conception orientée composants

Le développement présent des activités citées plus haut nous amène à les placer dans une prospective de long terme qui attaque des verrous technologiques importants en conception conjointe d'architecture matériel/logiciel. La modélisation de formalismes de haut niveau, c'est-à-dire dans ce cas :

- l'abstraction d'un composant (IP) VHDL par une description comportementale abstraite en SIGNAL,
- la traduction et la modélisation de descriptions de haut niveau (SYSTEMC ou SPECC) en SIGNAL,

ouvre la voie à l'utilisation de POLYCHRONY comme une plate-forme sémantique pour la représentation intermédiaire de multiples formalismes et la vérification, la validation d'architectures complexes et hétérogènes.

Pour cela, nous nous proposons de mettre en œuvre des techniques de test de conformité qui consistent à valider la modélisation de haut niveau d'un IP (par exemple en VHDL) par rapport à son simulateur (par exemple en SYSTEMC). Il est nécessaire de comparer les référentiels temporels et les représentations de données de l'IP et de son modèle.

L'utilisation de POLYCHRONY dans ce contexte d'application permet également d'apporter des réponses et des contributions dans d'autres domaines de préoccupation en aide à la conception, tels que la simulation rapide, la réutilisation, la vérification et la synthèse de protocoles.

2.5.3.4. Mise en œuvre

Afin de répondre à ces objectifs, nous proposons l'utilisation de la plate-forme POLYCHRONY, supportant le langage SIGNAL, afin de mettre en œuvre les méthodes formelles permettant la modélisation de SOC.

Cette modélisation consiste, d'une part, à permettre l'abstraction d'IPs existantes en vue de simulation rapide et, d'autre part, à vérifier la correction de leur assemblage, tel qu'il peut être vu au travers d'une description de haut niveau SYSTEMC, par exemple.

L'utilisation de la plate-forme POLYCHRONY permet l'accès à des outils de vérification et d'optimisation qui assurent le raffinement et la transformation de modèles non régressifs, c'est-à-dire préservant le sens des descriptions initialement données au regard d'un critère particulier (par exemple, l'équivalence de flots).

2.6. Prospective scientifique

Mots clés : *Polychrony*, *synchrone/asynchrone*, *spécification/mise en œuvre*, *modèle polychrone*, *raffinement*, *conception par composants*, *architecture GALS*, *méthodes formelles*, *programmation objet*, *héritage*, *Signal*, *Java*, *SystemC*.

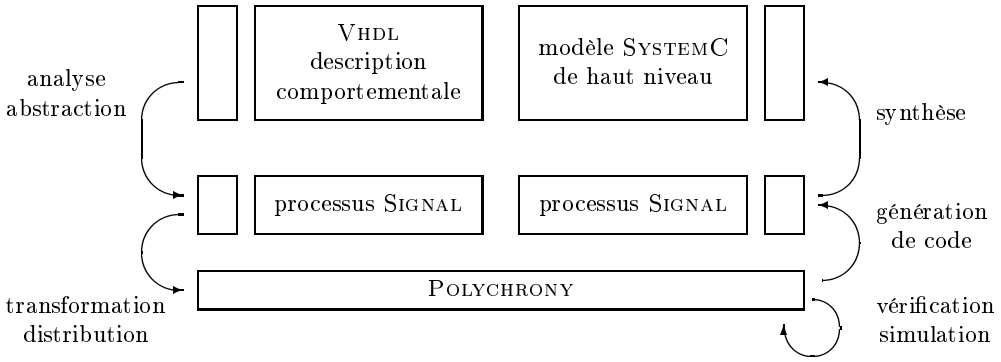

Figure 1.

2.6.1. Modélisation de systèmes et d'architectures embarquées

La plate-forme POLYCHRONY amtérialise les études et les réalisation du projet en offrant des services d'aide à la conception, basés sur un modèle relationnel de la concurrence, dit *polychrone* (i.e. à horloges multiples), formant un continuum de la spécification (relation) à la mise en œuvre (algorithme), un continuum du synchrone (le circuit) à l'asynchrone (le réseau).

Le projet Espresso participe au déploiment de cette technologie en tant qu'outil de conception et de programmation de haut niveau, au moyen du langage de programmation SIGNAL, mais aussi comme plate-forme ouverte d'aide à la conception vers des formalismes de haut niveau hétérogènes, afin de diffuser les services offerts par les méthodes formelles qu'il met en œuvre.

Ces développements se traduisent par la promotion de méthodologies de conception par raffinement, c'est-à-dire l'élaboration de systèmes de haut (spécification) en bas (implémentation) et par composants, c'est-à-dire de bas (composants) en haut (architecture). Ces deux méthodes trouvent une parfaite synergie dans la plate-forme POLYCHRONY.

Les prospectives du projet dans l'évolution de la plate-forme POLYCHRONY portent d'une part sur l'évolution du langage de programmation pour qu'il automatise des services plus généraux dans un flot de conception par raffinement, mais aussi de l'algorithmique de la plate-forme POLYCHRONY proprement dite, pour qu'elle facilite la conception par composants d'architectures GALS.

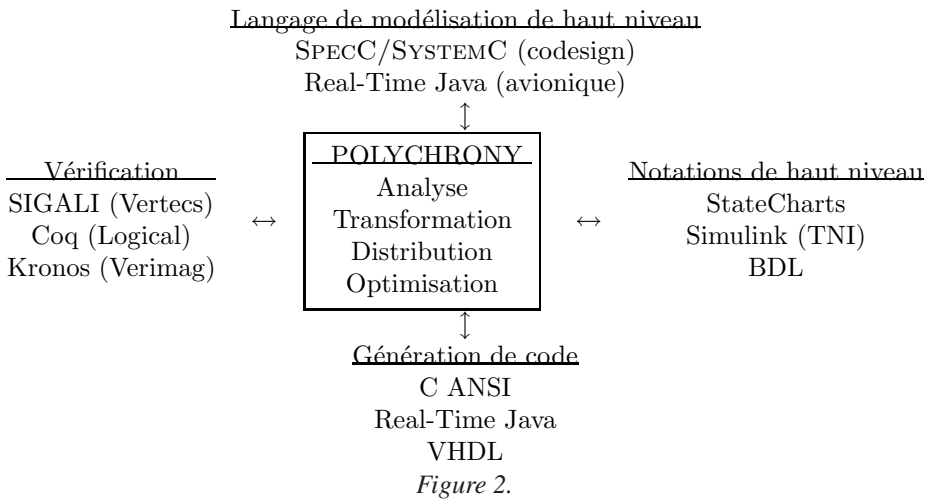

Figure 2.

L'évolution du langage de programmation SIGNAL porte sur l'automatisation de techniques formelles de synthèse et de vérification de raffinements de spécification. Pour cela, nous étudions le concept de co-optimisation, c'est-à-dire de compilation et de vérification conjointe, permettant de généraliser la synthèse du contrôle de systèmes embarqués non plus seulement à des invariants (des équations d'horloges), résolus au moyen de l'algorithmique de hiérarchisation de SIGNAL, mais également à des problèmes dynamiques (par exemple des équations d'horloges dépendantes de l'état du système) en faisant appel à des techniques d'interprétation abstraite et de *model-checking* borné.

Un autre aspect de l'évolution de SIGNAL porte sur la réflexion de la méthodologie de conception par raffinement au moyen de concepts syntaxiques de programmation objet. Ces travaux ont été développés par dans la thèse de Mickaël Kerbœuf [12], et vont progressivement être intégrés dans une réflexion de moyen terme sur l'évolution de SIGNAL.

L'évolution de la plate-forme POLYCHRONY et de son outillage algorithmique est en parfaite synergie avec cette réflexion. Elle porte sur :

- La réalisation d'algorithmes outillant les critères de validation et d »automatisation du concept de raffinement. Ces algorithmes permettent de comparer une spécification synchrone et sa mise en œuvre distribuée au moyen des notions d'invariance de flot et d'endo-isochronie [16].
- Des critères permettant de comparer des modèles possédant des échelles de temps et de représentation de données différentes (par exemple, pour comparer une addition synchrone $x = y + z$ et un circuit additionneur, au comportement plus complexe et aux structures de données plus concrètes).
- Le développement de la notion de morphisme pour procéder à l'abstraction de spécifications formelles sur des algèbres (e.g. max+) permettant de mesurer le comportement de spécifications SIGNAL sur différentes échelles.
- L'automatisation des techniques de raffinement par la réalisation d'algorithmes permettant la synthèse de protocoles et d'interfaces entre composants synchrones.
- Des optimisations spécifiques à des domaines d'application, et notamment la synthèse optimisée de circuits (thèse de Jean-Christophe Le Lann). [13].

L'ouverture de la plate-forme POLYCHRONY vers des formalismes amont et des outils spécifiques de vérification et de validation nous amène enfin à développer une interface robuste et une représentation intermédiaire unifiée pour l'échange d'information avec des outils de traduction (Java-SIGNAL, SYSTEMC-SIGNAL) et de vérification formelle (*model-checking* symbolique avec notamment SIGALI, *model-checking* temporisé avec Kronos, preuve de théorèmes avec Coq).

3. Fondements scientifiques

3.1. Spécification et programmation synchrone

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

Nous définissons la sémantique relationnelle 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 [27][28][29][35]. 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 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 relationnelle 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 [2][10].

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 $T : \mathbf{N} \longrightarrow A_1 \longrightarrow (D \cup \{\perp\})$.

Pour tout $k \in \mathbf{N}$, 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 donnés 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 $1_{\mathcal{P}}$ le processus ayant comme seul élément la trace (unique) sur l'ensemble vide de variables. On montre alors que $(\mathcal{P}_A, 1_{\mathcal{P}}, |)$ est un monoïde commutatif (cette propriété rend possible la transformation de programmes mentionnée en 3.1.2) :

$$\begin{gathered}(P_1|P_2)|P_3 = P_1|(P_2|P_3)\\ P_1|P_2 = P_2|P_1\\ P|1_{\mathcal{P}} = P\end{gathered}$$

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

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

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 [11] 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 terme 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 :

$$\text{Y := f(X1,...,Xn)} \quad : \begin{cases} \forall k, \mathtt{Y}_k = \perp \implies \mathtt{X1}_k = \ldots = \mathtt{Xn}_k = \perp \\ \forall k, \mathtt{Y}_k \neq \perp \implies \mathtt{Y}_k = \mathtt{f}(\mathtt{X1}_k, \ldots, \mathtt{Xn}_k) \end{cases}$$

- Retard (registre à décalage) :

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

- Extraction sur condition booléenne :

$$\texttt{Y := X when B} \quad : \forall k, \begin{cases} \texttt{B}_k \neq true \implies \texttt{Y}_k = \bot \\ \texttt{B}_k = true \implies \texttt{Y}_k = \texttt{X}_k \end{cases}$$

- Mélange avec priorité :

$$\texttt{Y := U default V} \quad : \forall k, \begin{cases} \texttt{U}_k \neq \bot \implies \texttt{Y}_k = \texttt{U}_k \\ \texttt{U}_k = \bot \implies \texttt{Y}_k = \texttt{V}_k \end{cases}$$

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 $\bot$). Par exemple, les deux premières primitives sont monocadencées : elles imposent que tous les signaux impliqués aient la même horloge. En revanche, dans les troisième et 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* [8] 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

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

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) : $Q(P)$ est une abstraction conservative.

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 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 [9] 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

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

- 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 $H \longrightarrow K$. 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 [8]. 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 [32]).

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 proposés par Manna et Pnueli [40][41], 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 - les nœuds ont trois valeurs possibles), 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.

4. Domaines d'application

4.1. Introduction

Mots clés : *conception synchrone*, *télécommunication*, *traitement temps réel du signal*, *énergie*, *transport*, *avionique*, *automobile*, *système embarqué*, *architecture matériel-logiciel*.

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 les membres du projet s'impliquent depuis longtemps dans des travaux liés aux télécommunications. Une longue coopération avec EDF a donné lieu à 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 des projets Sacres et SafeAir, les applications considérées relèvent du domaine de l'avionique. Enfin, nous nous intéressons de plus en plus à la conception conjointe d'architectures matériel/logiciel.

4.2. Télécommunications

Participants : Paul Le Guernic, Jean-Pierre Talpin.

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

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 Alpha, ainsi que dans de nouvelles collaborations dans lesquelles l'utilisation mixte des modèles synchrone et asynchrone est étudiée.

Le changement rapide de l'industrie des télécommunications a eu un impact fort sur le développement du logiciel dans ce domaine. Il semble que le génie logiciel pour les télécommunications ne puisse plus être orienté vers le développement de grands systèmes dont on maîtrise toute la conception. Le traditionnel « cycle en V » n'est plus pertinent. Les besoins s'expriment maintenant de plus en plus fortement en termes de méthodes permettant la conception, l'intégration et le test de composants. Ces composants doivent s'assembler

progressivement dans un environnement formé d'autres composants que l'on ne maîtrise pas entièrement (notamment du point de vue de la fiabilité). Plusieurs pistes de recherche ont été identifiées :

- Composants et intégration logicielle - la construction de services complexes de télécommunications (intégrant des aspects très différents comme l'Internet, la voix, des bases de données) pousse à intégrer des composants logiciels par des interfaces bien plus générales que les traditionnels protocoles. On ne maîtrise pas suffisamment les modèles d'architectures et de composants, l'équipement des composants avec les moyens de prouver (ou tester) leur intégration fiable, la gestion des configurations dynamiques et réparties.
- La validation robuste - on sait que la validation d'un ensemble de composants requiert des outils variés (preuve, vérification, simulation, test, supervision). Au delà de leur utilisation conjointe se pose le problème de les faire travailler simultanément sur des objets à plusieurs niveaux de développement.
- L'analyse de fiabilité - le génie logiciel est nettement en retard par rapport à d'autres types d'ingénierie. L'objectif ici est de pouvoir quantifier (voire surveiller) la fiabilité d'un assemblage de composants pour amener enfin l'industrie du logiciel à pouvoir offrir des garanties.

4.3. Avionique

Participants : Loïc Besnard, Abdoulaye Elhadji Gamatié, Thierry Gautier, Paul Le Guernic.

Mots clés : *conception synchrone*, *Signal*, *Arinc*.

Nos activités dans le domaine de l'avionique se sont développées depuis plusieurs années dans le cadre de projets européens, notamment Sacres et SafeAir. Elles concernent tous les aspects du processus de développement, et en particulier la génération de code sur une architecture temps réel.

Le domaine de l'avionique est clairement un domaine critique pour les systèmes enfouis, où les industrqies de pointe sont confrontées à une exigence de sécurité maximale. L'industrie européenne est particulièrement bien placée dans ce secteur et doit conserver et améliorer ses atouts pour répondre à la forte augmentation actuelle en fonctionnalité et en complexité. L'avionique est donc un domaine d'application privilégié pour les méthodes formelles en général et spécialement la technologie synchrone.

Dans le cadre de plusieurs projets européens successifs, nous avons été amenés à collaborer étroitement avec des industriels clés du domaine. Il s'agit des projets Synchron, Sacres (voir http://www.tni.fr/sacres), Syrf, puis SafeAir (voir section 7.5). D'autres programmes de collaboration se mettent actuellement en place dans le cadre du réseau Artist. La conception de systèmes avioniques s'appuie d'ores et déjà sur un processus de développement suivant le « cycle en V », sur l'emploi de technologies innovantes (utilisation des outils Statemate, Scade, Simulink, par exemple), sur l'utilisation de librairies temps réel standards (ARINC), sur le respect de normes de certification (DO-178B). L'objectif d'un projet comme SafeAir, qui se base sur l'utilisation de ces technologies, est de permettre de réduire de 35 à 40% l'effort de développement des systèmes logiciels aéronautiques, tout en augmentant leur fiabilité : passage du « cycle en V » vers un « cycle en Y », en utilisant des méthodes de conception de haut niveau et des outils de génération automatique de code et de vérification formelle.

4.4. Des systèmes embarqués aux architectures matérielles

Participants : Paul Le Guernic, Jean-Pierre Talpin.

Mots clés : *architecture matériel-logiciel*, *codesign*.

Nous avons mis en avant depuis longtemps les possibilités offertes par l'approche synchrone dans la modélisation de circuits numériques, l'élaboration de simulateurs et la synthèse de matériel. Nos activités dans le domaine de la conception conjointe matériel/logiciel prennent aujourd'hui une importance croissante dans le projet.

Aujourd'hui, un processeur haute-performance compte environ 100 millions de transistors et sa fréquence est de l'ordre du Giga-Hertz. Il est prévu que ce nombre pourrait atteindre 1 milliard de transistors en 2012 pour une fréquence de 10 Giga-Hertz. Aussi, la recherche sur le thème des architectures de systèmes embarqués s'inscrit dans un contexte d'évolution technologique extrêmement rapide tandis que ses domaines d'utilisation explosent (cartes à puces, téléphones mobiles, automobile, multimédia). Le domaine des SoC propose de nombreux défis scientifiques. D'abord, compte tenu des performances visées, la conception du matériel et celle des logiciels sont de plus en plus interdépendantes (apparaît la notion de conception conjointe). La recherche doit s'orienter vers la découverte de techniques modulaires, capables de traiter de manière intégrée les différents problèmes que posent la conception de tels systèmes. Dans le cas des systèmes critiques (avionique, automobile), les aspects temps réel et sûreté de fonctionnement sont primordiaux voire d'importance vitale à tout point de vue. La recherche de techniques permettant de garantir les contraintes associées à ces systèmes, tout en permettant une réutilisation des composants, constitue également un axe de réflexion très intéressant.

5. Logiciels

5.1. Plate-forme Polychrony pour Signal

Mots clés : *Polychrony*, *Signal*, *programmation synchrone*, *transformation de programme*, *Sildex*.

Contact : Loïc Besnard.

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

1. il nous permet d'étudier des extensions sémantiques ou algorithmiques du modèle synchrone ;
2. il nous permet de mieux comprendre les applications et de dégager ainsi des problématiques nouvelles ;
3. il est diffusé à des fins d'expérimentation et d'enseignement dans des laboratoires pour lesquels la version commerciale Sildex ne convient pas.

La plate-forme POLYCHRONY se compose d'un environnement de compilation de programmes SIGNAL et d'un éditeur graphique. L'outil de vérification SIGALI peut également être fourni dans la plate-forme. L'éditeur de SIGNAL permet à l'utilisateur de construire ses programmes sous forme textuelle et graphique. Il est un vecteur majeur pour la diffusion de l'environnement. L'environnement de compilation de programmes SIGNAL est quant à lui un outil interactif de conception d'applications.

SIGALI est décrit dans la section 5.2.

L'environnement POLYCHRONY peut être vu comme fournisseur d'un ensemble de services. Un service peut être ou non appliqué selon l'objectif : simulation, vérification formelle, compilation séparée, génération de code distribué, etc. Ces fonctionnalités sont accessibles au moyen d'options pour le compilateur et de manière interactive sous l'éditeur graphique.

La version actuellement diffusée de SIGNAL s'est progressivement enrichie dans le sens d'une meilleure expressivité. Ainsi, on y trouve maintenant certaines caractéristiques des Statecharts (les variables d'état) mais aussi des constructions plus déclaratives (à la BDL) comme les définitions partielles.

La plate-forme POLYCHRONY sera bientôt diffusée en logiciel libre.

5.1.1. Environnement de compilation

L'environnement de compilation de programmes SIGNAL est un outil interactif d'aide à la conception d'applications. L'architecture de l'environnement peut être vue comme un ensemble de services. Un service peut être ou non appliqué selon l'objectif : simulation, vérification formelle, compilation séparée, génération de code distribué, etc. Ces fonctionnalités sont accessibles au moyen d'options pour le compilateur « batch » et de manière interactive sous l'éditeur graphique.

Un programme SIGNAL est représenté de façon interne par un GHDC (graphe hiérarchisé aux dépendances conditionnées) qui constitue donc la structure principale de l'environnement.

On peut distinguer :

- un ensemble de fonctionnalités qui produisent un graphe hiérarchisé à partir d'un source SIGNAL ;
- un ensemble de fonctionnalités de transformation du graphe hiérarchisé, transformations qui restituent un graphe hiérarchisé ; ceci constitue le cœur du compilateur ;
- un ensemble de traitements qui produisent les sources d'autres outils.

5.1.1.1. Production du graphe.

L'ensemble des fonctionnalités pour la production du GHDC est constitué :

- **de l'analyse syntaxique et contextuelle,** qui fournit la représentation interne d'un programme source SIGNAL 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, etc.) n'est effectuée.

5.1.1.2. Transformations du graphe.

L'ensemble de traitements qui transforment le graphe hiérarchisé est constitué des étapes suivantes :

- **La compilation,** dont le rôle principal est de trianguliariser le système d'équations d'horloges et de détecter la présence de cycles de dépendance 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. Cette représentation sous forme de forêt d'arbres d'horloges (événements) correspond au format interne dit DC+ [49]. 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.
- **Les transformations inter-formats** définies dans le cadre du projet Sacres. Ces transformations, décrites d'abord sur le format DC+, sur les différents niveaux de sous-formats ayant été identifiés, s'appliquent sur la forme interne d'un programme SIGNAL (GHDC). Les différents sous-formats caractérisent une forme particulière du GHDC.
  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 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 sbDC+ (pour « sequentialized boolean DC+ ») est le format d'entrée effectif des générateurs de code. Il est produit pour les programmes (bDC+) sans cycles et sans contraintes. Un code sbDC+ est une liste de nœuds ordonnés selon les dépendances implicites et explicites du programme SIGNAL. Ceci permet d'écrire de nouveaux générateurs de code sans avoir à parcourir le graphe du programme.

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+ $\longrightarrow$ bDC+, bDC+ $\longrightarrow$ sbDC+, bDC+ $\longrightarrow$ STS, sont des fonctionnalités de l'environnement. Elles sont appliquées selon l'objectif de la compilation.

- **L'application du principe de substitution du langage.** On peut citer :
  - la transformation des booléens (opérateur logiques et relationnels) en événements ; cette opération peut être utile au calcul d'horloges afin de prouver des équivalences ;
  - 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.
- **Des opérations de partitionnement de graphe,** qui produisent un graphe constitué de nœuds représentant eux-mêmes des graphes. On peut citer :
  - la séparation contrôle/calculs, qui consiste à séparer la partie contrôle de l'application de la partie calcul.
  - la séparation état/partie combinatoire, qui consiste à séparer la partie état du programme du reste de l'application.
  - 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.

5.1.1.3. Production de code

Cet ensemble est constitué :

- **de la génération de code séquentiel,** qui passe par un tri topologique du graphe et qui produit du code C, C++, ou Java ;
- **de la restitution de source** SIGNAL**,** 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 une connexion avec l'outil Syndex est réalisée. L'outil Syndex (Y. Sorel, projet Ostre à Rocquencourt) permet d'effectuer une implantation optimisée sous contraintes temps réel sur une architecture multi-processeur.

5.1.2. Diffusion du logiciel

La version commerciale de SIGNAL est vendue par TNI sous la forme de l'environnement Sildex. La version Inria de SIGNAL, qui jusqu'à présent pouvait être obtenue dans le cadre d'une convention de mise à disposition gratuite signée pour un an renouvelable, sera bientôt accessible, en accès libre, sur le site Web du projet Espresso, dans le cadre de la plate-forme POLYCHRONY (voir section 6.1). POLYCHRONY est un logiciel déposé à l'APP sous le numéro IDDN.FR.001.160003.S.P.1996.000.10400.

Notre objectif est de fournir à court terme une distribution sous une licence de type logiciel libre.

5.2. Sigali

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

Contacts : Loïc Besnard, Hervé Marchand (projet Vertecs).

SIGALI est un système de calcul formel permettant la vérification de propriétés de programmes SIGNAL.

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 [8][9] 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. SIGALI peut être fourni dans la plate-forme POLYCHRONY, ou indépendamment de celle-ci (projet Vertecs).

6. Résultats nouveaux

6.1. Signal et l'environnement Polychrony

Mots clés : *programmation synchrone*, *Signal*, *Polychrony*, *transformation de programme*, *morphisme de programmes*, *évaluation de performance*, *module*.

La version actuellement diffusée de SIGNAL a été initialement 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.

6.1.1. Définition et évolution du langage

Participants : Loïc Besnard, Thierry Gautier, Bernard Houssais, Paul Le Guernic, Jean-Pierre Talpin.

L'essentiel du travail sur la définition du langage effectué cette année a eu comme objectif principal la mise à disposition effective de l'environnement POLYCHRONY avec une version de SIGNAL définie de manière précise et répondant aux principaux besoins des utilisateurs.

Cela nous a conduits, de façon pragmatique, à redonner par exemple une syntaxe des itérations de processus proche de ce qui avait existé dans la version H2 de SIGNAL, mais dans un cadre sémantique mieux formalisé. En ce qui concerne les modules, nous en avons défini une forme simple permettant de répondre aux besoins les plus courants. On obtient de cette manière, à travers les niveaux d'importation, l'équivalent des spécialisations des langages à objets. De même, l'équivalent des « virtuels » est obtenu au moyen des « externes ».

Le langage complet est maintenant décrit dans un manuel de référence [22], disponible avec l'environnement POLYCHRONY. Ce manuel décrit la syntaxe et la sémantique du langage. En particulier, une nouvelle formulation de la sémantique de traces a été utilisée. Elle inclut la définition de différentes classes sémantiques de processus : itérations de fonctions, processus endochrones, processus déterministes, processus réactifs. En parallèle avec le modèle des traces marquées des processus polychrones [16] (qui

s'exprime de manière un peu différente), le modèle utilisé introduit aussi les notions de composition asynchrone de processus et d'invariance de flots (propriété qui caractérise les processus pour lesquels la composition asynchrone est un raffinement valide de la composition synchrone). Enfin, il définit une relation d'implémentation entre deux processus.

Une nouvelle version d'un manuel d'initiation à la programmation en SIGNAL a également été rédigée [23], et est aussi fournie avec POLYCHRONY.

6.1.2. Méthodologie de programmation et modules

Participants : Jean-Pierre Talpin, Laurent Vibert.

En prolongement du modèle de concurrence polychrone présenté dans [16], nous nous intéressons au développement d'un système de modules dans lequel les interfaces et les implémentations partagent une même sémantique dénotationnelle, ceci afin d'étendre les travaux présentés dans [42].

Dans le cadre d'une programmation modulaire et de techniques de programmation orientées composants, nous nous intéressons à l'utilisation de systèmes de types comportementaux suffisamment précis. Un tel système permettrait en effet d'abstraire les processus de manière suffisante pour pouvoir simuler une architecture importante tout en évitant une explosion combinatoire.

6.1.3. Plate-forme Polychrony

Participants : Loïc Besnard, Thierry Gautier, Paul Le Guernic.

Nous avons désormais orienté le développement de l'environnement POLYCHRONY dans l'objectif d'une diffusion en tant que logiciel libre.

La ligne directrice que nous avons suivie en vue d'une telle diffusion est de considérer cet environnement comme fournissant un ensemble de fonctionnalités pouvant être appelées automatiquement ou à la demande, pour répondre à un besoin spécifique. POLYCHRONY peut ainsi fournir une boîte à outils adaptable aux besoins particuliers d'un utilisateur.

Cela nous a conduits à un travail important de restructuration de l'environnement.

Pour répondre aux besoins exprimés dans certains domaines d'utilisation (par exemple, le traitement d'images dans le cadre du projet Acotris), nous avons aussi élargi le spectre des programmes compilables en mettant en œuvre une forme de tableaux de processus définis dans le manuel de référence de SIGNAL.

Nous avons également poursuivi, en particulier pour les besoins du projet SafeAir (voir section 7.5), la mise en œuvre de morphismes de programmes SIGNAL, utilisés pour l'évaluation de performances. Le prototype a été testé et utilisé pour des démonstrations (par exemple pour le comité d'évaluation de l'Irisa). Nous avons ainsi illustré sur un exemple l'application de nombreuses fonctionnalités de l'environnement POLYCHRONY :

- Placement d'une application P sur une architecture multi-processeur. Le programme d'origine se récrit alors en la composition de constituants.
- Compilation globale de l'application.
- Partitionnement du graphe selon la description d'architecture.
- Synthèse des interfaces pour chacun des constituants. Ceci induit l'ajout de booléens d'entrée/sortie qui assurent l'endochronie de chacun d'eux.
- Production pour chacun des composants P_i obtenus à l'étape précédente du programme « image temporelle » $M(P_i)$.
- Production des simulateurs de chacun des composants par compilation de la composition de P_i et $M(P_i)$. Ces simulateurs sont obtenus par production des lignées (regroupement dans le même sous-graphe des nœuds qui dépendent du même sous-ensemble d'entrées de l'application).
- Génération de code en mode compilation séparée avec production des abstractions « boîtes grises » (correspondant à la structuration en lignées).
- Production de l'exécutable final par compilation des abstractions produites à l'étape précédente.

6.1.4. Nouveaux générateurs de code

Participants : Loïc Besnard, Thierry Gautier, Qiuling Pan, Luc-Michel Sévère.

De nouveaux générateurs de code ont été développés ou sont en cours de développement dans POLYCHRONY. Il s'agit de générateurs de code Java, Syndex et VHDL. Le générateur de code Java est construit par une spécialisation du code du générateur de code C++. On dispose maintenant de trois modes de génération de code Java :

- « Java standard »,
- « Java Cluster » qui prend en compte les *clusters* (lignées) en découpant le code,
- « Java T » (comme *Threads*) qui génère un *thread* par *cluster* plus un ordonnanceur.

Nous avons aussi étendu la bibliothèque graphique SIGNAL pour la simulation. On peut donc avoir un module de simulation entièrement en Java ; le code Java à engendrer est associé en tant que pragma au processus SIGNAL. L'interface graphique de l'application est générée automatiquement. Cette bibliothèque est documentée et intégrée à POLYCHRONY. Elle a été expérimentée sur un exemple du projet Acotris.

Les générateurs de code C, C++ et Java produisent « à la volée » le texte du langage cible par parcours du graphe du programme SIGNAL. Cette technique impose de prévoir a priori tous les parcours de structures internes puisqu'il n'est pas possible de modifier le contenu du fichier texte déjà engendré.

Pour les générateurs de code Syndex et VHDL, nous avons opté pour une solution par représentation intermédiaire, puis une décompilation produit le texte du langage cible. On peut décomposer le traitement en une fonctionnalité qui parcourt le graphe du programme, et fait appel à une bibliothèque de construction d'arbres abstraits (spécialisée pour chacun des langages cibles). Cette bibliothèque est construite sur une machine générale de traitement d'arbres existant dans POLYCHRONY et déjà utilisée pour l'analyse syntaxique de programmes SIGNAL.

6.2. Analyse de réactions synchrones

Participants : Paul Le Guernic, Mirabelle Nebut.

Mots clés : *modèle synchrone*, *spécification*, *horloge*, *synchronisation*, *absence*, *méthodes formelles*.

Nous avons défini un langage appelé $\mathcal{CL}$ pour la description des propriétés « instantanées » des processus flot de données, c'est-à-dire des propriétés de sûreté ne faisant référence qu'à un seul instant quantifié universellement. Ces propriétés comportementales sont vérifiées sur une abstraction statique des programmes. Nous donnons une procédure de décision pour $\mathcal{CL}$ ainsi qu'une traduction de SIGNAL en $\mathcal{CL}$, permettant de ramener le *model-checking* des programmes à la procédure de décision.

La validation formelle des systèmes est un domaine de recherche en pleine expansion, spécialement en ce qui concerne les systèmes à espace d'états infini et les aspects numériques. L'application de ces méthodes aux programmes synchrones est fondamentale, mais plus ou moins directe. C'est vrai en particulier dans le cas des modèles SIGNAL servant à la vérification : l'absence des variables est souvent explicitée par une valeur spéciale, ce qui empêche l'interfaçage direct des outils aux techniques existantes.

Dans le cadre de la thèse de Mirabelle Nebut [14], co-encadrée par Sophie Pinchinat (projet S4) et Paul Le Guernic, nous nous sommes intéressés à la manipulation de l'absence des variables synchrones, plus particulièrement dans les langages flot de données. Nous avons proposé un langage d'horloge appelé $\mathcal{CL}$, basé sur une extension par des aspects numériques des horloges booléennes utilisées dans le contexte de la compilation SIGNAL. Nous avons montré que $\mathcal{CL}$ permet d'interfacer naturellement les modèles étendus avec l'absence explicite à des théories standards, de plusieurs manières.

L'expressivité de $\mathcal{CL}$, clairement cernée, est en un sens maximale, puisque les formules du langage permettent d'exprimer toute assertion combinatoire, aspects numériques compris (étiquettes d'un système de transition, propriété de sûreté sans état, partie combinatoire d'un système). Grâce à son opérateur de négation, $\mathcal{CL}$ peut aussi exprimer un problème de « model-checking combinatoire ». $\mathcal{CL}$ joue naturellement le rôle de pivot entre formalismes.

Ce travail ouvre de nombreuses perspectives algorithmiques et d'implantation. La plus intéressante est sans doute de reprendre après [31] l'étude d'un outil de vérification pour spécifications polychrones numériques. D'une part on traitera cette fois l'absence par le biais de $\mathcal{CL}$; d'autre part on essaiera de tirer parti du travail déjà effectué par le compilateur pour minimiser la résolution de la partie combinatoire du système.

6.3. Orientation objet

Participants : Thierry Gautier, Michaël Kerbœuf, Jean-Pierre Talpin.

Mots clés : *programmation objet*, *héritage*, UML.

Nous nous intéressons à la conciliation des paradigmes synchrone et objet. La question posée concerne plus spécifiquement l'héritage comportemental. Nous introduisons un formalisme fondé sur le langage SIGNAL, et muni d'un opérateur de choix non-déterministe. Nous étudions l'orientation objet de ce formalisme en introduisant les notions d'encapsulation et d'héritage comportemental par renommage. Cette nouvelle approche permet de conserver le bénéfice du paradigme synchrone tout en raffinant, réutilisant et modifiant des composants grâce au pouvoir de l'héritage. La notion centrale de renommage présentée ici est une version spatiale de l'itération temporelle de l'algorithme classique de recherche des méthodes. La notion d'horloge de méthode permet de propager les informations de synchronisation et ainsi d'échapper à de nombreux cas d'anomalies de l'héritage.

Nous étudions par ailleurs une connexion du langage SIGNAL à un modèle UML.

6.3.1. Objets synchrones

Le SIGNAL-calcul [12] est un formalisme qui permet de spécifier de manière très basique un comportement synchrone. Il est similaire au langage noyau de SIGNAL. Cependant, les équations de base sont des équations *partielles* (comme définies par le connecteur : := de SIGNAL) et un opérateur de choix non-déterministe est introduit.

Sur un plan opérationnel, il y a une différence fondamentale avec SIGNAL : un processus primitif ne peut évoluer que dans un environnement qui définit des valeurs pour tous ses propres signaux, *et uniquement* pour ces signaux. On traite ainsi implicitement l'absence d'un signal par sa non-présence dans l'environnement, et la présence d'un signal non utilisé dans une équation interdit son activation. Ainsi, à un moment donné, le choix n'est indéterministe que si les alternatives peuvent réagir dans un même environnement. De cette manière, on déporte le traitement de l'absence au niveau des alternatives d'un choix, et non pas au niveau des équations. Ce mécanisme permet d'allier facilement préemption et encapsulation.

L'orientation objet du modèle repose sur la notion centrale d'héritage comportemental par renommage. On introduit dans un premier temps des notions d'encapsulation, d'abstraction de comportements (les classes), et de nommage de classes. Les objets sont classiquement des instances de classes. Ils correspondent à des versions encapsulées des processus du calcul initial. Le principe d'héritage s'appuie sur une analogie avec l'algorithme classique de recherche de méthode dans les langages objets (*method lookup semantics* à la Smalltalk), mais avec une version *spatiale* de l'itération temporelle de l'algorithme initial. Ce mécanisme permet de modifier dynamiquement la structure d'un objet afin d'y faire co-exister les méthodes (ou signaux définis) initiales, les méthodes ajoutées, et les méthodes récrites (auxquelles on accède classiquement par une pseudo-variable `super`). Cette réorganisation de l'objet permet l'appel *synchrone* d'une méthode, en d'autres termes l'échange d'un message en un temps nul. Dans l'état actuel des travaux, nous nous intéressons à des objets statiques de premier ordre. Puisque l'échange d'objet n'est pas possible, la topologie des interactions est connue avant l'exécution. Le schéma de renommage appliqué localement et dynamiquement à la création de l'instance d'une classe modifiée par héritage peut donc être appliqué globalement et statiquement à l'ensemble du système.

6.3.2. Connexion Signal-Uml

L'application de méthodes objets pour la conception de logiciels s'inscrit de plus en plus souvent dans le cadre d'un modèle UML (*Unified Modeling Language*). Les technologies synchrones ont dans un tel cadre une place qu'il convient de préciser, tant sur le plan théorique qu'en ce qui concerne les aspects pragmatiques.

Faisant suite aux études réalisées précédemment autour du modèle BDL, et notamment sur la traduction des Statecharts de UML en BDL [47] (ainsi que sur la traduction des Statecharts de Statemate en SIGNAL [26]), nous avons, avec nos partenaires du projet Acotris, défini et expérimenté un modèle de traduction d'une application spécifiée en utilisant la méthodologie Accord/UML [38] conçue au CEA-List, vers un modèle SIGNAL. L'un des objectifs est de pouvoir bénéficier ainsi des capacités de génération et de vérification formelle offertes dans le modèle synchrone. Nous avons notamment défini une traduction de SIGNAL vers l'outil Syndex, logiciel de CAO niveau système pour l'aide à l'implantation de systèmes distribués temps réel embarqués [21] (voir section 6.1).

Une première traduction des modèles Accord/UML vers SIGNAL s'appuie sur une sémantique synchrone des systèmes à états-transitions, et est donc non conforme à la sémantique d'exécution asynchrone définie dans UML (laquelle considère que les événements sont gérés dans des files d'attente). Elle offre cependant de manière immédiate un accès non dégradé à la technologie synchrone. Il n'est pas inutile alors de s'interroger sur la possibilité de proposer une sémantique synchrone des Statecharts de UML.

Une autre traduction respecte le standard UML et représente alors le modèle d'exécution Accord/UML en SIGNAL. Ceci est possible dans la mesure où on se limite à considérer des files d'attente de taille bornée.

Toujours dans le cadre du projet Acotris, une autre voie qu'il nous semble intéressant d'explorer pourrait être la spécification de propriétés SIGNAL (propriétés multi-horloge) en utilisant le formalisme OCL (*Object Constraint Language*) de UML.

6.4. Mise en œuvre distribuée et modélisation d'architectures de communication

Participants : Loïc Besnard, Abdoulaye Elhadji Gamatié, Thierry Gautier, Paul Le Guernic.

Mots clés : *Signal*, *programmation synchrone*, *code distribué*, *architecture hétérogène*, *communication synchrone/asynchrone*, *description d'architecture*, *multi-tâche préemptif*, *OS temps réel*, *Arinc*.

La génération de code distribué pour les programmes synchrones pose deux grands types de difficultés. 1/ La compilation directe 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. 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.

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.

La méthodologie que nous proposons s'appuie sur la modélisation des architectures de communication en définissant en Signal un ensemble de primitives génériques sur la base de standards (ARINC dans le cadre du projet SafeAir). Cette modélisation est utilisée pour permettre l'évaluation temporelle de l'application décrite sur l'architecture d'implantation. Les modèles considérés doivent permettre d'étudier des ordonnancements préemptifs et, lorsque c'est possible, utiliser les résultats de méthodes d'analyse existantes (RMA...).

Nous avions déjà défini une bibliothèque Signal de modèles de composants de communication et synchronisation. Ces modèles reposent sur les spécifications de la norme ARINC [48], utilisée dans le domaine de l'avionique. Dans ce contexte, étant données une ou plusieurs applications, il est procédé à un découpage fonctionnel qui tient compte des ressources en temps et en espace mémoire disponibles. L'unité de ce partitionnement est appelée *partition* (c'est l'équivalent d'un programme dans le cas d'un environnement

contenant une seule application). La partition quant à elle est découpée en de nouvelles entités dites *process*. Le document [48] définit ces objets, ainsi que les mécanismes de communication (ex. *buffer, sampling port...*) et de synchronisation (comme le sémaphore). La bibliothèque a été consolidée et complétée avec les modèles des autres composants nécessaires à la description d'une application. Entre autres, les *process,* les services de gestion des *process* utiles à la définition d'un modèle OS temps réel, etc. Une introduction à l'approche utilisée pour la modélisation est présentée dans [18]. Un rapport de recherche la décrit plus en détail [20].

Deux applications sont en cours de développement pour permettre de valider l'approche. La première concerne la modélisation d'une application avionique réelle fournie par Airbus Toulouse (partenaire du projet SafeAir). Quant à la seconde, il s'agit de proposer un modèle du support d'exécution de programmes Java temps réel (JVMRT) utilisant la bibliothèque de composants ARINC (voir section 6.5). Dans les deux cas, la modélisation permet d'utiliser les outils de la plate-forme POLYCHRONY pour faire de la vérification, de la simulation, et de la validation.

Enfin, nous étudions une traduction de Signal vers les automates temporisés. Cela permettra d'accéder à des outils comme Kronos (de Verimag), et faire ainsi de l'évaluation temporelle.

6.5. Modélisation de haut niveau d'applications embarquées multi-tâches

Participants : Abdoulaye Elhadji Gamatié, Bruno Le Dez, Jean-Pierre Talpin.

Mots clés : *multi-tâche*, *Java temps réel*, *exécutif temps réel*, *Signal*, *Arinc*.

L'ingénierie d'applications embarquées en avionique passe par plusieurs étapes de conception allant du prototypage sur station de travail, à la réalisation de simulateurs puis le déploiement de composants logiciels sur des architectures embarquées. La plate-forme POLYCHRONY permet d'aider à la conception d'application dans ce flot de conception en permettant la capture en amont de modélisation de haut niveau (par exemple en Java temps réel) et leur spécialisation, c'est-à-dire la production, correcte par construction, d'exécutifs temps réel, en utilisant les techniques formelles mises en œuvre dans la plate-forme POLYCHRONY.

Nous proposons un modèle du support d'exécution de programmes Java temps réel (JVMRT) utilisant la bibliothèque de processus ARINC 653 de l'environnement POLYCHRONY ainsi qu'un schéma de traduction d'applications temps réel Java vers SIGNAL. Cette modélisation et cette traduction permettent l'utilisation des outils de vérification, de transformation de programmes (optimisation, distribution) de la plate-forme POLYCHRONY pour l'aide à la conception d'applications embarquées décrites au moyen du langage de programmation Java.

L'objectif du travail effectué est d'une part, de réduire le support d'exécution nécessaire (la JVM, qui est en fait un système d'exploitation temps réel assez standard) à son strict minimum. En effet, la bibliothèque ARINC 653 de POLYCHRONY (voir section 6.4) permet de simuler ce support d'exécution pour une architecture d'application donnée en sérialisant les blocs (sections critiques) de l'application au moyen de partitions, faisant appel à un jeu de primitives système minimal (manipulation d'horloges, d'interruptions matériel). La traduction des tâches Java temps réel dans le langage déclaratif SIGNAL de la plate-forme POLYCHRONY permet la compilation d'un code cible efficace. En particulier, l'utilisation de l'algorithmique de hiérarchisation de POLYCHRONY permet de restructurer globalement le contrôle dans l'application modélisée.

Afin de pouvoir transformer une application Java temps réel en SIGNAL, on utilise un découpage des tâches en processus de façon à se ramener à une architecture similaire à celle définie dans la norme ARINC 653 (une partition composée de sections critiques élémentaires). On obtient un processus pour chaque tâche. Un processus est synthétisé qui simule le comportement de l'ordonnanceur Java. Des mécanismes de communication, modélisés au moyen de la bibliothèque ARINC de SIGNAL, permettent de coordonner l'exécution des tâches.

Les applications de cette technique sont nombreuses et prometteuses. On peut en effet connecter la modélisation SIGNAL de l'application à des outils de vérification symbolique (e.g. SIGALI) ou temporisée (e.g. Kronos) afin de valider les expressions de besoins. Cette modélisation est un démonstrateur de l'utilisation de POLYCHRONY comme plate-forme sémantique ouverte pour l'ingénierie par composants d'applications embarquées.

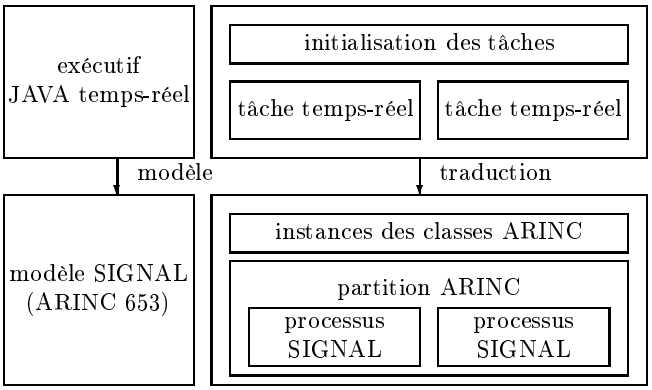

Figure 3.

Les domaines d'applications sont multiples. Initialement pensé pour l'avionique comme outil facilitant le déploiement de composants de simulation sur des architectures embarquées, le traducteur permet également d'envisager l'utilisation de Java comme support de programmation de haut niveau pour la conception et la simulation d'architectures logiciel/matériel, utilisant POLYCHRONY pour la production d'une architecture d'implémentation efficace.

6.6. Modélisation d'architectures et conception conjointe matériel/logiciel

Participants : Paul Le Guernic, Jean-Pierre Talpin.

Mots clés : *Signal*, *modèle polychrone*, *synchrone/asynchrone*, *architecture Gals*, *conception conjointe matériel/logiciel*, *codesign*, *raffinement*, *SpecC*.

Nous définissons dans un cadre nouveau le modèle mathématique relationnel qui fonde le langage SIGNAL. Le caractère « polychrone » (ou multi-horloge) du modèle est mis en relief : il permet la description de circuits et de systèmes possédant plusieurs horloges, et il rend possible le « raffinement ».

6.6.1. Un modèle polychrone des architectures globalement asynchrones et localement synchrones

Nous proposons dans [16] une refondation du modèle mathématique du langage SIGNAL en proposant une sémantique dénotationelle relationelle fondée sur des travaux antérieurs sur les structures synchrones [44] et les *models of computation* de Lee et al. [39]. L'originalité du modèle polychrone des traces marquées, présenté dans cet article, est qu'il est fondé sur une structure d'ordre partiel entre événements qui donne une ontologie purement relationnelle à la sémantique de SIGNAL. Dans ce modèle, le comportement d'un processus est décrit par un ensemble de traces où les signaux sont des ensembles totalement ordonnés d'événements. Cependant, au cœur d'une trace, les événements ne sont que partiellement ordonnés entre eux. Ceci permet de rendre compte de manière primitive des relations de synchronisation partielle des opérateurs de base de SIGNAL (échantillonnage et mélange déterministe). Une structure algébrique est élaborée autour de ces domaines pour rendre compte des phénomènes de latence, d'abord dans un cadre synchrone (le bégaiement), et ensuite dans un cadre asynchrone (la temporisation). Nous obtenons ainsi une structure de semi-treillis donnant directement une sémantique globalement asynchrone et localement synchrone à partir d'une structure de traces à horloges multiples. Armés de cette structure, nous proposons un modèle de conception d'architectures GALS partant de spécifications polychrones et utilisant une méthode formelle de raffinement fondée sur deux critères de validation formelle : l'endochronie, qui permet d'isoler un composant robuste dans une architecture en cours de conception ; et l'invariance de flot, qui permet de vérifier la correction du raffinement d'une spécification polychrone par une architecture constituée de processus endochrones et de protocoles de communication préservant les flots d'événements. Nous introduisons une procédure de décision garantissant la propriété d'invariance de flot pour une spécification polychrone, l'endo-isochronie. Cette procédure montre que le raffinement correct (c'est-à-dire préservant les flots) d'une spécification polychrone par une architecture

distribuée ne peut que consister en l'échange d'horloges de resynchronisation entre les composants de l'architecture, comme c'est le cas dans les travaux réalisés notamment dans la thèse de Pascal Aubry [25] et mis en œuvre dans la plate-forme POLYCHRONY.

6.6.2. Méthodologie de raffinement d'architectures matériel-logiciel

Dans [19], rédigé conjointement avec Rajesh Gupta (UCSD), Sandeep Shukla (Virginia Tech), Frédéric Doucet (UCI), nous réalisons une étude de cas de conception d'une architecture matérielle en faisant un parallèle entre les méthodologies de conception du langage de conception SPECC et SIGNAL, somme toute assez proches. Nous montrons que SIGNAL peut être utilisé comme une plate-forme sémantique formelle pour la dérivation « correcte par construction » de mise en œuvre matériel-logiciel au moyen de langage de modélisation de haut niveau. Nous évoquons les techniques permettant de modéliser un processus SPECC (appelé *behavior*) au moyen de SIGNAL mais aussi comment modéliser le support d'exécution nécessaire à la simulation de SPECC. Ce travail offre de nombreuses perspectives en vue du test (formel) de conformité entre modélisations de haut niveau, de la simulation, de la synthèse de protocoles et d'interfaces.

6.6.3. Modélisation de protocoles faiblement synchronisés

Dans [17], nous avons participé à la modélisation d'un protocole de communication non standard, car faiblement synchronisé, au moyen du langage SIGNAL et, en collaboration avec Hervé Marchand (projet Vertecs) réalisé la preuve de ce protocole au moyen de SIGALI. De cette étude ont découlé de nombreuses réflexions qui nous ont amenés notamment à proposer la notion d'invariance de flot dans [16].

7. Contrats industriels

7.1. Coopération bilatérale avec la société TNI-Valiosys

Mots clés : *Signal*, *Sildex*.

La société TNI, qui développe et commercialise l'environnement Sildex pour SIGNAL, est un partenaire associé à nombre de nos activités.
http://www.tni-valiosys.com/
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'est également poursuivie au sein du projet européen SafeAir.

Artist a permis de financer une partie de l'allocation de thèse de David Berner (doctorant depuis novembre 2002), dont les travaux au sein du projet Espresso porteront sur le développement de la thématique des architectures matériel-logiciel dans le projet (voir section 2.5.3) en coopération avec la société TNI-Valiosys, et plus particulièrement sur les aspects du test de conformité et de conception par composants, en collaboration avec nos partenaires NSF.

7.2. Coopération bilatérale avec la société Thomson Multimédia

Les aspects « simulation rapide », c'est-à-dire l'instrumentation formelle permettant de synthétiser automatiquement un simulateur d'architecture rapide à partir de composants VHDL existants, seront étudiés plus particulièrement par Bruno Gacel (doctorant à partir de janvier 2003) en collaboration avec la société Thomson Multimédia, dans le cadre d'un programme Cifre.

7.3. Projet RNTL Expresso, convention no2 01 C 0299 00 31307 01 1 (06/2001-05/2003)

Participants : Bruno Le Dez, Paul Le Guernic, Jean-Pierre Talpin.

Mots clés : *Java temps réel*, *systèmes enfouis*, *approche synchrone*, *inférence de régions*.

L'objectif du projet Expresso est de réaliser l'atelier mettant en jeu les composants nécessaires à l'environnement de développement d'applications temps réel Java pour des systèmes critiques enfouis ou embarqués et susceptibles d'être certifiés au sens des standards tels que le DO178-B.

La communauté temps réel qui souhaite utiliser le langage Java pour ses qualités intrinsèques (orientation objet, sécurité, etc.) ne peut le faire aujourd'hui sans payer le prix du manque de performance du code interprété ou du non-déterminisme de la machine virtuelle sous-jacente.

L'objectif du projet Expresso est de réaliser l'atelier mettant en jeu les composants nécessaires à l'environnement de développement d'applications temps réel Java pour des systèmes critiques enfouis ou embarqués et susceptibles d'être certifiés au sens des standards tels que le DO178-B.

Les composants retenus dans ce cadre sont la compilation et l'optimisation du *Java Byte code* vers le code binaire natif de la cible, une machine virtuelle Java décomposable de façon graduelle en un exécutif Java et des outils de vérification des propriétés temps réel et de l'ordonnancement des applications Java.

Les extensions temps réel du langage Java qui seront mises en œuvre s'appuieront sur les standards émergeants du « Sun Community Process » et du J Consortium. Le développement de ces extensions sera rendu accessible en mode « logiciel libre ».

Les membres partenaires de ce projet comprennent les industriels Thomson-CSF Detexis et EDF qui définissent les besoins et évaluent les résultats du produit, les sociétés de produits et services informatiques Aonix et Silicomp qui s'associent pour fabriquer la chaîne de compilation et l'exécutif Java et en feront la commercialisation, les laboratoires Inria/Irisa et Verimag, experts en techniques formelles qui complètent l'atelier logiciel avec des outils d'analyse temps réel.

L'environnement Java intégré (IDE) final sera disponible au bout de deux ans. Les applications ainsi produites s'exécuteront soit sur un système d'exploitation Posix, soit sur le noyau Raven pour machine nue dans le cas des systèmes certifiés ou fortement contraints.

Les aspects techniques et les retombées technologiques de ce travail pour ce qui concerne le projet Espresso sont décrits en section 6.5.

7.4. Projet RNTL Acotris, convention no2 00 C 0527 00 31307 01 1 (02/2001-07/2003)

Participants : Loïc Besnard, Thierry Gautier, Paul Le Guernic, Qiuling Pan.

Mots clés : *temps réel*, *Uml*, *Signal*, *SynDEx*.

Le projet Acotris a pour objet d'aider à la conception d'applications temps réel embarquées par l'intégration de la méthodologie et des concepts issus des approches synchrones au formalisme standard UML.
http://www.acotris.c-s.fr
Les partenaires du projet Acotris sont CS-SI, le CEA-List, MBDA (ex-Aérospatiale Matra Missiles), Sitia et l'Inria (projets Espresso et Ostre).

L'objectif du projet est d'aider à la conception d'applications temps réel embarquées imposant un niveau de parallélisme important, en se basant sur un support de conception UML et les outils de génération, dimensionnement et placement des applications issus des approches synchrones, ou plus largement multi-horloges.

Pour atteindre cet objectif, le projet propose :

- d'annoter les descriptions UML par des propriétés complémentaires (ex. : annotations multi-horloges). La démarche envisagée repose sur l'utilisation des mécanismes d'extension normalisés pour obtenir un nouveau profil UML dédié à ce domaine applicatif. Nous pourrons ainsi contribuer à l'évolution de la norme UML ;
- de mettre en place une passerelle UML - SIGNAL - SYNDEX qui permettra à l'utilisateur de concevoir progressivement son application décrite en UML en s'appuyant sur les techniques formelles de validation (voir section 6.3.2) ;

- de poursuivre cette conception jusqu'à la réalisation en utilisant des techniques de conception conjointe matériel/logiciel (*co-design*) avec une chaîne de développement et de simulation permettant d'évaluer différentes architectures multi-composant (processeurs et/ou circuits dédiés) et d'exécuter les applications en temps réel (SYNDEX).

7.5. Projet IST SafeAir, convention no1 00 C 0149 00 31307 00 5 (01/2000-07/2002)

Participants : Loïc Besnard, Abdoulaye Elhadji Gamatié, Thierry Gautier, Paul Le Guernic.

Mots clés : *système enfoui*, *méthode formelle*, *Statecharts*, *Scade*, *Sildex*, *Simulink*, *Lustre*, *Signal*, *avionique*, *description d'architecture*, *multi-tâche préemptif*, *OS temps réel*, *Arinc*, *vérification*, *validation de code*.

Le projet IST SafeAir (« Advanced Design Tools for Aircraft Systems and Airborne Software »), qui a pris la suite du projet Esprit Sacres, avait pour objectif de réaliser un environnement de développement pour les systèmes avioniques (ASDE), qui réponde aux besoins cruciaux en sûreté de fonctionnement pour les systèmes enfouis. Cet environnement doit permettre aux concepteurs de systèmes critiques embarqués de réduire significativement le risque d'erreurs et le temps de conception. L'approche proposée s'appuie sur des outils industriels existants, notamment, Statemate, Scade, Simulink, et offre de nouveaux outils pour la modélisation d'architectures, la vérification formelle, et la validation de code. Le langage Scade, basé sur Lustre, est utilisé comme langage pivot.
http://www.safeair.org

7.5.1. Présentation générale

Le projet IST-1999-10913 SafeAir a débuté en janvier 2000. Il a regroupé les partenaires suivants : Aérospatiale Matra Airbus, devenu Airbus France, DaimlerChrysler Aerospace Airbus, devenu Airbus Deutschland GmbH, IAI (Israël), Snecma Control Systems (France), Telelogic Technologies Toulouse (France), TNI (France), I-Logix (USA), Siemens (RFA), Offis (RFA), Inria (France), Weizmann Institute of Science (Israël).

L'environnement ASDE (« Avionics Systems Development Environment ») développé dans le projet SafeAir doit permettre :

- de réduire significativement l'effort de validation à l'intégration grâce à l'emploi de techniques de vérification formelle,
- de fournir une intégration des étapes de conception, depuis les outils de modélisation de niveau système jusqu'à une génération de code automatique respectant les standards DO-178B qui s'appliquent dans les systèmes embarqués critiques en avionique,
- d'offrir une approche permettant de prouver automatiquement la consistance du source et du code généré, ce qui permettrait de réduire considérablement les tests unitaires.

Sur le plan technique, l'architecture ASDE [24] est illustrée par la figure 4.

L'utilisateur de ASDE peut construire ses modèles à partir de plusieurs composants, chaque composant étant conçu avec l'un des outils Sildex, Simulink, Statemate et Scade. Les communications entre ces composants peuvent être synchrones ou asynchrones. L'outil ModelBuild est un nouvel outil, basé sur Sildex, qui a été développé au cours du projet. Il permet de décrire et de simuler l'intégration des différents composants d'un modèle sur une architecture. L'outil ModelVerify permet de vérifier des propriétés du modèle global et de ses composants. L'intégration se fait par échange de fichiers Scade.

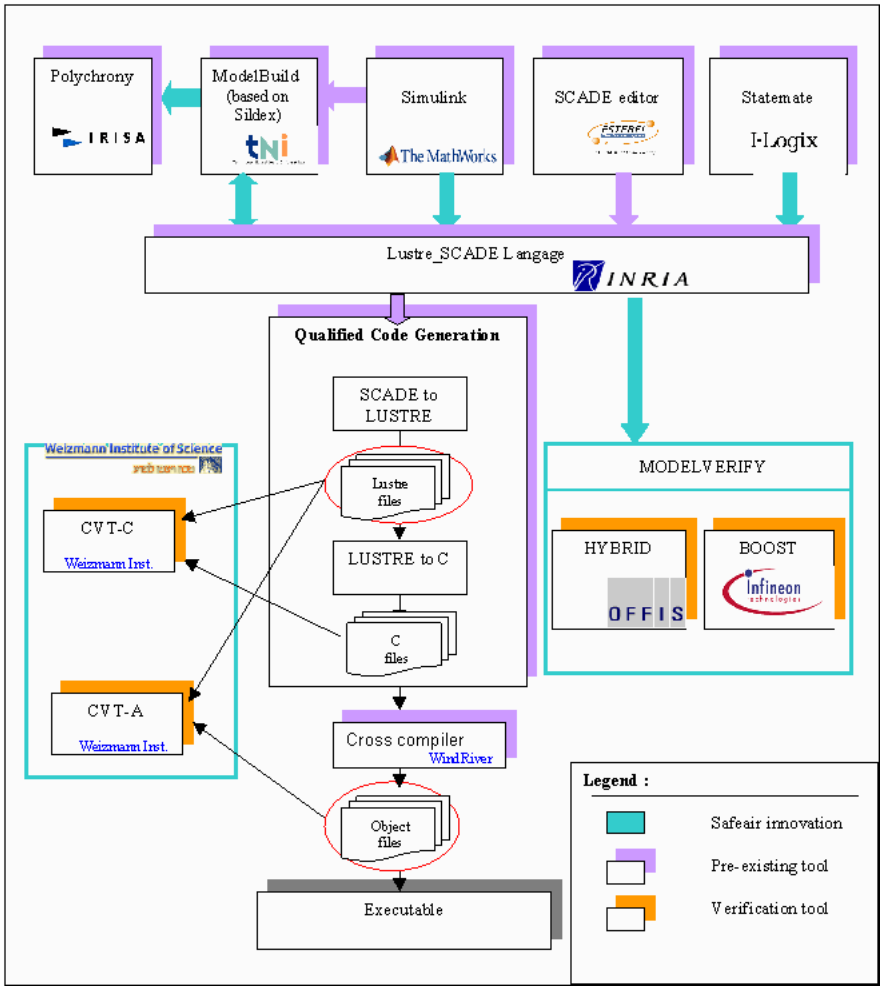

Figure 4. Architecture fonctionnelle de ASDE

7.5.2. Activités de Espresso dans le cadre du projet SafeAir

Les projets EP-ATR puis Espresso ont été impliqués dans SafeAir principalement sur les aspects suivants :

- définition de l'architecture ASDE ;
- rapprochement Lustre-SIGNAL (par exemple, la notion de *node* de Lustre a été ajoutée à SIGNAL) ;
- définition et mise à disposition d'un ensemble de fonctionnalités de transformations correctes de programmes en vue de la vérification et de la génération de code ;
- études d'abstractions de programmes ;
- définition en SIGNAL d'une bibliothèque de composants de communication pour une mise en œuvre temps réel (voir section 6.4) ;
- étude des caractéristiques temporelles d'une application pour une mise en œuvre temps réel (morphismes de programmes SIGNAL).

SafeAir a notamment été utilisé comme cadre de développement et d'expérimentation pour la version « domaine public » de la plate-forme POLYCHRONY.

8. Actions régionales, nationales et internationales

8.1. Actions internationales

8.1.1. Collaboration nsf-Inria

Dans le cadre du programme de collaboration entre l'Inria et la NSF, nous travaillons en étroite collaboration avec le projet Balboa, initialement basé à UCI, et conjointement animé par Rajesh Gupta (UCSD) et Sandeep Shukla (Virginia Tech). Cette collaboration est très fructueuse, puisqu'elle a été le cadre de nombreux échanges (voir ci-dessous), diffusions de résultats (voir section 9.1), de travaux scientifiques avec la rédaction d'un premier article (voir section 6.6.2) mais surtout la création de la conférence ACM et IEEE MEMOCODE (modèles et méthodes formelles pour le codesign). Les discussions qu'elle a suscitées ont été fondatrices des orientations du projet dans le domaine du *codesign* (voir section 2.5.3).

Dans le cadre de cette collaboration NSF-Inria avec le projet Balboa de l'Université de Californie à San Diego, Jean-Pierre Talpin et Paul Le Guernic ont, avec Rajesh Gupta (UCSD) et Sandeep Shukla (Virginia Tech), créé la première édition de la conférence MEMOCODE (voir http://www.irisa.fr/MEMOCODE). La conférence MEMOCODE porte sur les méthodes et modèles formels pour le *codesign,* et rassemble de nombreux chercheurs universitaires (MIT, Princeton, CMU, Columbia, UCSD, VTech, Inria, etc.) et industriels (Synopsis, Motorola, Mentor Graphics, Philips, Cadence) proposant notamment de répondre à des études de cas sur les défis technologiques présents du domaine.

Toujours dans le cadre de cette collaboration, nous avons engagé un partenariat de long terme avec le projet Balboa, initialement localisé à UCI, et maintenant conjointement supervisé par Rajesh Gupta (UCSD) et Sandeep Shukla (Virginia Tech). Cette collaboration vise à l'utilisation de la plate-forme d'aide à la conception POLYCHRONY dans l'outil de conception SYSTEMC Balboa en mettant en œuvre des techniques de traduction, de vérification et de synthèse permettant la réalisation « correcte-par-construction » d'architectures logiciel-matériel au moyen du langage de conception SYSTEMC. Il s'agit notamment de donner un type comportemental de composants SYSTEMC au moyen de processus SIGNAL, et de synthétiser automatiquement des *wrappers* entre composants SYSTEMC existants afin d'assurer leur interopérabilité, correcte d'un point de vue sémantique, une fois déployés sur une architecture donnée.

Dans ce but, le projet Espresso a accueilli Frédéric Doucet, étudiant de Rajesh Gupta à UCI pour un stage de formation et de tutorat, supervisé par Loïc Besnard, en vue de se familiariser à l'utilisation de SIGNAL. Cette visite a été conclue par la diffusion (sous forme LGPL) de l'outil POLYCHRONY à UCI, UCSD et VTech, avec qui nous allons réaliser l'intégration de son algorithmique à Balboa.

8.1.2. Réseau d'excellence Artist

Nos travaux sur la modélisation de protocoles (voir section 6.6.3) entrent dans le cadre des travaux réalisés dans le cadre du réseau d'excellence Artist, où nous élaborons actuellement plusieurs programmes de collaborations.

9. Diffusion des résultats

9.1. Animation de la communauté scientifique

Paul Le Guernic est membre du bureau exécutif du RNTL, « Réseau National de recherche et d'innovation en Technologies Logicielles ». Il est aussi membre du comité d'orientation du RMNT, « Réseau de recherche en Micro et Nano Technologies ».

Jean-Pierre Talpin a participé au comité de programme de la conférence IEEE *Digital automation and test Europe* (DATE'2003) qui se tiendra à Munich en mars 2003.

Thierry Gautier est membre du comité de programme de « SLAP'2003 : Synchronous Languages, Applications, and Programming ».

Paul Le Guernic et Jean-Pierre Talpin sont, avec Sandeep Shukla et Rajesh Gupta, les initiateurs de la conférence ACM et IEEE MEMOCODE. Paul Le Guernic est membre de son comité de pilotage. Jean-Pierre Talpin préside avec Sandeep Shukla son comité de programme et assure la direction des finances de l'événement.

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.

Thierry Gautier, Bernard Houssais et Loïc Besnard ont donné des cours de DESS-Isa et Diic 2^{e} année consacrés à la programmation temps réel.

9.3. Participation à des colloques, séminaires, invitations

Jean-Pierre Talpin a donné un colloque intitulé « Polychrony for system-design » à l'Université de Californie à Irvine, dans le cadre des conférences *Cal(IT)*^{2} *events* (voir http://www.calit2.net).

Thierry Gautier a présenté la plate-forme POLYCHRONY au congrès « SIM.OUEST 2002 : MARINE - Virtual Shipbuilding, the current state of affairs » (Nantes, 28-29 novembre 2002).

Jean-Pierre Talpin, Abdoulaye Gamatié et Mirabelle Nebut ont participé à « Synchron 2002 - 9th international Open Workshop on Synchronous Reactive languages » (Agelonde, 25-29 novembre 2002).

Jean-Pierre Talpin présentera un tutoriel commun avec R. Gupta (UCSD), Stephen Edwards (Columbia) et S. Shukla (Virginia Tech), intitulé « High-level modeling and validation methodologies for embedded systems-bridging the productivity gap », à la conférence IEEE *VLSI design,* à New-Delhi.

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

10. Bibliographie

Bibliographie de référence

[1] A. BENVENISTE, B. CAILLAUD, P. LE GUERNIC. *From synchrony to asynchrony.* in « CONCUR'99, Concurrency Theory, 10th International Conference », série Lecture Notes in Computer Science, volume 1664, Springer, éditeurs J. C. M. BAETEN, S. MAUW., pages 162-177, août, 1999.

[2] A. BENVENISTE, P. LE GUERNIC, C. JACQUEMOT. *Synchronous programming with events and relations : the Signal language and its semantics.* in « Science of Computer Programming », volume 16, 1991, pages 103-149.

[3] T. GAUTIER, P. LE GUERNIC. *Code generation in the SACRES project.* in « Towards System Safety, Proceedings of the Safety-critical Systems Symposium, SSS'99 », Springer, éditeurs F. REDMILL, T. ANDERSON., pages 127-149, Huntingdon, UK, février, 1999, ftp://ftp.irisa.fr/local/signal/publis/articles/SSS-99:format_dist.ps.gz.

[4] 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.

[5] 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 Cosynthesis for Reconfigurable Systems », IEE, pages 6/1-6/9, Février, 1996, ftp://ftp.irisa.fr/local/signal/publis/articles/HWSWCRS-96:profiling.ps.gz.

[6] H. MARCHAND, P. BOURNAI, M. LE BORGNE, P. LE GUERNIC. *Synthesis of Discrete-Event Controllers based on the Signal Environment.* in « Discrete Event Dynamic System : Theory and Applications », numéro 4, volume 10, octobre, 2000, pages 347-368.

[7] H. MARCHAND, M. SAMAAN. *On the Incremental Design of a Power Transformer Station Controller using Controller Synthesis Methodology.* in « FM'99 - Formal Methods », série Lecture Notes in Computer Science, volume 1709, Springer, éditeurs J. M. WING, J. WOODCOCK, J. DAVIES., pages 1605-1624, Toulouse, France, septembre, 1999, ftp://ftp.irisa.fr/local/signal/publis/articles/FM99:control_synth_appli.ps.gz.

[8] 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, pages 163-173, 1995, ftp://ftp.irisa.fr/local/signal/publis/articles/PLDI-95:compil.ps.gz.

[9] 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 », série Lecture Notes in Computer Science, numéro 891, Springer Verlag, éditeurs C. LEWERENTZ, T. LINDNER., pages 115-129, janvier, 1995.

[10] P. LE GUERNIC, T. GAUTIER. *Data-Flow to von Neumann : the Signal approach.* in « Advanced Topics in Data-Flow Computing », éditeurs J. L. GAUDIOT, L. BIC., pages 413-438, 1991, ftp://ftp.irisa.fr/local/signal/publis/articles/ATDFC-91:sem_distr.ps.gz.

[11] P. LE GUERNIC, T. GAUTIER, M. LE BORGNE, C. LE MAIRE. *Programming Real-Time Applications with Signal.* in « Proceedings of the IEEE », numéro 9, volume 79, Septembre, 1991, pages 1321-1336, ftp://ftp.irisa.fr/local/signal/publis/articles/ProcIEEE-91:gen_lang.ps.gz.

Thèses et habilitations à diriger des recherche

[12] M. KERBŒUF. *Orientation objet d'un calcul de processus synchrones.* thèse de doctorat, Université de Rennes 1, IFSIC, décembre, 2002.

[13] J.-C. LE LANN. *Simulation et synthèse de circuits s'appuyant sur le Modèle Synchrone.* thèse de doctorat, Université de Rennes 1, IFSIC, mars, 2002.

[14] M. NEBUT. *Réactions synchrones : spécification et analyse.* thèse de doctorat, Université de Rennes 1, IFSIC, novembre, 2002.

Articles et chapitres de livre

[15] A. BENVENISTE, P. CASPI, S. EDWARDS, N. HALBWACHS, P. LE GUERNIC, R. DE SIMONE. *The Synchronous Languages Twelve Years Later.* in « Proceedings of the IEEE, special issue on Embedded Systems », 2002.

[16] P. LE GUERNIC, J.-P. TALPIN, J.-C. LE LANN. *Polychrony for system design.* in « Journal of Circuits, Systems and Computers. Special Issue on Application Specific Hardware Design », 2002.

Communications à des congrès, colloques, etc.

[17] A. BENVENISTE, P. CASPI, LE GUERNIC. P., H. MARCHAND, J.-P. TALPIN, S. TRIPAKIS. *A Protocol for Loosely Time-Triggered Architectures.* in « Proc. of 2002 Conference on Embedded Software (EMSOFT'02) », série LNCS, volume 2491, Springer, éditeurs J. SIFAKIS, A. SANGIOVANNI-VINCENTELLI., octobre, 2002.

[18] A. GAMATIÉ, T. GAUTIER. *Modeling of Modular Avionics Architectures Using the Synchronous Language SIGNAL.* in « Proceedings of the Work In Progress session, 14th Euromicro Conference on Real Time Systems, ECRTS'02 », Vienne, Autriche, pages 25-28, juin, 2002.

[19] J.-P. TALPIN, P. LE GUERNIC, S. K. SHUKLA, R. GUPTA, F. DOUCET. *A polychronous model for high-level component-based system design.* in « Digital Automation and Test Europe », IEEE Press, mars, 2003, à paraître.

Rapports de recherche et publications internes

[20] A. GAMATIÉ, T. GAUTIER. *Synchronous Modeling of Modular Avionics Architectures using the SIGNAL Language.* rapport technique, Irisa, décembre, 2002.

Divers

[21] L. BESNARD, T. GAUTIER, Q. PAN, C. MACABIAU, Y. SOREL. *SIGNAL $\longrightarrow$ SynDEx translation, V1.0.* juin, 2002, Analyse et Conception à Objets Temps Réel pour Implantation asynchrone/Synchrone (ACOTRIS).

[22] L. BESNARD, T. GAUTIER, P. LE GUERNIC. *SIGNAL V4-INRIA version : Reference Manual.* décembre, 2002, documentation Polychrony.

[23] B. HOUSSAIS. *The Synchronous Programming Language SIGNAL, A Tutorial.* avril, 2002, documentation Polychrony.

[24] SAFEAIR TEAM. *Final report.* 2002, Advanced Design Tools for Aircraft Systems and Airborne Software (SafeAir).

Bibliographie générale

[25] P. AUBRY. *Mises en œuvre distribuées de programmes synchrones.* thèse de doctorat, Université de Rennes 1, IFSIC, octobre, 1997, ftp://ftp.irisa.fr/techreports/theses/1997/aubry.ps.gz.

[26] J.-R. BEAUVAIS, É. RUTTEN, T. GAUTIER, P. LE GUERNIC, Y.-M. TANG. *Modelling Statecharts and Activitycharts as Signal equations.* in « ACM Transactions on Software Engineering and Methodology », numéro 4, volume 10, 2001.

[27] A. BENVENISTE, G. BERRY. *The Synchronous Approach to Reactive and Real-Time Systems.* in « Proceedings of the IEEE », numéro 9, volume 79, Septembre, 1991, pages 1270-1282.

[28] 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 », numéro 803, Springer-Verlag, éditeurs J. W. DE BAKKER, W. DE ROEVER, G. ROZENBERG., pages 1-45, Juin, 1993.

[29] G. BERRY. *Real-Time Programming : special purpose or general purpose languages.* in « Information processing 89 », Elsevier Science Publisher B.V., éditeurs G. X. RITTER., 1989.

[30] G. BERRY, G. GONTHIER. *The* ESTEREL *synchronous programming language : design, semantics, implementation.* in « Science of Computer Programming », numéro 2, volume 19, novembre, 1992, pages 87-152.

[31] F. BESSON, T. JENSEN, J.-P. TALPIN. *Polyhedral Analysis for Synchronous Languages.* in « Static Analysis », série Lecture Notes in Computer Science, volume 1694, Springer, éditeurs A. CORTESI, G. FILÉ., pages 51-68, 1999.

[32] R. BRYANT. *Graph-Based Algorithms for Boolean Function Manipulations.* in « IEEE Transaction on Computers », numéro 8, volume C-45, Août, 1986, pages 677-691.

[33] E. M. CLARKE, E. A. EMERSON, A. P. SISTLA. *Automatic Verification of finite-State Concurrent Sytems Using Temporal Logic Specifications.* in « ACM Transactions on Programming Langages and Systems », numéro 2, volume 8, Avril, 1986, pages 244-263.

[34] N. HALBWACHS, P. CASPI, P. RAYMOND, D. PILAUD. *The Synchronous Dataflow Programming Language Lustre.* in « Proceedings of the IEEE », numéro 9, volume 79, September, 1991, pages 1305-1320.

[35] N. HALBWACHS. *Synchronous programming of reactive systems.* Kluwer, 1993.

[36] D. HAREL. *Statecharts : A Visual Formalism for Complex Systems.* in « Science of Computer Programming », numéro 3, volume 8, June, 1987, pages 231-274.

[37] B. JEANNET, N. HALBWACHS, P. RAYMOND. *Dynamic Partitioning in Analyses of Numerical Properties.* in « Static Analysis Symposium, SAS'99 », série Lecture Notes in Computer Science, Springer, Venezia (Italy), septembre, 1999.

[38] A. LANUSSE, S. GÉRARD, F. TERRIER. *Real-Time Modeling with UML : the ACCORD Approach.* in « UML98 : Beyond the Notation », série LNCS, volume 1618, Springer-Verlag, Lecture Notes in Computer Science, éditeurs J. BEZIVIN, P. A. MULLER., 1998.

[39] E. A. LEE, A. SANGIOVANNI-VINCENTELLI. *A framework for comparing models of computation.* in « IEEE Transactions on Computer-Aided Design », numéro 12, volume 17, décembre, 1998.

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

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

[42] D. NOWAK, J.-P. TALPIN, T. GAUTIER, P. LE GUERNIC. *An ML-like module system for the synchronous language Signal.* in « European Conference on Parallel Processing (Euro-Par'97) », Springer-Verlag, LNCS 1300, pages 1244-1253, août, 1997, ftp://ftp.irisa.fr/local/signal/publis/articles/EuroPar-97:modul.ps.gz.

[43] D. NOWAK, J.-P. TALPIN, T. GAUTIER, P. LE GUERNIC. *An ML-like module system for the synchronous language Signal.* in « European Conference on Parallel Processing (Euro-Par'97) », Springer-Verlag, LNCS, août, 1997.

[44] D. NOWAK, J.-P. TALPIN, P. LE GUERNIC. *Synchronous Structures.* in « Proceedings of the 10th International Conference on Concurrency Theory (CONCUR'99) », série LNCS, volume 1664, Springer, August, 1999, ftp://ftp.irisa.fr/local/signal/publis/articles/TPHOLs-98:sem_verif.ps.gz.

[45] V. RUSU. *Analyzing Automata with Presburger Arithmetic and Uninterpreted Function Symbols.* rapport technique, numéro 4100, INRIA, 2001.

[46] J.-P. TALPIN, A. BENVENISTE, B. CAILLAUD, C. JARD, Z. BOUZIANE, H. CANON. BDL, *a language of distributed reactive objects.* in « International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC'98) », IEEE Press, avril, 1998.

[47] Y. WANG. *UML et technologie synchrone pour les systèmes réactifs distribués.* thèse de doctorat, Ifsic, Université de Rennes 1, décembre, 2001.

[48] AIRLINES ELECTRONIC ENGINEERING COMMITTEE. *Avionics Application Software Standard Interface - ARINC Specification 653.* janvier, 1997, Aeronautical Radio, INC. Maryland (USA).

[49] SACRES CONSORTIUM. *The Declarative Code* DC+*, version 1.4.* novembre, 1997, ftp://ftp.irisa.fr/local/signal/publis/research_reports/ Esprit project EP 20897 : Sacres.