
HABILITATION À DIRIGER DES RECHERCHES

présentée devant

**L'Université de Rennes 1
Institut de Formation Supérieure
en Informatique et en Communication**

par

Éric RUTTEN

Programmation sûre des systèmes de contrôle/commande :
le séquençement de tâches flot de données dans les langages réactifs

soutenue le 20 décembre 1999 devant le jury composé de

M.	Jean-Pierre Banâtre	Président
MM	Jean-Pierre Elloy	Rapporteur
	Nicolas Halbwachs	Rapporteur
	Lionel Marcé	Rapporteur
MM	Bernard Espiau	Examineur
	Paul Le Guernic	Examineur
	Klaus Winkelman	Examineur

Résumé

Le contexte de ce travail est la *programmation sûre et structurée* ou de haut-niveau (au sens d'indépendante de la mise en œuvre) de contrôleurs pour des systèmes en interaction avec un *environnement physique* ayant sa dynamique propre. Ces systèmes se caractérisent par leur hybridité (continu/discret), leur complexité (diversité des aspects à prendre en compte), et la criticité de leur sûreté. Ils se retrouvent dans des contextes applicatifs divers, et leur modélisation doit prendre en compte ce caractère mixte. Dans le domaine du contrôle de procédés physiques, il s'agit de l'interaction entre la régulation et les lois de commande, et les transitions ou commutations entre ces modes continus. Les systèmes robotiques en sont une occurrence, ainsi que les contrôleurs industriels ou embarqués (avionique, transports, ...). La programmation des contrôleurs de ces systèmes requiert l'usage de techniques particulières pour assurer leur correction, d'une part du fait de leur complexité, et d'autre part du fait du caractère critique de leur sûreté. En particulier, un modèle formel est nécessaire pour fournir un support à des outils d'analyse qui offrent une assistance à la validation.

On traite ici de façons dont on peut intégrer dans un langage de programmation des notions répondant à ces besoins de formalisme mixte de description et de support pour des outils d'assistance à la conception de contrôleurs. Une exploration du problème commence par diverses formes de programmation robotique en termes de la mission à accomplir : elle permet de dégager des éléments de base, en l'occurrence le séquençement tâches ou la préemption pour l'aspect discret, et les processus à flot de données pour l'aspect continu.

Ces éléments sont intégrés dans la définition, dans le cadre du langage à flots de données synchronisés SIGNAL, d'une structure de tâche associant un processus à un intervalle d'activité, hors duquel il est interrompu ou suspendu (c'est-à-dire redémarré, le cas échéant, respectivement depuis son état initial déclaré ou bien depuis son état courant à la suspension). Les principes de base du contrôle des tâches en sont repris dans la modélisation de langages de contrôle/commande présentant une combinaison de formalismes impératifs et à flot de données. Il s'agit de STATEMATE (STATECHARTS et ACTIVITYCHARTS), et des langages de la norme IEC 1131 de programmation des automatismes industriels (dont une forme de GRAFCET). Ceci permet de bénéficier pour ces langages des outils de vérification et de génération de code de l'environnement SIGNAL, et, au travers du format d'échange synchrone DC+, de l'ensemble de la technologie synchrone.

Des applications sont traitées dans différents contextes expérimentaux : le contrôleur d'une cellule de production, la vision active pour la reconstruction 3D en robotique, l'automatisme de contrôle d'un transformateur électrique, la modélisation comportementale en animation et simulation.

Les perspectives se présentent à la fois dans des développements de l'étude des langages multi-formalismes et dans la prise en compte de spécificités du domaine de la programmation de système de contrôle/commande.

Mots-clefs : Systèmes réactifs, séquençement de tâches, préemption, flot de données, systèmes hybrides, programmation synchrone, robotique.

Remerciements

Merci à :

Jean-Pierre Banâtre, Professeur à l'Université de Rennes 1, Directeur de l'IRISA pendant les années qui ont suivi ma thèse, de m'avoir suggéré d'être candidat à un poste de chercheur à l'IRISA ;

Jean-Pierre Elloy, Professeur à l'École Centrale de Nantes, qui apporte un regard temps-réel et non-strictement-synchrone à mes travaux ;

Nicolas Halbwachs, Directeur de Recherche du CNRS à Grenoble, pour son regard expert, et sa présence dans un des laboratoires de mon nouveau milieu Grenoblois ;

Lionel Marcé, Professeur à l'Université de Bretagne Occidentale, directeur il y a un certain temps de ma thèse, après laquelle nous avons suivi des chemins différents mais parallèles ;

Bernard Espiau, Directeur de Recherche à l'INRIA-Rhône-Alpes, dont j'ai pu apprécier l'écoute et l'ouverture d'esprit, à diverses reprises au cours de mon évolution ;

Paul Le Guernic, Directeur de Recherche à l'IRISA/INRIA-Rennes, pour son accueil dans le projet où mon travail m'a amené jusqu'ici, et pour l'ambition et la rigueur auxquelles il m'a exposé ;

Klaus Winkelman, Chercheur à Siemens ZT, pour l'intérêt de nos contacts dans le cadre du projet Esprit SACRES qu'il coordonnait, comme dans des circonstances plus académiques ;

et aussi :

au projet EP-ATR dans son ensemble, contexte vital de mon activité, et plus particulièrement à ceux avec qui j'ai travaillé directement : Sophie Pinchinat, Hervé Marchand, Thierry Gautier, Loïc Besnard, Patricia Bournai, Albert Benveniste, Roland Houdebine, Yan-Mei Tang, Tocheou Pascaline Amagbe-gnon... et ceux qui ont eu la patience de me subir, en thèse : Jean-René Beauvais, Sylvain Machard, Fernando Jiménez-Fraustro, et stage de DEA : Florent Martinez, Christine Sinoquet, Sébastien Gicquel, Antoine Hahusseau, Mirabelle Nebut ;

à l'IRISA, laboratoire vivant et lieu d'échanges multiples, pas seulement professionnels, et spécialement à : François Chaumette, Éric Marchand, Stéphane Donikian, Philippe Besnard, Sylvie Thiébaux ... et à la bibliothèque, au service des missions, à la cafet, à l'équipe système ... ;

à l'INRIA-Rhône-Alpes, laboratoire tout aussi vivant qui m'accueille maintenant, et aussi aux divers autres éléments du paysage de la recherche Grenobloise où je commence à me reconnaître, en particulier le LAG et VERIMAG ;

à ceux, hors de l'IRISA, avec qui j'ai eu le plaisir de coopérer au fil du temps : Olivier Roux à l'IR-CyN (Nantes) (de qui je tiens aussi l'idée du rappel d'activités à chaque section du document), R.K. Shyamasundar au TIFR (Bombay, Inde), Mazen Saman à la DER d'EDF, ceux avec qui j'ai travaillé dans le projet Esprit SACRES (particulièrement Xavier Méhaut de TNI, Philippe Baufreton et Hugues Granier de la SNECMA), et au cours de mon post-doctorat ERCIM : Ève Coste-Manière et Daniel Simon à l'INRIA Sophia-Antipolis, Joachim Hertzberg à la GMD (Sankt-Augustin, Allemagne), Paul ten Hagen, Farhad Arbab et Ivan Herman au CWI (Amsterdam, Pays-Bas) ;

à toutes les personnes qui ont fait que la vie a été belle ou intense tout ce temps-là, à qui je laisse le soin de se reconnaître (certaines savent pourquoi, d'autres pas) ;

à Pierrick Nicolas, pour m'avoir fait envisager de suivre un DEA, au départ de tout ça.

Avant-propos

Ce document fait la synthèse d'une dizaine d'année de recherches autour du thème de la programmation des systèmes réactifs, et en particulier des systèmes de contrôle/commande dont les systèmes robotiques sont une instance. Il reprend l'ensemble de mes travaux, depuis la thèse jusqu'à aujourd'hui, autour de ce thème : il s'agit de proposer des langages de programmation pour ces systèmes mixtes, fondés sur un modèle formel qui puisse être à la base d'outils d'assistance à la conception, et confrontés à des applications. Je tente ici notamment d'établir le point commun conceptuel entre des résultats parfois techniquement variés. Les travaux les plus anciens sont esquissés brièvement, pour introduire quelques éléments de la problématique, tels que les notions de tâche et de séquençement de tâches.

L'essentiel de ce document se focalise sur mes activités à l'IRISA/Inria-Rennes, dans le projet Ep-Atr (Environnement de Programmation pour Application Temps-Réel). Il s'agit plus spécifiquement du séquençement de tâches flot de données dans les systèmes réactifs (ici, le langage SIGNAL) et de son application à la programmation robotique et à des systèmes de contrôle.

Ce document est synthétique, et ne rend compte que de relativement peu de détails ; on pourra trouver dans les articles cités un exposé plus complet.

Par ailleurs, je décrirai régulièrement, au cours du document, les activités correspondant aux travaux décrits : publications, outils, enseignement et encadrements, collaborations.

Organisation du document

Un chapitre introductif (Chapitre 1) expose le contexte du travail : la programmation sûre et structurée de systèmes de contrôle/commande, et la problématique à laquelle je me suis attaqué : les langages multi-formalismes et leurs environnements de programmation. Il résume ensuite les notions de base sur lesquelles on travaille, en décrivant les travaux passés exploratoires, et comment ils préparaient la suite.

Le chapitre suivant (Chapitre 2) décrit ma contribution sur les modèles de programmation de systèmes de contrôle/commande et le multi-formalisme, dans le contexte de la programmation synchrone. Il s'agit de structures de séquençement de tâches flot de données, concrétisées par *GTi*¹, extension de SIGNAL. Des travaux liés concernent la préemption dans les systèmes réactifs, autour de l'algèbre de processus PAL. Les principes de ces travaux sont repris dans le cadre de la modélisation de langages plus complexes que sont STATEMATE (STATECHARTS et ACTIVITYCHARTS) et les langages de la norme IEC 1131 (dont le GRAFCET, sous la forme des SEQUENTIAL FUNCTION CHARTS).

Puis un chapitre (Chapitre 3) décrit la confrontation de ces formalismes à des applications, notamment à la robotique et à des systèmes de contrôle : un contrôleur de cellule productique illustre l'utilisation de SIGNAL, et SIGNAL*GTi* est appliqué à un système de vision active en robotique, à un contrôleur d'automatisme de disjoncteur dans un transformateur électrique, et à un système de simulation et d'animation.

Enfin, j'expose des perspectives (Chapitre 4) des points de vue des langage multi-formalisme et du séquençement de tâches flot de données, de la programmation des systèmes robotiques ou de contrôle/commande, et d'un élargissement des domaines d'application.

1. *GTi* pour *G*estion de *T*âches et d'*i*ntervalles

Chapitre 1

Problématique

Dans ce chapitre, après un exposé du contexte dans lequel se situent ces travaux, et de leur motivation (Section 1.1), je définirai la problématique à laquelle je m'attaque (Section 1.2), avant de faire l'exposé synthétique de mes premières contributions (Section 1.3) ; elles préparent mes contributions principales qui seront présentées de façon plus détaillée dans les chapitres suivants (Chapitres 2 et 3).

1.1 Contexte et motivation

Le contexte de ce travail est la *programmation sûre et structurée* ou de haut-niveau (au sens d'indépendante de la mise en œuvre) de contrôleurs pour des systèmes en interaction avec un *environnement physique* ayant sa dynamique propre. Des exemples typiques se trouvent dans le contrôle et la commande en robotique, plus généralement dans les systèmes électro-mécaniques, les automatismes industriels, les systèmes embarqués, en avionique, ou dans les transports. Ces systèmes se caractérisent par leur hybridité (continu/discret), leur complexité (diversité des aspects à prendre en compte), et la criticité de leur sûreté.

Systèmes hybrides. Leur fonctionnement fait intervenir des aspects continus (qu'en physique et automatique on modélise par des équations différentielles) et des aspects discrets (modélisés en informatique, en tant que systèmes réactifs [52], par des systèmes de transition, automates à états). Cette mixité pose le problème d'un découplage et d'un interfaçage permettant de s'abstraire, dans l'étude d'un aspect, de l'autre, tout en tenant compte des interactions entre les deux. La modélisation de ces systèmes peut bénéficier d'une approche *multi-formalisme* de leur description.

Systèmes complexes. Leur conception fait intervenir des techniques et des compétences diverses pour leur régulation et commande, leur mise en œuvre, et la définition de leurs missions, en termes de leurs modes de fonctionnement, et de la gestion de la commutation entre ces modes. Ce dernier point en particulier a avantage à pouvoir être formulé en termes autres que de mise en œuvre, et découplé des lois de commande. De même les lois de commande ne s'étudient pas en termes de leur mise en œuvre par des tâches de calcul sur un système d'exploitation. Ceci suggère une *structuration*, distinguant par exemple différents niveaux d'intervention; un exemple en est la structuration de l'environnement ORCCAD [24].

Systèmes à sécurité critique. Les conséquences d'un dysfonctionnement mettent souvent en danger les personnes (transportées dans le cas d'un avion par exemple), ou les installations (coûteuses et distantes, et ne laissant pas la possibilité d'intervention directe, dans le cas de la robotique d'exploration planétaire par exemple). Ces effets se présentent de façon plus flagrante que dans le cas de perturbations de systèmes informatiques déconnectés d'un environnement physique. Ceci induit un besoin d'*assistance à la conception* ou à l'opération de ces systèmes, sous forme de méthodes d'analyse, de validation et de mise en œuvre correcte. Pour être d'une aide effective, ces méthodes doivent être dotées du support d'outils automatisant cette assistance. Pour effectuer leurs calculs, ces outils requièrent des modèles formels des comportements du système. L'approche synchrone [18, 47] est une réponse à ce besoin, fournissant un ensemble de langages de différents styles, fondés sur un modèle formel indépendant de la mise en œuvre, qui sert à l'analyse de cohérence des spécifications, à leur compilation et à leur vérification. L'intégration de tels outils, éventuellement autour de formats d'échanges, fournit des environnements de conception, dont SACRES, décrit en Section 2.2.1, est un exemple [44].

Il existe donc un besoin de modèles de programmation qui prennent en compte la mixité continu/discret, permettent de structurer la spécification de façon à découpler ces aspects, et reposent sur un modèle formel qui puisse offrir un support à des outils d'assistance à la spécifications, l'analyse et la mise en œuvre des systèmes. L'approche synchrone à la programmation des systèmes réactifs et temps-réels est un cadre dans lequel on peut définir la problématique et apporter une contribution vers une solution.

1.2 Problématique

Dans ce contexte, la problématique abordée dans ce travail est le *séquencement de tâches flot de données dans les systèmes réactifs*, modélisant les contrôleurs de systèmes hybrides, et comprenant l'assistance à sa spécification et analyse, et ses applications, en particulier à la *robotique*.

Modèles mixtes.

Notions de base. Les deux notions de séquencement et de tâche reflètent le caractère hybride des systèmes qui nous intéressent.

Les *tâches* se conçoivent au sens de processus qui présentent une certaine continuité dans leur interaction avec l'environnement du contrôleur. Cette continuité peut correspondre au caractère continu échantillonné d'un algorithme de contrôle, ou à celui de l'interaction des systèmes réactifs avec leur environnement. Un exemple est la mise en œuvre d'une loi de commande en automatique, boucle infinie reflétant l'échantillonnage de la fonction continue, sans fin prédéfinie, constituant un mode d'interaction avec ou de contrôle de l'environnement. Ce caractère se prête naturellement à la description par *flot de données*.

En particulier leur démarrage, leur suspension et reprise, et leur arrêt ne sont pas nécessairement définis de façon intrinsèque. Le *séquencement* dont il est question vient alors précisément remplir le rôle de contrôler, de l'extérieur, à quels instants on démarre ou arrête une tâche, et vers quelles autres tâches la commutation se fait. Une telle séquence peut elle-même être traitée comme une tâche dont le contrôle est assuré de l'extérieur.

Exemples. On trouve des occurrences de mécanismes de séquencement de l'extérieur¹ similaires dans des contextes divers :

- dans les systèmes d'exploitation, sous la forme de mécanismes de gestion de processus ou de tâches (*tasking*) (dans Unix : l'interruption $\wedge C$, la suspension $\wedge Z$, la reprise fg ou en tâche de fond bg). On peut noter que même si les processus peuvent avoir une terminaison bien définie, ces opérations ne la considèrent pas.
- dans les systèmes réactifs, où une formalisation de ces mécanismes se trouve dans les opérations de préemption [21],
- dans les systèmes de simulation et d'animation, où on modélise aussi bien les caractéristiques mécaniques et cinématiques des éléments simulés, que des comportements discrets [10],
- dans les systèmes de traitement du signal, ou en particulier de vision, où la reconnaissance des caractéristiques géométriques d'un ensemble d'objets fait intervenir le séquencement de tâches d'acquisition [75],
- dans les systèmes robotiques où on séquence, sur l'occurrence d'événements internes ou externes, des lois de commande [24],
- dans les systèmes de contrôle automatisé d'expériences spatiales, embarqués à bord de stations ou satellites,
- dans les systèmes de contrôle/commande en général, où il s'agit justement de combiner des phases continues et des changements de phases discrets [70, 90].

Par ailleurs, des exemples de mixité de comportements comparables se trouvent dans des domaines moins directement liés aux systèmes hybrides ou à sécurité critique :

- dans les systèmes multi-média, où les tâches de gestion d'affichages statiques, ou de données en mouvement ou de vidéo ou de son doivent être coordonnées, enchaînées et synchronisées sur l'occurrence d'événements [57];
- dans les systèmes de représentation de connaissance, où les raisonnements complexes font intervenir des hiérarchies et des enchaînements de tâches et sous-tâches [86],
- dans les systèmes combinant des fonctionnalités de vision par exemple (suivi d'éléments et de personnes dans une scène, reconnaissance de mouvement et de geste, ...) pour rendre des services plus complexes [81].

1. Des mécanismes où un contrôle de l'activité d'un processus se fait depuis l'*intérieur* existent aussi sous la forme de mécanismes d'exception, ou de gestion de points de reprise.

Structures de langages de programmation. Dans les langages de programmation, cette problématique a donné lieu à la conception de formalismes divers, à différents niveaux d'abstraction, notamment vis-à-vis de la prise en compte dans le modèle du comportement des tâches elles-mêmes.

Séquençement de tâches externes. On trouve par exemple :

- les langages de commande des systèmes d'exploitation, ou les fonctions des systèmes de gestion de processus (ou *tasking*),
- les langages de coordination, qui considèrent l'organisation de la coopération entre des processus définis indépendamment. Un exemple en est MANIFOLD [8], où des processus connus par leurs entrées et sorties sont interconnectés au moyen de flots, démarrés ou arrêtés par des processus coordinateurs.
- les langages réactifs dits orientés contrôle plutôt que données : on y décrit les réactions à l'environnement en termes de lancement, suspension, reprise et interruption de tâches extérieures. Des exemples en sont des langages textuels comme ELECTRE [87], où on décrit l'activation de modules externes, et ESTEREL [22] et son mécanisme de gestion de tâche *exec* [27]. Dans ces langages est menée une étude systématique de la préemption, menant à des classifications des opérateurs [21].

Ces langages considèrent un séquençement en interaction relativement faible ou restreinte avec les tâches séquencées. Si on considère la structure hiérarchique des langages réactifs, ainsi que dans ARGOS [71] ou STATECHARTS [50], les structures de préemption de sous-programmes peuvent elles-mêmes être vues comme du séquençement de tâches définies par ces sous-programmes. On a alors une interaction plus riche entre les tâches et leur séquençement ou préemption; on a alors aussi le moyen d'avoir un modèle du comportement complet, sur lequel faire des analyses plus fines.

Toutefois, au niveau où on invoque une tâche définie dans un formalisme différent, vue comme externe, elle n'est considérée que par l'intermédiaire d'événements de contrôle génériques (début, terminaison, ...), en ignorant son comportement spécifique.

Multi-formalisme et séquençement de tâches flot de données. Une approche multi-formalisme, comme en l'occurrence le séquençement de tâches flot de données, permet d'aller au-delà de cette frontière, et de restaurer une situation similaire à celle évoquée au paragraphe précédent, où l'inter-opération (au sens où les interactions entre tâches et séquençement sont bi-directionnelles et explicites) prend une place plus importante. On considère des tâches définies par des réseaux de processus à flots de données, et un séquençement impératif, comme dans :

- l'intégration d'ARGOS [71] et de LUSTRE [49], permettant l'association à un état d'ARGOS d'un nœud LUSTRE [56],
- les automates de modes [74], qui combinent de façon compositionnelle les équations différentielles et les automates hiérarchisés,
- dans STATEMATE, le contrôle des activités d'ACTIVITYCHARTS à partir de STATECHARTS [51],
- le système de programmation de système robotiques ORCCAD [24], où on combine divers niveaux de spécification, notamment :
 - les tâches-modules, éléments calculatoires de base dans le flot de données,
 - les tâches-robot, flot de données entre modules, avec un contrôle de base (démarrage, arrêt, exceptions),
 - les procédures-robot, séquençement de tâches-robot.

Assistance à la conception. Concernant la sûreté critique des systèmes qui nous intéressent, on souhaite pouvoir proposer des méthodes de conception comprenant la validation des systèmes, et utilisant des outils d'assistance qui permettent l'analyse de leur correction; ces outils reposent eux sur l'utilisation de modèles formels du comportement des contrôleurs. Des environnements de conception comme celui proposé par le projet SACRES, décrit en Section 2.2.1, fondent leur architecture sur cette articulation.

Modèles formels. Certains langages mentionnés plus haut sont définis avec une sémantique formulée dans des modèles formels, reposant de façon générale sur la théorie des systèmes à événements discrets ou les automates à états. Cette approche, utilisée dans le domaine des langages de conception des applications embarquées [44], l'est aussi ailleurs, comme dans les protocoles de communication.

Ces définitions permettent, pour un programme écrit dans un tel langage, de construire (automatiquement, par compilation) une représentation du système de transition décrivant une abstraction de son comportement. Il existe diverses façons de constituer cette représentation; elle n'est pas nécessairement énumérative de l'espace des états : des représentations symboliques et optimisées peuvent être utilisées.

Outils automatisés. Cette formalisation a pour avantage de fournir une base concrète de construction d'outils d'analyse des programmes, prenant en compte leur sémantique, dans leurs aspects statiques comme dans leur comportement dynamique. Sur une telle représentation, les algorithmes de calcul et de décision peuvent être mis en œuvre. Ils servent de fonctionnalités de base à :

- la compilation (au sens de combinaison des fonctionnalités suivantes, adaptée à l'objectif poursuivi, qu'il soit d'exécution ou de vérification) ,
- la vérification de cohérence (recherche de cycles causaux),
- la vérification de propriétés invariantes vis-à-vis de l'état du programme,
- la vérification de propriétés sur la dynamique de leur comportement,
- l'optimisation (qui peut se caractériser différemment suivant l'objectif, par exemple la vérification ou l'exécution),
- la production d'exécutables corrects par construction du code généré,
- la validation du code généré vis-à-vis de la spécification,
- la simulation,
- la synthèse de contrôleur à événements discrets, à partir de modèles sous-spécifiés laissant lieu à restriction par contraintes,
- d'autres combinaisons voient le jour, offrant des fonctionnalités nouvelles, ou renouvelant les pratiques ; on peut penser notamment à la génération automatique de jeux de test à partir de modèles du comportement du programme à tester.

Ces outils, rassemblés dans des environnements de conception et réalisation, répondent au besoin d'assistance à la conception des systèmes à sécurité critique. On peut noter que les fonctionnalités réalisées par chacun d'eux ont intérêt à être découplées et interfacées pour qu'elles puissent inter-opérer au mieux, et que les opérations qu'elles réalisent sur les modèles puissent être réutilisées dans des applications différentes. Par exemple l'optimisation du code généré peut bénéficier de l'analyse des comportements dynamiques, qui déterminerait des états inaccessibles. Il est important que ces environnements soient ouverts, ainsi que les outils eux-mêmes, au sens où ils soient capables d'échanger des représentations du programme étudié, sous forme d'un format commun. Un tel ensemble de formats a été défini pour les langages synchrones, où le format déclaratif DC (*Declarative Code*) est la version simple, sous-ensemble syntaxique et sémantique, de DC+ [102]. Une approche plus profonde dans ce sens consiste à construire les outils de manière à offrir à d'éventuels utilisateurs avancés une interface de programmation d'application (*API, Application Programming Interface*) dans laquelle ils puissent spécifier leurs propres algorithmes de calcul ou transformation.

Méthodes de conception. L'utilisation de ces outils est soutenue par des méthodes qui tentent notamment de mener à une utilisation optimisée des outils, et de limiter l'expertise requise quant aux modèles formels sous-jacents². Ceci peut se traduire par :

- l'adaptation à des langages de spécification existants, éventuellement spécifiques de domaines d'application, parfois dits «langages-métier». Des exemples en sont les langages de programmation des systèmes de contrôle industriel, spécialisés pour les systèmes à base d'automates programmables industriels (API) (*PLC, Programmable Logic Controllers*) comme le GRAFCET [32], ou ceux de la norme IEC 1131 [29].
- Outre la spécialisation, on peut aussi être amené à étudier des langages plus élaborés, dans le sens où ils présentent plus de constructeurs que des langages réduits à l'essentiel des primitives, tout en étant réductibles au même modèle de base. Un exemple en est STATEMATE [51], dans lequel on trouve, surajoutés au principe de départ d'automates parallèles hiérarchiques, un grand nombre de constructions de différents styles, concernant les structures de données comme le contrôle, pour le confort des programmeurs et la structuration des spécifications.
- l'élaboration de méthodes de spécification des propriétés à vérifier qui évitent l'emploi de langages éloignés de la culture des utilisateurs. On peut par exemple utiliser des motifs de propriétés simplifiés (éventuellement jusqu'à être générés à partir de quelques paramètres donnés par l'utilisateur), ou la technique des observateurs où le langage de programmation lui-même est utilisé pour construire un processus reconnaissant la propriété en définissant un état la caractérisant.

2. Typiquement, il paraît souhaitable d'éviter que la conception du contrôleur ne nécessite un doctorat en informatique spécialisé dans les méthodes formelles, ce qui n'est guère exigible d'un ingénieur, en plus de la compétence en automatique ou traitement du signal liée à la nécessaire connaissance du procédé à contrôler.

En résumé, la problématique à laquelle contribue ce travail est alors essentiellement de :

- définir des structures mixtes de langage de programmation des systèmes hybrides, par séquençement (discret) de tâches flot de données (continues échantillonnées);
- permettant l'assistance à la conception par l'utilisation d'outils d'analyse formelle, en particulier par l'accès aux outils liés aux techniques de la programmation synchrone, notamment ceux connectés à SIGNAL et DC+;
- en confrontation avec des applications :
 - en robotique : vision, téléopération,
 - dans les systèmes de contrôle discret : automatisme de transformateur, contrôleur de cellule d'assemblage.

1.3 Approche suivie et premières contributions

1.3.1 Approche suivie

Ma contribution à cette problématique a suivi une approche qui consiste en :

- des propositions de langages de spécification ou programmation de séquençement de tâches, dans des contextes de robotique ou de programmation synchrone,
- la définition de ces langages reposant sur des modèles formels, qui sont utilisés :
 - pour la description précise et rigoureuse de leur sémantique,
 - pour la construction d'outils concrets d'analyse des programmes, par simulation, interprétation ou par accès à des outils de vérification
- le traitement d'expérimentations, dans les domaines d'application au regard desquels la conception des langages est faite, en particulier la programmation des systèmes de contrôle/commande.

Le choix de ce domaine d'application a l'avantage d'être typique de la problématique, puisqu'il s'agit de systèmes à la fois hybrides et à sécurité critique, et d'offrir un cadre expérimental auquel confronter la validité des choix de primitives et de structures des langages, ou de fonctionnalités offertes par les outils d'assistance. Ce choix ne nuit pas pour autant à la possibilité d'applications plus générales; on reste en particulier indépendant d'aspects spécifiquement robotiques comme la programmation en termes de configurations géométriques de systèmes mécaniques articulés, domaine de langages comme LM.

Ce travail s'est déroulé en plusieurs phases, liées par la problématique, qui apparaissait sous diverses formes, et par l'enrichissement des concepts qui s'en dégageaient, plutôt que par la réutilisation directe de résultats techniques. J'exposerai dans la suite de ce chapitre brièvement comment mes activités de recherche doctorale et post-doctorale ont constitué une exploration du sujet, dont sont ressorties les notions essentielles développées plus tard. Ensuite, dans les chapitres 2 et 3, j'exposerai plus en détail mes contributions des dernières six ou sept années, concernant la programmation sûre des systèmes de contrôle/commande, et plus particulièrement le séquençement de tâches flot de données dans les systèmes réactifs, et ses applications au contrôle de systèmes hybrides : ces travaux sont traités dans le cadre cohérent des langages synchrones, et de SIGNAL en particulier.

La Section 1.3.2 présente mes premiers travaux dans la problématique, en thèse à l'IRISA³ à Rennes. Ils ont concerné un langage impératif et réactif de spécification de plans d'actions robotiques [101, 97]. Il s'agissait de pouvoir décrire les missions d'un système robotique à un haut niveau d'abstraction, indépendamment de l'architecture d'exécution, mais aussi du choix d'algorithme de contrôle effectif. Une première version a été définie à des fins de simulation de missions en téléopération. Une modélisation logique et temporelle du langage le définissait, et servait de support à un prototype de simulateur. Une application à la téléopération a concerné des missions d'un système de manipulation de MATRA-ESPACE. Des développements ultérieurs de cette approche ont concerné, vers le niveau de l'exécution (en début de post-doctorat ERCIM⁴ à l'INRIA⁵ Sophia-Antipolis), un langage de programmation d'applications, avec encodage et traduction vers ESTEREL [30], et vers le niveau décisionnel (en milieu de post-doctorat ERCIM à la GMD⁶ à Sankt Augustin, en Allemagne), la génération de plans temporels [95].

3. Institut de Recherche en Informatique et Systèmes Aléatoires (regroupant l'INRIA-Rennes, l'UPRESA 6074 du CNRS, l'Université de Rennes 1 et l'INSA de Rennes. Site web : <http://www.irisa.fr>

4. *European Research Consortium in Informatics and Mathematics* : Consortium Européen de Recherche en Informatique et Mathématiques, regroupant à sa fondation l'INRIA, la GMD et le CWI, rejoints depuis par des institutions représentant divers pays d'Europe. Site web : <http://www.ercim.org>

5. Institut National de Recherche en Informatique et Automatique. Site web : <http://www.inria.fr>

6. *Gesellschaft für Mathematik und Datenverarbeitung GmbH* : Centre National de Recherche en Mathématiques et Informatique Allemand. Site web : <http://www.gmd.de>

Jusque là, les tâches sont considérées comme extérieures, même si les trois niveaux d'exception dans ORCCAD [30, 24] sont une tentative d'affiner l'inter-opération entre tâches et séquencement. Or le caractère continu de ces tâches de commande et régulation suggère l'utilisation de flot de données; leur séquencement implique alors la re-configuration de réseaux flot de données. Cet aspect est abordé dans mes travaux (en fin de post-doctorat ERCIM au CWI⁷ à Amsterdam, Pays-Bas) concernant MANIFOLD, un langage événementiel de configuration de réseaux flots de données. Il s'agit de lui proposer une définition formelle sous la forme d'une sémantique opérationnelle structurée (SOS); celle-ci a servi de support à un interpréteur [9]. Ces travaux sont décrits de façon synthétique en Section 1.3.3.

1.3.2 Séquencement de tâches externes

Un langage réactif de planification pour la téléopération.

Ce travail s'est fait sous la direction de Lionel Marcé, et avec Jean-Christophe Paoletti. Les motivations concernent ici la contribution à un environnement de programmation robotique au niveau tâche, où il s'agit du séquencement d'actions vues sous l'angle de leurs contraintes logiques. Elles peuvent concerner divers aspect de la commande, cinématique, géométrie et trajectoire, poursuite de cible ou évitement d'obstacle, mouvements coordonnés de plusieurs actionneurs. On les considère comme blocs de base des plans d'actions, où on se concentre sur les aspects logiques et qualitatifs, plutôt que quantitatifs. On s'intéresse aussi plus particulièrement à la robotique de coopération (par exemple de téléopération), où il s'agit de partage des décisions entre automatismes et opérateur humain plutôt que de robotique autonome. Dans ce dernier domaine, il existe des techniques de planification, reposant sur des modèles logiques de représentation de l'action et d'environnements changeants.

Dans ce contexte, le travail a consisté en la conception d'un modèle logique et temporel de plans d'actions de robots, sous la forme d'un langage de planification impératif, comportant des structures de contrôle réactives [101, 97]. La définition formelle des opérateurs du langage est en termes de la logique temporelle à intervalles de Allen, décrivant les arrangements temporels des actions et sous-plans. Une assistance à la conception des plans est offerte sous la forme d'un prototype de simulateur logique et temporel, fondé sur la définition formelle. Il a servi dans une application à l'assistance à l'opérateur en planification de missions de robotique spatiale (en collaboration avec MATRA ESPACE).

Langage. Les actions étant considérées au niveau tâche, on les décrit au niveau logique, comme en planification, par des prédicats caractérisant l'état de l'environnement dans lequel elles interviennent.

Les actions primitives sont définies en termes de leurs contraintes logiques :

- préconditions, qui doivent être vérifiées avant l'exécution de l'action,
- conditions, qui doivent être vérifiées sur toute la durée de l'action,
- et effets (ou postconditions) satisfaits après l'exécution de l'action (ils peuvent néanmoins l'être dès avant la fin de celle-ci).

Vis-à-vis du temps que prend l'exécution des actions, on considère deux types d'actions :

- les actions à durée déterminée : leur durée est définie intrinsèquement (exemple : action vide (sans pré-, post- ni conditions) de durée nulle),
- les actions à durée indéfinie : leur durée est déterminée par l'environnement dans lequel elles s'exécutent, c'est-à-dire la structure de contrôle englobante (exemple : l'attente, action vide sans durée prédéfinie).

Les opérateurs de séquencement sont classiques ou réactifs :

Les opérateurs indépendants de l'environnement définissent le séquencement seulement en terme des tâches ou sous-plans séquencés.

- des opérateurs aux interprétations classiques comprennent :
 - la séquence $seq(P1, P2)$ qui démarre $P2$ à la fin de $P1$,
 - la conditionnelle $cond(C, P1, P2)$ qui démarre, si C est vérifiée, $P1$, sinon $P2$,
 - le parallélisme $par(P1, P2)$ qui démarre $P1$ et $P2$ ensemble, et termine à la fin du dernier terminé.
- l'abstraction de sous-plans, ou décomposition d'action en sous-plan, qui permet de définir une action par un plan, en vue de le réutiliser; cette structure permet la définition de structures de contrôle dérivées.

⁷ Centrum voor Wiskunde en Informatika : Centre National de Recherche en Mathématiques et Informatique Néerlandais. Site web : <http://www.cwi.nl>

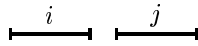
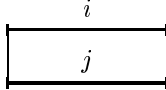
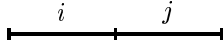
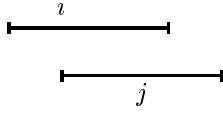
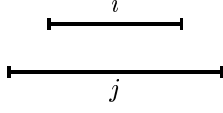
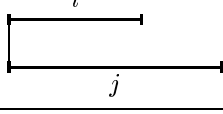
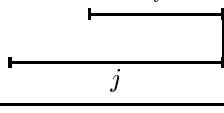
relation	symbole	réciroque	exemple
i before j	b	a	
i equals j	=		
i meets j	m	mi	
i overlaps j	o	oi	
i during j	d	di	
i starts j	s	si	
i finishes j	f	fi	

FIG. 1.1 – Langage de planification : les treize relations temporelles entre intervalles de Allen

- des opérateurs associés aux actions à durée indéfinie, qui servent à déterminer de diverses manières cette durée. Un exemple en est *as-long-as* qui associe l'action à durée indéfinie à un plan en parallèle, dont la fin détermine la fin de l'action.

Les opérateurs de réaction à l'environnement font eux, à la différence des précédents, explicitement référence à des prédicats caractérisant l'état de l'environnement d'exécution du plan :

- au niveau des actions à durée indéfinie, on peut lier la durée de son exécution à la vérification d'une condition sur l'environnement, par exemple par *while-c(C,A)*,
- au niveau des plans, on peut définir des règles de réaction par *when(C,P)*, où le plan P est déclenché quand la condition C est vérifiée.

Modèle. Le modèle utilise comme formalisme de base la logique temporelle à intervalles de Allen [2]. Ces intervalles sont liés par des relations, rappelées en Figure 1.1. Une relation disjonctive est notée $I_1 (r_1 r_2) I_2$. Sur ces bases, une table de transitivité a été construite, permettant de déterminer, connaissant deux relations r_1 et r_2 entre trois intervalles I_1 , I_2 et I_3 , c'est-à-dire $I_1 r_1 I_2$ et $I_2 r_2 I_3$, une troisième relation r_3 obtenue transitivement : $I_1 r_3 I_3$. Dans cette logique, la vérité d'une proposition P sur un intervalle de temps I est notée : $vrai(I, p)$. On aura besoin de se donner un prédicat $dure(I, D)$, associant une durée quantitative D à un intervalle I . Les techniques permettant de mettre en œuvre efficacement un raisonnement temporel de cette sorte ont été étudiées en profondeur, notamment en ce qui concerne l'algorithmique associée et sa complexité, les relations avec les algèbres de relations entre instants, et la mise en œuvre d'outils de gestion de systèmes de contraintes temporelles [83].

La modélisation a consisté en la définition des actions et des opérateurs de séquençement en termes de cette logique. L'exécution des actions et des plans sur un intervalle est représentée par :

exec(Intervalle, Action-ou-Plan)

Sa signification est que l'action ou le plan est exécuté sur l'intervalle I . Par exemple, une action primitive est un quintuplet, noté sous forme de terme :

action(A, Durée, Préconditions, Conditions, Effets).

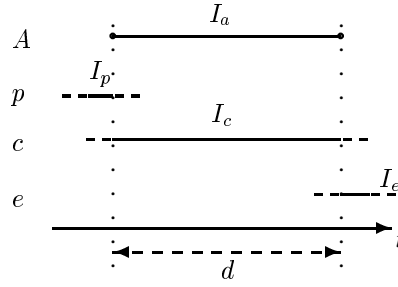


FIG. 1.2 – Langage de planification : une action primitive dans le temps.

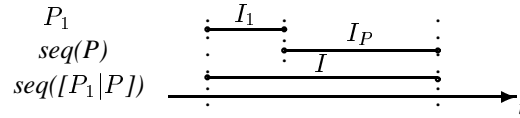


FIG. 1.3 – Langage de planification : la séquence dans le temps.

La façon dont cette action s'inscrit dans le temps est définie par :

$$\begin{aligned} exec(I_a, A) \iff & \text{dure}(I_a, \text{Durée}) \\ & \wedge (\forall p \in \text{Préconditions}) (\exists I_p) (\text{vrai}(I_p, p) \wedge I_a \text{ (mi oi f d) } I_p) \\ & \wedge (\forall c \in \text{Conditions}) (\exists I_c) (\text{vrai}(I_c, c) \wedge I_a \text{ (s f d =) } I_c) \\ & \wedge (\forall e \in \text{Effets}) (\exists I_e) (\text{vrai}(I_e, e) \wedge I_a \text{ (d s o m) } I_e) \end{aligned}$$

et illustrée par la Figure 1.2.

Quant aux opérateurs, on donne l'exemple de la séquence, notée :

$$seq(\text{ListeP})$$

où la liste de sous-plans *ListeP* a pour forme $[P_1|P]$, P_1 étant le premier sous-plan dans la liste, et P le reste de la liste. Cet opérateur de contrôle indique que les sous-plans de la séquence sont exécutés l'un après l'autre, dans l'ordre de la liste, à l'intérieur d'un intervalle global I , comme illustré en Figure 1.3 :

$$exec(I, seq([P_1|P])) \iff (\exists I_1, I_P) \quad (I_1 \text{ s } I \wedge I_1 \text{ m } I_P \wedge I_P \text{ f } I \wedge exec(I_1, P_1) \wedge exec(I_P, seq(P))).$$

Outil. L'outil construit sur la base de ce modèle est le prototype de simulateur TEMPUS FUGIT. C'est une mise en œuvre simple et partielle (ne reprenant notamment pas la réaction à l'environnement). Ses objectifs sont l'assistance à la planification et à la préparation de missions d'un système téléopéré. Il s'agit de représenter le comportement d'un robot dans sa disposition temporelle et ses effets logiques sur les propriétés décrivant l'environnement. La méthode pour cela consiste à compléter une base de faits temporels, associant prédicats et intervalles de temps, en y transcrivant les conséquences de l'exécution d'un plan d'actions. La maquette de TEMPUS FUGIT a été réalisée en QUINTUS PROLOG sur SUN 3/60.

Une application à des missions d'un système téléopéré, étudié par MATRA-ESPACE, a servi à illustrer et valider l'utilisabilité du langage et de l'outil.

Suites. Ces travaux ont eu des répercussions sur la suite de mes activités, notamment dans le cadre du séquencement de tâches robot dans ORCCAD, et de la génération de plans temporels (décrits plus loin). Ils ont en trouvé aussi dans des travaux menés en parallèle des miens sur l'exécution réactive d'une partie du langage, notamment dans les thèses de Jean-Christophe Paoletti [85] (à l'IRISA), et aussi, indirectement et plus tard, Erwan Le Rest [68] (UBO) et Jean-Luc Fleureau [39] (ENSIETA).

Thèse de Doctorat de l'Université de Rennes I, mention Informatique (13 Juillet 1990) : *Représentation en logique temporelle de plans d'actions dotés d'une structure de contrôle impérative ; application à l'assistance à l'opérateur en téléopération*. Jury : Président : J.P. Banâtre; Rapporteurs : C. Barrouil, M.O. Cordier; Examineurs : G. André, B. Espiau, M. Ghallab, L. Marcé (Directeur), E. Sandewall. [101]

journal [97], conférences (dont [100, 94]), rapports de recherche.

outil : prototype de simulateur TEMPUS FUGIT, en Prolog.

encadrements : co-encadrement avec Lionel Marcé de deux stages de DEA et co-encadrement avec Jean-Christophe Paoletti d'un projet de Maîtrise /4è année INSA Informatique, Rennes

collaboration : étude de cas avec MATRA-ESPACE (Guy André).

Développements de cette approche.

Ces travaux ont eu des suites dans deux directions complémentaires, ayant trait l'une à l'aspect programmation robotique, l'autre aux modèles temporels et d'action :

Un langage de programmation d'applications robotiques qui reprend certaines des structures de contrôle, et certains aspects des actions (pré- et post-conditions), pour proposer une abstraction au niveau des tâches dans la programmation de systèmes robotiques, avec modélisation dans un langage synchrone,

un générateur de plans temporels qui utilise une description logique et temporelle des actions pour synthétiser des ordres partiels d'actions remplissant un objectif.

Un langage de programmation d'applications robotiques. Dans le contexte d'ORCCAD [24], ces travaux se situent parmi les travaux préliminaires ayant ensuite mené aux concepts plus stables actuels. Ils ont été menés avec Ève Coste-Manière, Bernard Espiau et Daniel Simon.

Il s'agit de la spécification et de l'exécution de comportements de systèmes robotiques au niveau de la synchronisation et du séquençement d'actions considérées comme élémentaires. Il s'agit donc d'intervenir à un niveau intermédiaire, parmi beaucoup de travaux aux niveaux :

- plus «bas», au sens où il s'agit du contrôle et de la commande des robots par des lois éventuellement référencées capteur : ces comportements sont ceux qui, encapsulés dans une structure de *tâche-robot*, sont considérés comme actions élémentaires.
- plus «haut», au sens où il s'agit d'algorithmes décisionnels qui produisent automatiquement les séquençements d'actions : ces travaux concernent la planification et le raisonnement sur l'action.

L'aspect critique de la sécurité des systèmes robotiques considérés donne lieu ici aussi à une réponse à la nécessité d'outils d'analyse. En l'occurrence, on a recours à l'utilisation d'un langage synchrone : ESTEREL [22, 27], et des outils qui lui sont associés. La notion de synchronisme concerne la composition de processus dans le langage : des processus parallèles communiquent de façon instantanée pour procéder à une réaction globale à l'environnement (cette notion sera présentée plus avant en Section 2.1). Cette composition donne lieu à une formalisation et à des calculs qui fondent la compilation de programmes ESTEREL en automates déterministes, où les transitions peuvent comporter des actions simultanées comme le démarrage de plusieurs tâches. L'environnement de programmation associé à ESTEREL comporte divers outils, parmi lesquels un compilateur qui procède à la vérification de la consistance des synchronisations, des outils de vérification de propriétés sur la dynamique du programme, un générateur de code séquentiel le mettant en œuvre. Il existe aussi des résultats concernant des schémas d'exécution connectant le programme synchrone à son environnement asynchrone [7].

Dans ce contexte, on souhaite proposer à des utilisateurs qui ne soient pas des experts de programmation (*a fortiori* synchrone) des structures faciles à manipuler, et qui correspondent aux entités de base de ce type d'applications, tout en assurant l'accès aux outils des environnements de programmation. J'ai donc contribué à proposer l'encapsulation des lois de commande dans des *tâches-robot* dotées d'un contrôle (au sens discret) de base, qui gère le démarrage, le traitement d'exceptions, l'arrêt ; j'ai proposé ensuite un langage de programmation d'application pour séquençer ces tâches-robot [30], qui reprend certains aspects du langage de spécification exposé plus haut [97, 101].

Langage. Ses opérateurs de séquençement manipulent des conditions et des actions.

- Les *conditions* sont des éléments de base de ce langage, produites par des capteurs logiques ou bien des observateurs les calculant à partir de mesure quantitatives.

- Les actions sont les *tâches-robot*, qui encapsulent une loi de commande élémentaire (par exemple suivi de trajectoire, assemblage contrôlé en force, tâche référencée vision) en lui donnant un comportement réactif de base, défini en terme de conditions :
 - des préconditions, dont on attend qu’elles soient vérifiées pour lancer la loi de commande,
 - des observateurs, qui évaluent pendant l’exécution de la tâche une condition, avec associée à chacun une procédure de traitement,
 - des postconditions, dont la vérification permet la terminaison de la tâche; on peut noter qu’une loi de commande n’a pas de terminaison pré-définie en elle-même,
 - une procédure de commande, qui effectue un pas d’évaluation de la commande, en suivant la loi.
- Le séquençement se fait par les opérateurs du langage de programmation d’applications (LPA) inspirés en partie par un sous-ensemble de ceux proposés dans mes travaux antérieurs [101, 97] :
 - la séquence SEQ enchaîne un plan immédiatement à la suite de l’autre,
 - le parallélisme PAR démarre les sous-plans ensemble, et termine avec le dernier d’entre eux,
 - le branchement conditionnel COND évalue une condition et choisit un sous-plan en conséquence,
 - la boucle LOOP est une itération infinie sur un sous-plan,
 - des structures de contrôle réactives lient le démarrage d’un plan à l’attente d’une condition (par exemple $\text{WHEN}(C, P)$),
 - le traitement d’exception est considéré à trois niveaux pré-établis :
 - à l’intérieur des tâches, il s’agit du traitement associé aux observateurs,
 - au niveau d’un sous-plan, on peut associer un traitement à une exception en stoppant sa propagation,
 - au niveau global de l’application, on peut stopper le plan (en mettant le système dans un état stable) et rendre le contrôle à un opérateur humain.

Modèle. J’ai donné pour chacune de ces structures (tâches-robot et structures de contrôle) une modélisation par codage en ESTEREL, utilisé comme langage formel en considérant que cela présentait les avantages combinés du langage structuré et de sa sémantique. Chaque tâche-robot est dotée d’un contrôleur générique qui a le comportement décrit plus haut. Les opérateurs de séquençement sont traduits dans des contextes ESTEREL, qui sont construits en utilisant les structures de préemption d’ESTEREL pour le traitement différencié des exceptions.

Outil. Ce modèle a donné lieu à un prototype de traducteur/préprocesseur du langage de programmation d’applications vers ESTEREL (en Prolog).

Suites. Ce travail préliminaire dans ORCCAD a eu des suites dans les structures de *procédures-robot* proposées dans la thèse de Konstantinos Kappellos [58], répondant à celles de *tâche-robot* et de *tâche-module* à ce niveau du séquençement. Dans cette lignée a ensuite été défini MAESTRO [31], qui fournit un langage-métier à l’utilisateur, qui n’a plus besoin de connaître ESTEREL pour programmer une application.

Activités en programmation robotique

publications : conférence [30], rapports de recherche.

outil : prototype de traducteur en ESTEREL, écrit en Prolog

collaborations : première partie de post-doc ERCIM, INRIA Sophia Antipolis (France), projet PRISME (sept. 1990–mars 1991).

Co-organisation de la *Journée sur la programmation des robots au niveau tâche : du raisonnement temporel au temps réel*, le 14 Mars 1991 à l’INRIA Sophia-Antipolis.

Un générateur de plans temporel. Ce travail s’est fait avec avec Joachim Hertzberg. Vis-à-vis de la programmation robotique telle qu’envisagée jusqu’ici, et ligne globale de mes travaux, il s’agit du niveau décisionnel, «haut», au sens où la génération de plans doit produire, automatiquement, un séquençement d’actions. La spécification de comportement se fait alors sous forme d’objectifs à remplir par le plan à générer. En tant que fonctionnalité, ceci semble comparable à la synthèse de contrôleur étudiée dans le domaine des systèmes à événements discrets [25], avec laquelle le lien serait à étudier (voir Section 4.2).

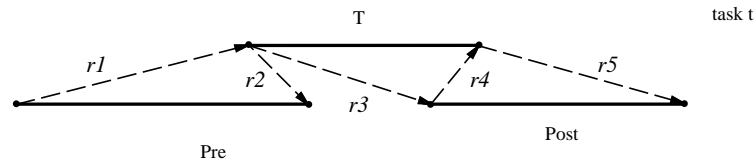


FIG. 1.4 – Planification temporelle : les contraintes temporelles pour une action.

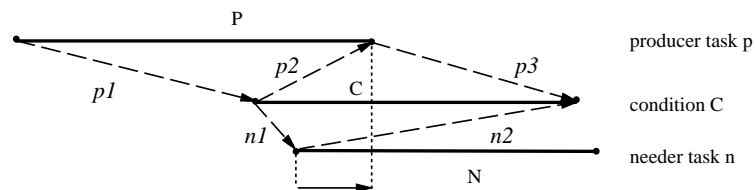


FIG. 1.5 – Planification temporelle : la contrainte pour les conditions, à la fois post- et précondition.

Vis-à-vis du domaine de l'intelligence artificielle (IA), la génération de plans, ou planification, est une des motivations pour les travaux sur la gestion de contraintes temporelles en IA. Mon travail concerne ici l'intégration des deux, plus particulièrement :

- la planification non-linéaire [53], utilisée comme algorithme de génération de base. Un plan est un ensemble d'actions partiellement ordonné. Chaque action est définie par des préconditions et postconditions (ainsi qu'une durée dans le cas temporel). Le but ou objectif est un ensemble de conditions qui doivent être vérifiées à l'issue du plan. Un tâche ou action est considérée comme générant ses postconditions, et requérant ses préconditions. Ceci établit des dépendances entre actions requérant des préconditions et actions les générant. Il y a ainsi un ordre temporel entre actions, et un ordre de dépendance, ou causal, inclus. Cette dépendance est remise en cause si une action détruisant la condition (c'est-à-dire ayant pour effet de la rendre fausse) a lieu entre le producteur et l'action qui la requiert : on dit qu'il y a conflit.
- la gestion de contraintes temporelles quantitatives entre des instants [83]. Une base de données temporelles contient des faits associant une proposition à un intervalle de temps sur lequel elle est vérifiée ; cet intervalle est défini par ses instants de début et de fin, et une relation temporelle entre eux, donnant l'intervalle de valeur $[min, max]$ dans lequel se trouve la durée les séparant. La gestion des contraintes consiste à maintenir, à chaque modification de la base de données quant aux contraintes, une image minimisée des relations, et la consistance de l'ensemble. Bien que ne faisant pas de déductions logiques quant aux faits, la gestion de la base peut faire intervenir une notion de faits contradictoires, pour lesquels des intervalles se chevauchant donnent lieu à une coupure de persistance (*persistence clipping*), par contrainte entre la fin de l'intervalle de l'un et le début de celui de l'autre.

Dans leur combinaison dans un planificateur temporel, j'ai utilisé cette notion de coupure de persistance pour gérer les conflits.

Langage. La représentation des actions et des plans dans le gestionnaire de contraintes temporelles joue le rôle des langages vus précédemment : la description du comportement et l'accès aux calculs formels sous-jacents. Les actions sont définies par des pré- et post-conditions, et la façon dont elles s'inscrivent dans le temps (en termes de contraintes dans MTMM) est illustrée par la Figure 1.4, où les lignes pleines représentent les intervalles des faits, et les pointillés les contraintes r_i entre instants aux bornes des intervalles. On utilise en fait uniquement des contraintes de type *avant* et *après*, codées respectivement par $[0, +\infty]$ et $[-\infty, 0]$.

Modèle. Les relations entre actions sont représentées par des contraintes qui permettent la production de la postcondition avant la fin de l'action, et son utilisation en tant que précondition après le début de l'action; ainsi on autorise un parallélisme d'exécution entre ces deux actions, comme l'illustre la Figure 1.5 où sont montrées en pointillés les contraintes p_i et n_i , ainsi que l'intervalle d'exécution parallèle possible du producteur et de l'action requérant sa précondition, entre les ligne pointillées.

Les conflits sont traités à l'aide du gestionnaire de contraintes temporelles, et en particulier de la coupure de persistance ; une action destructrice d'une condition est productrice d'un fait contradictoire : la coupure de persistance

posera les contraintes qui feront que la condition coupée ne pourra servir de précondition à une action ultérieure qui en aurait besoin, et pour laquelle il faudra générer une instance d'action productrice.

Outil. Le prototype TRIPTIC utilise ces concepts, en présentant un algorithme de génération de plans non-linéaire au-dessus d'un gestionnaire de contraintes temporelles quantitatives sur des instants (MTMM) [83], (en Lisp).

Activités en planification

publications : journal [95], conférence, rapports de recherche.

outil : prototype de planificateur, interfacé avec le gestionnaire de contraintes temporelles MTMM, en Lisp

collaborations : deuxième partie de post-doc ERCIM, GMD, Sankt Augustin (Allemagne), qwertz Projekt, Artificial Intelligence Research Division (mars-sept. 1991).

1.3.3 Réseaux flots de données dynamiques.

Au long des travaux précédents, il est apparu qu'il serait intéressant de pouvoir considérer les tâches séquencées plus en détail. Plutôt que de ne les considérer que par leur démarrage et arrêt (que ce dernier soit dû à la tâche elle-même, ou imposé de l'extérieur, par préemption), ou même de distinguer des types de signaux de contrôle plus élaborés, comme les trois types d'exceptions considérés dans ORCCAD, on peut vouloir se donner un cadre dans lequel spécifier et analyser les contenu des tâches en même temps que leur séquençement, et leur interaction. Or le caractère continu de ces tâches de commande et régulation suggère l'utilisation de flot de données; leur séquençement implique alors la reconfiguration de réseaux flot de données. Le multi-formalisme qui nous intéresse est donc du type combinant séquençement et flot de données.

Je présente ici brièvement mes travaux à ce sujet en lien avec le langage MANIFOLD, dont je tire quelques enseignements qui s'avèreront pertinents dans la suite de mes travaux.

Un langage événementiel de configuration de réseaux flots de données. Ces travaux ont été menés avec Paul ten Hagen, Fahrad Arbab et Ivan Herman. Le cadre de cette étude de modélisation de reconfiguration dynamique, sur l'occurrence d'événements, de réseaux de flots de données entre opérations, est le langage parallèle MANIFOLD [8]. J'ai travaillé à la spécification formelle de MANIFOLD, pour à la fois clarifier sa spécification informelle, et dégager certains aspects essentiels de son comportement. Une sémantique en est donnée sous forme de règles de sémantique opérationnelle structurale (SOS). Celle-ci est codée dans un interpréteur qui fournit ainsi un outil de simulation de programmes [93]. J'ai défini un langage simplifié, MINIFOLD, pour extraire certains aspects essentiels de ce type de langage [9].

Langage. MANIFOLD est un langage de coordination pour l'orchestration des communications entre des processus coopérant dans une application distribuée ou massivement parallèle [8]. Les aspects de communication et de calcul y sont séparés, de telle sorte que les processus de calcul n'ont aucune information quant à leur communication avec d'autres processus, et des processus coordinateurs gèrent la communication entre des processus, sans connaître le type de calcul effectué. Il s'agit donc d'un parallélisme focalisé non tant sur l'amélioration des performances de calcul que sur la décomposition modulaire et la ré-utilisabilité des processus. En ceci, il s'agit d'un langage ressortant de la problématique des langages de coordination [42]. Les processus coordinateurs démarrent, arrêtent et interconnectent dynamiquement les entrées et sorties des calculs; en cela, ce langage présente des concepts intéressants vis-à-vis de la problématique de langage mixte qui nous intéresse, pour la description de séquençement de modes continus. Par ailleurs, la définition formelle de la sémantique d'un noyau de MANIFOLD correspond au souci de travailler sur des modèles formels, à partir desquels construire des outils.

Parmi les concepts de base de MANIFOLD, on a les *processus*, considérés comme des boîtes noires, dont on connaît les *ports* de connexion en entrée et en sortie, à travers lesquels ils échangent des unités d'information avec leur environnement. Les opérations internes de ces processus peuvent être écrites en MANIFOLD aussi, ce qui permet une forme de spécification hiérarchique. Les processus spécifiés dans d'autres langages sont dits *processus atomiques*. Les interconnexions sont établies entre les ports et sont appelées *flots*. Ils sont installés et détruits dynamiquement, par des processus constructeurs eux-mêmes non-nécessairement partie de la communication. Indépendamment des flots, existe la notion d'*événements*, qui sont émis par les processus atomiques, diffusés de façon asynchrone à toute l'application, et éventuellement captés par des *processus coordinateurs* écrits en MANIFOLD qui les traitent. Un processus coordinateur constitue donc un gestionnaire d'un sous-ensemble du réseau de flots, et, en réaction à l'émission d'un événement, il change d'état: il détruit les connexions qu'il gérait, et installe la configuration décrite au

regard de l'événement reçu, en procédant éventuellement au lancement de nouvelles instances de processus, ou à des désactivations, et aussi à l'émission d'événements, internes ou diffusés dans l'environnement. La définition de MANIFOLD [8] est informelle et assez complexe ; j'ai proposé une sémantique opérationnelle structurale pour une partie de MANIFOLD.

Modèle. Un modèle simplifié : MINIFOLD, fait la synthèse des points caractéristiques [9]. La Figure 1.6 donne l'exemple d'un programme MINIFOLD comprenant deux processus atomiques A1 et A2 avec leurs entrées, sorties, et événements, et deux coordinateurs C1 et C2. Chacun de ces derniers a ses états (désignés dans la Figure 1.6 respectivement par s_i et s'_j), définis par l'événement (noté $\langle \text{événement} \rangle$. $\langle \text{processus-émetteur} \rangle$) à la réaction duquel l'état correspond, et une expression des flots installés (notés $\langle \text{port-source} \rangle \rightarrow \langle \text{port-destination} \rangle$), avec un opérateur + de composition). La Figure 1.7 illustre graphiquement les sous-ensembles de réseaux définis par chacun de ces états.

Une application est alors un ensemble de processus, dont l'état se traduira par l'ensemble des connexions gérées en parallèle par les coordinateurs, qui réagissent aux événements émis par les processus atomiques. La Figure 1.8 illustre les états résultants pour l'exemple. La composition parallèle des processus est ici considérée comme synchrone, au sens où les transitions des différents processus sont synchronisées dans une transition globale.

Un modèle alternatif, décrit directement en termes de systèmes de transitions étiquetées finis [11], a été envisagé (voir Figure 1.9 l'automate pour l'exemple, où s_0 et s'_0 sont les états initiaux, sans flots installés). Le lien entre les deux ressort de la même problématique que la compilation des langages synchrones, par exemple celle d'ESTEREL [22].

Outil. La sémantique opérationnelle structurale (SOS) de MANIFOLD, construite comme celle de MINIFOLD mais plus complexe, a été mise en œuvre sous forme d'interpréteur/simulateur des règles de la sémantique, d'abord en Prolog, puis codée dans le système ASF+SDF à base d'algèbres de processus [59] avec Sylvie Thiébaux [93].

Activités sur les flots de données dynamiques

publications : conférences (dont [9]), rapports de recherche (dont [93]).

outil : prototype d'interpréteur appliquant les règles de SOS, première version en Prolog, deuxième version en ASF+SDF.

collaboration : troisième partie de post-doc ERCIM, CWI, Amsterdam (Pays-Bas), Interactive Systems Dept. (sept. 1991–août 1992).

Interaction/interopération entre séquençement et tâches. Ces travaux, effectués hors du cadre des langages réactifs (même si le caractère événementiel de MANIFOLD l'en rapproche), ont dégagé divers points qui constitueront les caractéristiques des travaux futurs, dans un cadre différent :

- la notion d'état de contrôle défini par le réseau flot de données actif.

D'une part, en rapport avec la notion, présentée dans la problématique (Section 1.2), de tâche réalisant une interaction continue son l'environnement, les réseaux flot de données en sont une représentation naturelle. Une telle tâche, ou un ensemble de ces tâches, définit alors un mode d'interaction avec l'environnement, et les applications des systèmes qu'on considère comprennent des commutations entre différents modes. Il est donc

```

atomic A1 in i out o event e1
atomic A2 in i out o event e1, e2

coordinator C1      in i      out o
( e1.A1: C1.i->A1.i + A1.o->A2.i .                (s1)
  e1.A2: A2.o->A1.i + A1.o ->C1.o . )              (s2)
coordinator C2      in i      out o
( e1.A1: A2.o->C2.o .                                (s'1)
  e1.A2: C2.i ->A2.i + A2.o ->C2.o .                (s'2)
  e2.A2: A2.o ->A2.i . )                             (s'3)
    
```

FIG. 1.6 – *Flots de données dynamiques: code MINIFOLD pour un exemple à deux coordinateurs.*

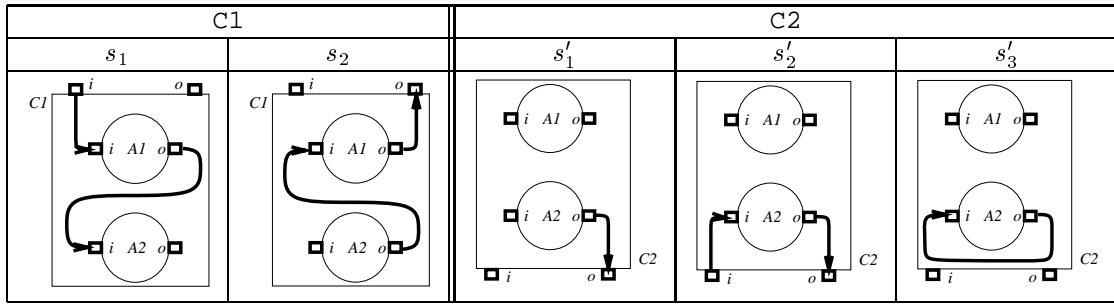


FIG. 1.7 – Flots de données dynamiques : sous-réseaux des coordinateurs C1 et C2.

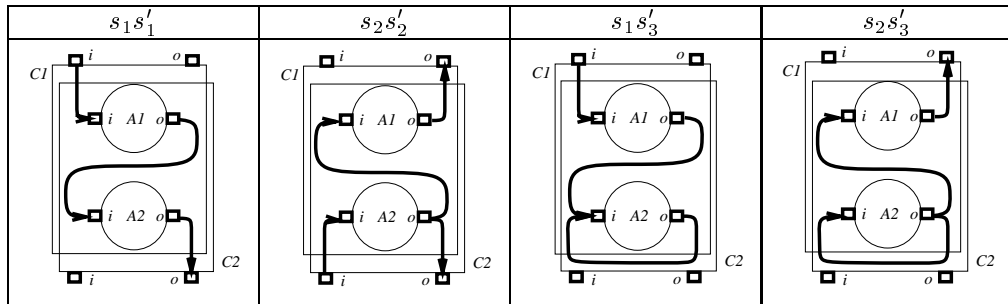


FIG. 1.8 – Flots de données dynamiques : états globaux du réseau de l'application.

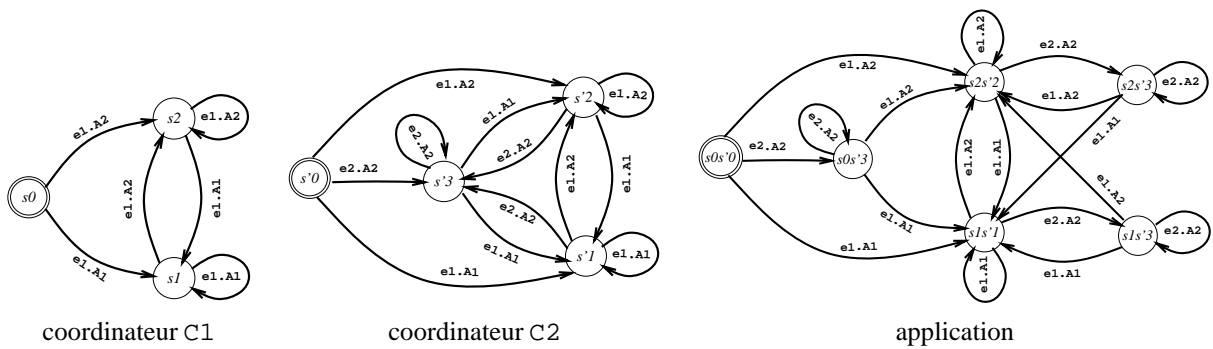


FIG. 1.9 – Flots de données dynamiques : automates pour les coordinateurs C1 et C2, et l'application.

intéressant de noter que, d'autre part, les automates à états finis peuvent se construire à partir de langages plus structurés, dont ESTEREL est un exemple.

Même si la façon dont intervient le séquençement dans MANIFOLD est différente, l'articulation entre tâche et séquençement, et leur interaction, y a pris alors une forme bien définie, plus profonde et plus claire.

- l'intérêt des modèles à base de système de transitions, ou automates à états finis, pour la définition et la compilation comme pour la vérification.

En effet, l'effort de définition formelle des structures de langage proposées contribue à leur clarté et à la limitation de leur complexité. Il permet aussi la construction d'outils au fonctionnement directement liés à cette sémantique, moins dépendants de leur mise en œuvre (compilateur ou simulateur).

1.4 Vers le séquençement de tâches flot de données

De ces travaux, variés par leur contenu (modèles utilisés) comme par les domaines thématiques dans lesquels ils s'inscrivaient (robotique, intelligence artificielle, informatique de coordination), se dégagent des notions qui auront un écho dans la suite, et constitueront même les concepts essentiels de mes propositions ultérieures de langage ou modélisation dans un contexte synchrone :

- le séquençement de tâches, ici vu comme un langage impératif présentant des opérateurs, certains réactifs, spécifiant la disposition temporelle relative de processus ou tâches contrôlées de l'extérieur.
- des actions structurées, où les tâches peuvent elles-mêmes comporter une séquence interne, par exemple celle des tâches-robot d'ORCCAD, liée aux notions de pré- et postconditions. À ces niveaux de hiérarchie de tâches et sous-tâches correspondent clairement les structures de préemption dans les langages réactifs.
- au niveau le plus bas de cette hiérarchie, l'association d'états de connection de réseaux flot de données à des états d'un système de transition, qui offre un mécanisme de séquençement de processus flot de données.

Concernant la modélisation, on voit par ailleurs qu'on peut utiliser les langages (ou formats) synchrones comme constructeurs structurés de tels modèles, en bénéficiant alors des chaînes de compilation et de vérification disponibles.

La suite du document présente, de façon plus détaillée, comment sur ces bases de départ, et les notions essentielles ayant été dégagées, mes travaux ont pris la forme de contributions sur le séquençement de tâches flot de données dans les systèmes réactifs (Chapitre 2) et ses applications à des systèmes de contrôle/commande, notamment robotiques (Chapitre 3).

Chapitre 2

Séquençement de tâches flot de données

Ce chapitre, après de brefs rappels sur la programmation synchrone (Section 2.1) et SIGNAL (Section 2.2.1), fait une présentation de *GTi* (Section 2.2.2). Il s'agit d'une proposition d'extension au langage SIGNAL avec des notions d'intervalle de temps et de tâche, avec une modélisation en SIGNAL par le biais de schémas de traduction, et la mise en œuvre d'un outil sous la forme d'un pré-processeur au compilateur SIGNAL. On présente aussi brièvement des travaux de nature plus théorique sur la préemption dans les langages réactifs, reposant sur l'algèbre de processus PAL.

En Section 2.3, on poursuit par la présentation de la réutilisation de concepts et résultats de *GTi* dans le cadre de la modélisation en SIGNAL de langages plus répandus et plus élaborés, dans le sens où leurs structures de base sont plus complexes, moins primitives, que celles de SIGNAL. Ce caractère plus complexe est dû notamment à la combinaison de plusieurs formalismes, et donne lieu à des problèmes intéressants de modélisation. Il s'agit de :

- STATEMATE, qui regroupe les STATECHARTS et les ACTIVITYCHARTS (en Section 2.3.1),
- la norme IEC 1131 de programmation des automates programmables industriels, et en particulier sa partie concernant les langages, qui eux-même comprennent les SEQUENTIAL FUNCTION CHARTS, variante du GRAFCET (en Section 2.3.2).

2.1 Langages réactifs et synchrones

Plutôt qu'un exposé complet, disponible ailleurs [18, 47, 48], je fais ici un bref rappel de quelques principes de base, et je mentionne les principales directions de travaux dans le domaine.

2.1.1 Principes

Réactivité

Une typologie devenue classique distingue des classes de systèmes selon leur interaction avec leur environnement, au sens de l'influence que peut avoir sur leur déroulement, entre leur démarrage et leur terminaison, la dynamique de cet environnement [52] :

- les systèmes transformationnels se caractérisent par l'absence d'interaction : les données fournies au démarrage définissent entièrement tout le comportement subséquent ; un exemple en est un logiciel de calcul numérique, ou un compilateur classique ;
- les systèmes interactifs procèdent à des interactions avec l'environnement à un rythme dont ils décident : ils n'acceptent d'entrées que quand ils sont prêts à les recevoir ; un exemple en est un système d'exploitation, ou un logiciel de réservation sur des lignes aériennes ;
- les systèmes réactifs interagissent avec l'environnement au rythme de ce dernier : leur comportement est défini entièrement en réaction à celui de l'environnement ; des exemples se trouvent dans les systèmes de contrôle de procédés physiques mentionnés dans les motivations de ce travail en Section 1.1, comme par exemple les contrôleurs embarqués en avionique ou dans les sites industriels automatisés.

On peut considérer que la mise en œuvre d'un contrôleur réactif est sa transformation en programme interactif, pour lequel le rythme de synchronisation avec l'environnement a été adapté à ce dernier ; ce programme interactif pouvant être lui-même une suite d'actions transformationnelles, à chaque réaction. Dans le cas d'une mise en œuvre

distribuée, on peut descendre à un grain d'action transformationnelle plus fin, où à l'intérieur d'une réaction, les processus peuvent communiquer entre eux de façon interactive. Toutes ces notions sont donc plutôt complémentaires qu'exclusives, et contribuent à des degrés divers d'abstraction des systèmes considérés. À la notion de réactivité on peut ajouter celle de «pro-activité», où il s'agit de pouvoir définir un comportement non seulement en termes de réactivité, mais aussi en termes interactifs voire relationnels. Comme on le verra dans la suite, le langage SIGNAL est pro-actif. Cette caractéristique pourrait être utilisée pour représenter les transformations progressives, depuis la spécification synchrone d'origine vers la mise en œuvre désynchronisée dépendante d'une architecture, en termes de transformations de programmes SIGNAL [12, 20] ; le modèle formel de SIGNAL peut alors servir à analyser et valider ces transformations.

Avant de présenter l'interprétation synchrone de la réactivité [47], on peut mentionner qu'il en existe des formes non strictement synchrones telles que STATECHARTS et STATEMATE [51], et ses variantes nombreuses, ou SUGAR-CUBES [28]. La différence réside dans le fait que les effets de la présence et/ou valeur d'une entrée ou d'un événement ne sont pas tous considérés dans le même instant de réaction : les effets indirects sont reportés, par exemple avec un délai à l'instant suivant. Une réactivité asynchrone se trouve dans ELECTRE [92], au sens du style de spécification, où il s'agit du contrôle (présentant éventuellement de la simultanéité) de tâche extérieures non-instantanées, en réaction à un seul événement d'entrée à la fois.

Synchronisme

L'approche synchrone [47, 48] rassemble une famille de langages, et les travaux autour d'eux concernant leur définition, leur compilation, leurs mises en œuvre et leur analyse et vérification. C'est l'opérateur de composition synchrone et son interprétation qui forme leur point commun.

La composition synchrone. La composition synchrone est une opération composant deux processus définis par leur comportement. Elle est similaire à la composition synchronisée de systèmes de transitions étiquetés [11] en ce qu'un événement «émis» (dont la présence est définie) par l'un des processus, peut «être reçu par» (influencer) la (ou les) transition(s) tirable(s) par un autre processus, pour définir la transition du système global résultant de la composition. Alors, dans ce système de transition global, l'émission et la réception de l'événement sont considérés dans la même transition, c'est-à-dire dans la même réaction, c'est-à-dire encore dans le même instant logique vis-à-vis des entrées. Dans le cadre de la compilation des langages synchrones, on est donc amené à calculer tout l'ensemble des conséquences (directes et indirectes) de la présence d'une entrée ou d'un événement, ce qui fait intervenir une notion de point fixe sur les actions effectuées ou événements présents dans une réaction. Ceci a fait dire qu'on pouvait considérer les communications synchrones entre processus comme instantanées, ou de durée nulle : on voit qu'il s'agit d'une image intuitive aidant à la compréhension du modèle, et non d'une représentation irréaliste, ni d'une exigence insatisfiable sur un mécanisme d'échange de données.

Quelques spécificités. Du point de vue de la spécification de contrôleurs, cette opération de composition synchrone permet la décomposition d'un programme sans introduction de complexité ou d'imprédictibilité supplémentaire (par exemple délais de transmission d'information) qui ne seraient dus qu'à cette décomposition, et non au comportement voulu. Par comparaison, dans STATEMATE on peut aussi décomposer hiérarchiquement un comportement en sous-comportements composés. Mais si on définit par exemple un événement en cascade suivant cette décomposition, chaque niveau propageant vers le haut les contributions de niveaux inférieurs, alors chaque niveau introduira un décalage d'un pas ou instant logique [46]. Ceci introduit une complexité peu intuitive, qui compromet la réutilisabilité de sous-programmes, et rend leur modification délicate. La composition synchrone permet de s'affranchir de la profondeur d'une décomposition hiérarchique, ou du nombre de sous-comportements composés en parallèle. Cette signification de la composition synchrone est illustrée par diverses applications [65].

Du point de vue de la structure des systèmes de transitions résultants, on obtient des modèles où les actions de processus parallèles sont considérées comme simultanées. Dans les méthodes asynchrones, il est plus usuel de ne considérer de systèmes de transitions qu'étiquetés par une seule action à la fois : de telles actions donneraient lieu alors à un modèle où seraient représentés tous les entrelacements possibles, avec tous les états intermédiaires associés. Ce qui représente utilement des phases d'une synchronisation par échange de messages par exemple, n'introduit que du coût supplémentaire quand il s'agit de composer des spécifications partielles. On obtient donc dans l'approche synchrone un système de transition plus compact, ce qui favorise la vérification ou la compilation.

2.1.2 Langages synchrones

Langages impératifs

Vis-à-vis de leur style de programmation, les langages synchrones sont habituellement classés [47, 48] en impératifs (textuels ou graphiques) et flot de données. Parmi ces derniers se présente une distinction plus sémantique concernant les classes de comportements (en termes d'entrées-sorties) représentables.

ESTEREL¹ décrit au moyen d'instructions impératives des structures de contrôle séquentielles, parallèles, conditionnelles, itératives, dotées de points d'attente de l'occurrence d'événements ou signaux (événements valués) [22]. Des structures préemptives permettent de gérer, sur l'occurrence d'un signal, l'interruption d'un sous-programme, éventuellement sous forme de mécanisme d'exception [21]. Il est compilé vers un système de transitions représenté symboliquement, et où toutes les communications apparaissant entre les branches parallèles ou dans la hiérarchie de la spécification sont résolues par le calcul de la réaction globale.

ELECTRE² gère des tâches externes (au sens où leur comportement est défini dans un autre formalisme) appelées modules, au moyen de divers opérateurs de préemption et de gestion des événements [87]. Les événements d'entrée ne sont acceptés qu'un à la fois, dans la perspective où l'environnement naturellement asynchrone ne peut les produire de façon strictement simultanée. Ces événements peuvent être traités immédiatement ou mémorisés pour traitement ultérieur. Le traitement peut consister en diverses actions sur les modules, telles que interruption (nécessaire ou facultative), ou suspension (la reprise se faisant au point d'interruption).

ARGOS³ est un langage graphique à base de graphes à états concurrents hiérarchiques [71]; il constitue une variante de STATECHARTS où sont résolus divers problèmes de clarté de la sémantique, et de composabilité, en adoptant l'approche synchrone stricte. Autour de ce langage sont menés des travaux concernant des points de sémantique synchrone, de programmation multi-formalisme en relation avec LUSTRE, et d'extensions temporelles et hybrides [72].

SYNCCHARTS⁴ est lui aussi un langage graphique à base de graphes à états concurrents hiérarchiques [6]. Il a un jeu d'opérateurs plus élaboré, notamment en ce qui concerne la préemption, où il reprend les concepts d'ESTEREL. Il présente d'ailleurs une modélisation en termes d'ESTEREL, qui donne lieu à une mise en œuvre produisant un programme ESTEREL correspondant au programme SYNCCHARTS donné en entrée.

Langages déclaratifs

Les langages déclaratifs sont de type équationnel, avec une syntaxe graphique à base de blocs-diagrammes, et une sémantique en termes de flots de données [19]. Un programme est un système d'équations, ou un graphe d'opérations liées par les données échangées, et la sémantique est qu'un flot d'entrées donne lieu à l'exécution des calculs définis dans les opérateurs, qui produisent leur sorties, elles-mêmes consommées en entrée par d'autres opérateurs, ou produites en sortie du programme.

LUSTRE⁵ est un langage flot de données fonctionnel : un programme définit une fonction des flots d'entrées (suites de valeurs typées) vers ceux de sortie [49]. Les opérateurs sont structurés en nœuds, composés d'instructions temporelles (accès à la valeur précédente, ou délai ; préfixe «suivi de» flot ; sous-échantillonnage ; maintien de valeur courante).

SIGNAL⁶ est relationnel, dans le sens où il permet de décrire des relations de synchronisation entre des flots, sans préjuger de leur caractère d'entrée ou de sortie, ce qui banalise les interfaces et permet de s'exprimer librement à leur sujet [63]. Cette relation peut donner lieu à la définition d'un ensemble de sorties possible pour un flot d'entrée, ou aussi à un ensemble de flots internes possibles pour une entrée et une sortie. Par là, il modélise aussi le non-déterminisme et les comportements «pro-actifs» (où les comportements internes se déroulent sans nécessairement

1. Site web pour ESTEREL : <http://www-sop.inria.fr/meije/esterel/esterel-eng.html>

2. Site web pour ELECTRE : http://www.ircyn.prd.fr/Productique/Temps_Reel/Temps_ReelF.html

3. Site web pour ARGOS : <http://www-verimag.imag.fr/SYNCHRONE/argonaute-english.html>

4. Site web pour SYNCCHARTS : <http://www-mips.unice.fr/~andre/synccharts.html>

5. Site web pour LUSTRE : <http://www-verimag.imag.fr/SYNCHRONE/lustre-english.html>

6. Site web pour SIGNAL : <http://www.irisa.fr/ep-atr/>

la stimulation directe par une entrée venant de l'environnement). Dans ce sens, il s'agit d'un langage de programmation par contraintes, où on pose un ensemble d'équations définissant un ensemble de solutions (comportements respectant la spécification). On peut le réduire progressivement jusqu'à obtenir éventuellement une solution fonctionnelle, pour laquelle on sait alors produire une version exécutable de façon déterministe. La Section 2.2.1 précisera les caractéristiques de SIGNAL.

Formats synchrones

Les langages synchrones sont tous dotés d'un compilateur mettant en œuvre des algorithmes d'analyse et de transformations spécifiques. Cependant leurs similitudes dans le domaine sémantique décrit, et l'applicabilité de fonctionnalités associées aux uns sur des spécifications écrites dans les autres, ont motivé la définition de formats d'échanges entre les outils. Ce sont des langages à la syntaxe moins élaborée que dans le cas d'un langage de programmation destiné à des utilisateurs. Ils correspondent à des degrés divers de généralité, DC+ [102] étant le plus proche des spécifications et le plus expressif (notamment en ce qui concerne le non-déterminisme et la multiplicité de cadences), et DC (*Declarative Code*) en étant une version simplifiée, sous-ensemble syntaxique et sémantique.

2.1.3 Approches du multi-formalisme dans les langages réactifs

Plus particulièrement proches de la problématique exposée en Section 1.2, on trouve des approches liées aux modèles et techniques synchrones du multi-formalisme, combinant formes impératives et équationnelles.

ARGOS et LUSTRE. Dans l'intégration de LUSTRE et d'ARGOS [56], les deux langages sont plongés dans un cadre sémantique commun, et où un état d'ARGOS peut être raffiné en un nœud LUSTRE. Un aspect de cette composition est que les nœuds LUSTRE, quand on ré-entre dans l'état ARGOS qu'ils raffinent, doivent être réinitialisés, ce qui requiert une gestion explicite dans la sémantique. Une évolution de ce travail se trouve dans la compilation d'ARGOS en équations Booléennes [73] où les états se trouvent encodés dans des mémorisations, et l'entrée dans un état, avec ré-initialisation des sous-automates, donne lieu à un encodage par construction d'un événement de remise à l'état initial propagé le long de la structure hiérarchique et parallèle du modèle.

Automates de modes.⁷ Les automates de modes [74] sont une extension de LUSTRE par des constructeurs à base d'automates, permettant de décrire de façon simple des modes de fonctionnement d'un système, et la commutation entre ces modes. Chaque mode est défini par le système d'équations associé qui calcule les flots. On peut composer parallèlement ou hiérarchiquement ces automates. Il s'agit d'une évolution des travaux précédents, où sont étudiés notamment les questions de gestion de compositionnalité, définition multiple de flot et règles de portée ou de priorité.

L'environnement ORCCAD.⁸ C'est un environnement de conception de contrôleurs pour la robotique [24]. Il concerne différents niveaux de programmation, qui concernent souvent des utilisateurs aux connaissances différentes. La conception des lois de commande de systèmes robotiques, et leur mise en œuvre sont encapsulés dans des modules, définis avec leurs entrées, sorties, initialisation et fonction de calcul. On peut composer ces modules dans un graphe à flot de données, avec différents types de synchronisations. Un tel graphe flot de donnée est lui-même encapsulé dans une tâche, qui lui adjoint des conditions de démarrage, d'arrêt, des exceptions. Enfin, le séquençement de ces tâches est l'objet d'un autre niveau, où il s'agit de commuter entre différentes lois de commandes, selon des événements relatifs aux capteurs ou aux exceptions pour lesquelles il faut définir un traitement. La définition en structures ESTEREL des tâches et de leur enchaînement offre la possibilité de bénéficier des outils de compilation et de vérification associés. On a donc une forme de combinaison de formalismes impératif et à flot de données, ce dernier restant externe au modèle synchrone.

Il existe d'autres instances de combinaison de formalismes de séquençement et de flot de données, notamment dans le cadre des langages de large diffusion détaillés plus loin (voir Section 2.3) :

- GRAFCET (tout au moins sa variante SEQUENTIAL FUNCTION CHARTS), lié, dans le cadre de la norme IEC 1131 [29], à des formalismes de bloc-diagrammes (voir Section 2.3.2). Des travaux concernent son intégration avec SIGNAL [67] ou ESTEREL [40].

7. Site web pour les automates de modes :

<http://www-verimag.imag.fr/PEOPLE/Florence.Maraninchi/MATOU/index.phtml>

8. Site web pour ORCCAD : <http://www.inrialpes.fr/iramr/pub/Orccad/>

- STATECHARTS et les autres langages de STATEMATE [51], notamment les ACTIVITYCHARTS, qui ont des aspects bloc-diagrammatiques (voir Section 2.3.1). Eux aussi ont été étudiés au regard de SIGNAL [45] ou ESTEREL [104].

Activités générales sur les langages réactifs et synchrones

enseignement : Intervention dans l'option PATR (*Programmation d'Applications Temps-Réel*, resp. Paul Le Guernic) du DEA d'Informatique de l'IFSIC, Université de Rennes 1 : généralités sur les systèmes réactifs, systèmes de transitions finis (produit synchronisé, équivalences), langage ESTEREL, son extension CRP et sa sémantique, et STATEMATE (env. 15h par an, de 95 à 99).

collaborations : Participation aux réunions du Groupement C2A (Collaboration CAO Automatique) qui fédérait la recherche effectuée en France autour de la programmation synchrone, avec des partenaires industriels, tels que Schneider-Électrique, Dassault-Aviation, Thomson, EDF, Snecma. J'y ai participé à partir de Janvier 1991. J'y ai co-organisé un séminaire sur la programmation synchrone en robotique, les 17 et 18 Octobre 1994.

2.2 SIGNAL, GTi et la préemption

Dans le contexte présenté précédemment, je propose de définir, dans le cadre de SIGNAL (brièvement présenté en Section 2.2.1), une extension appelée GTi⁹ comportant des structures de tâches et de préemption (Section 2.2.2) ; la préemption dans les langages réactifs est étudiée aussi au travers d'un formalisme d'algèbre de processus (Section 2.2.3).

2.2.1 SIGNAL

SIGNAL [63] est un langage et un environnement de spécification et de programmation de systèmes réactifs ou temps-réel, faisant partie de la famille des langages synchrones [47]. Il est développé dans le projet Ep-Atr à l'IRISA/Inria-Rennes, depuis une quinzaine d'années, notamment de façon permanente par Paul Le Guernic (chef de projet), Albert Benveniste, Loïc Besnard, Patricia Bournai, Thierry Gautier. Le langage SIGNAL est un langage déclaratif décrivant des flots de données synchronisés. Il est construit autour d'un noyau minimal d'opérateurs primitifs. Ceux-ci manipulent des signaux, qui sont des suites non bornées de valeurs typées, associés à une horloge. Celle-ci détermine les instants auxquels les valeurs sont présentes; par exemple, un signal X dénote la séquence $(x_t)_{t \in T}$ de données indexées par le temps t dans un domaine T , et son horloge est notée \hat{x} . Des signaux d'un type particulier appelés *event* sont caractérisés seulement par leur horloge, c'est-à-dire leur présence (ils ont la valeur booléenne *true* à chaque occurrence). Étant donné un signal X , son horloge est donnée par l'expression \hat{X} , qui donne l'événement présent simultanément à X . Les constructeurs du langage permettent de spécifier dans un style équationnel des relations entre les signaux, c'est-à-dire entre leurs valeurs et entre leurs horloges. Des systèmes d'équations sont construits en utilisant la composition.

Le compilateur se livre à une analyse de la consistance du système d'équations, et détermine si les contraintes de synchronisation ont une solution. Si c'est le cas, et si le programme est contraint de façon à calculer une solution unique, alors un code exécutable peut être produit (en C ou en Fortran). Si ce n'est pas le cas, la compilation constitue une analyse de consistance de la spécification partielle, et ses résultats peuvent servir dans une perspective de vérification des comportements dynamiques, ou de compilation modulaire [20].

Le langage SIGNAL

Le noyau de SIGNAL. En SIGNAL, les opérateurs de base définissent des processus élémentaires, chacun correspondant à une équation :

Les opérateurs fonctionnels. Ils sont définis sur les types du langage (par exemple la négation booléenne du signal $E : \text{not } E$). Le signal (Y_t) , défini par la fonction f dans : $Y_t = f(X_{1t}, X_{2t}, \dots, X_{nt})$ est écrit sous la forme : $Y := f(X_1, X_2, \dots, X_n)$. Les expressions fonctionnelles sont monochrones, ce qui signifie que les signaux Y, X_1, \dots, X_n sont dits synchrones : ils partagent la même horloge. En d'autres termes, pour calculer la valeur de Y_t , tous les X_i doivent être disponibles à l'instant t ; pour cette raison ils sont contraints à avoir la même horloge, qui est aussi celle de Y .

9. GTi pour *Gestion de Tâches et d'intervalles*

Le retard. Il donne la valeur passée d'un signal, ce qui est généralement noté $ZX_t = X_{t-d}$, avec la valeur initiale $ZX_i = V_i$, pour $0 \leq i < d$; en SIGNAL, pour le cas simple où $d = 1$, avec initialisation à V_0 , on écrit: $ZX := X\$1 \text{ init } V_0$. Le délai est monochrome lui aussi, c'est-à-dire que X et ZX ont la même horloge.

Le filtre. Le filtrage d'un signal X selon une condition C est écrit: $Y := X \text{ when } C$. On l'appelle aussi sous-échantillonnage ou sélection. Cet opérateur est polychrone: les opérandes et le résultat n'ont pas la même horloge. Le signal Y est présent si et seulement si X et C sont présents au même instant et que C a la valeur `true`. Quand Y est présent, sa valeur est celle de X . Ainsi, Y est moins fréquent que X et que C à la fois: l'intersection des horloges de X et de C (c'est-à-dire les instants où l'expression peut être évaluée) inclut l'horloge de Y (qui ne comporte que les instants où C s'évalue à `true`). Quand on veut désigner l'intersection des horloges des signaux X et Y , on note en SIGNAL: $X \wedge Y$.

La fusion. On définit la fusion Z de deux signaux X et Y par: $Z := X \text{ default } Y$. La valeur de Z est celle de X quand il est présent, sinon celle de Y quand il est présent. Cet opérateur aussi est polychrone: l'horloge de Z est l'union de celles de X et Y , elle est donc plus fréquente que chacune d'elles. Quand on veut désigner l'union des horloges des signaux X et Y , on note en SIGNAL: $X \vee Y$.

La composition. Les processus élémentaires peuvent être composés par l'opérateur commutatif et associatif « $|$ » qui dénote l'union des systèmes d'équations. En SIGNAL, pour des processus P_1 et P_2 on écrit: $(| P_1 | P_2 |)$.

Par exemple, l'équation $x_t = f(x_{t-1}) + 1$ peut aussi s'écrire:

$$\begin{cases} x_t = f(zx_t) + 1 \\ zx_t = x_{t-1} \end{cases}$$

ce qui s'écrit en SIGNAL:

$$(| X := f(ZX) + 1 | ZX := X\$1 |)$$

Le langage est construit autour de ce noyau et comporte des opérateurs dérivés pour les tableaux ou les variables par exemple. On peut définir des schémas de processus, qui ont un nom, des paramètres et des signaux d'entrée et de sortie typés, un corps et des déclarations locales. Les instances de schémas de processus rencontrées dans un programme sont expansées par un préprocesseur du compilateur.

Une particularité: le sur-échantillonnage. Pour illustrer cette présentation de SIGNAL d'un exemple, je choisis celui du décompteur, qui présente une spécificité de SIGNAL: le sur-échantillonnage, ou «accélération interne». L'exemple classique d'un processus SIGNAL dont l'activité interne (et en l'occurrence les sorties) est plus fréquente (qualitativement parlant) que les entrées est celui du décompteur donné en Figure 2.1. Ce processus est déclaré par la structure de modèle de processus désignée par `process`, avec des signaux en entrée (X entier) et en sortie (Y entier).

```
process decompteur = (? integer X;
                      ! integer Y )
(| Y := X default ZY - 1
 | ZY := Y$1 init 0
 | X ^= when (ZY=0)
 | );
end
```

FIG. 2.1 – SIGNAL: exemple de sur-échantillonnage: le processus décrémenteur d'entrées.

Ce programme accepte l'entrée X quand elle est présente, ou bien décrémente dans Y la valeur de ZY . Cette dernière est la valeur précédente de Y conservée au moyen d'un délai (seule forme de mémorisation en SIGNAL). On passe par toutes les valeurs intermédiaires jusqu'à 0 sur le signal Y , comme l'illustre la trace en Figure 2.2. Aux instants où la valeur précédente de Y (c'est-à-dire ici ZY) est nulle, on prend ou accepte une nouvelle entrée X : ceci est spécifié par la contrainte de synchronisme ($\wedge=$). C'est donc l'état interne du processus qui décide de la synchronisation avec les entrées.

X	3				5					7	...	
Y	3	2	1	0	5	4	3	2	1	0	7	...
ZY	0	3	2	1	0	5	4	3	2	1	0	...

FIG. 2.2 – SIGNAL: trace du processus décrémenteur d'entrées.

```

process cascade = (? integer X;
                    ! integer Y )
( | Yint := decompteur(X)
  | Y := decompteur(Yint)
  | );
where integer Yint
end
    
```

FIG. 2.3 – SIGNAL : exemple de sur-échantillonnage : décompteurs en cascade.

On est donc bien ici dans le cas d'un comportement pro-actif, dans le sens où des actions internes, et des sorties, peuvent avoir lieu sans présence d'une entrée à laquelle réagir. Pour ce type de spécification, il n'est pas toujours possible de proposer (ou de générer automatiquement) une mise en œuvre : dans le cas général, plusieurs sur-échantillonnages non reliés entre eux ne déterminent pas de façon univoque à quels instants les actions doivent être effectuées ; toutefois, dans d'un sur-échantillonnage unique, ou de sur-échantillonnages en cascade, comme illustré en Figure 2.3 l'environnement de programmation SIGNAL et son compilateur peuvent générer une mise en œuvre.

Le modèle et ses utilisations

Les modèles de programme SIGNAL construits suivant leur sémantique sont à la base d'un large spectre de travaux concernant tant l'analyse formelle et la vérification, que les mises en œuvre logicielles comme matérielles. Je donne ici juste un bref aperçu.

Le modèle. Le langage SIGNAL est doté d'un modèle en termes de processus, qui sont des systèmes d'équations sur des signaux. Ceux-ci sont des suites de valeurs à laquelle est associée une horloge, qui définit un ensemble discret des instants auxquels ces valeurs sont présentes. Les équations posent des contraintes sur les horloges, dans le sens où elles décrivent la relation que doivent respecter ces horloges quant à leur présence relatives. Cette relation sera analysée par la compilation, dans le but d'éventuellement lui trouver une forme fonctionnelle si cela est possible. Les propriétés de la composition de processus permettent de transformer des programmes SIGNAL de façon à en séparer des parties destinées à des traitements différents : par exemple on peut séparer la partie Booléenne de celle comportant les calculs non-Booléens (numériques). Cette partie Booléenne représente le contrôle de l'application, au sens où il s'agit des conditions vis-à-vis desquelles se font ou non les calculs. Elle représente les valeurs à vrai, à faux et les absences de signaux : elle est pour cela fondée sur un calcul tri-valué, dans $\{-1, 0, +1\}$, c'est-à-dire $\mathbb{Z}/3\mathbb{Z}$. C'est elle qui définit un système dynamique qui peut être étudié sous plusieurs aspects à des fins de vérification ou de synthèse : étude de l'ensemble des états admissibles (partie statique), et calcul dynamique s'appuyant sur la représentation équationnelle d'un automate.

La compilation et les schémas d'exécution de SIGNAL. La partie statique est utilisée à la compilation, sur une structure de graphe de dépendances conditionnées de flot de données. Ceci inclut des fonctionnalités diverses, telles que :

- vérification de la consistance du système d'équations induit par le programme, c'est-à-dire existence d'une solution, absence de cycle de causalité,
- détermination d'une hiérarchisation des horloges (qui se ressent dans le caractère imbriqué des conditionnelle du code généré),
- optimisation (par détection d'horloges nulles, et du caractère de «code mort» des calculs associés à ces horloges) [3],
- évaluation de performances (en tenant compte des conditions éventuellement exclusives) [61],
- détermination de profil d'entrée/sortie de sous-graphe, au sens de graphe des dépendances conditionnées entre entrées et sorties, en vue de la compilation modulaire, séparée, et de la distribution des programmes.

Dans le cadre de nos activités dans le projet SACRES, nous avons travaillé, avec Sylvain Machard (en thèse), Albert Benveniste, Loïc Besnard, Thierry Gautier et Paul Le Guernic, à une exploration des mises en œuvre distribuées de SIGNAL, plus précisément de modèles dans le format DC+ [20]. Ce dernier point rend ces techniques applicables notamment pour des programmes STATECHARTS traduits en DC+ (voir Section 2.3.1). Il s'agit de définir des méthodologies de manipulation et transformation de graphes DC+ de façon à obtenir des partitions en sous-graphes, avec analyse des communications résultantes, pour être capable de générer un code qui mette en œuvre la spécification

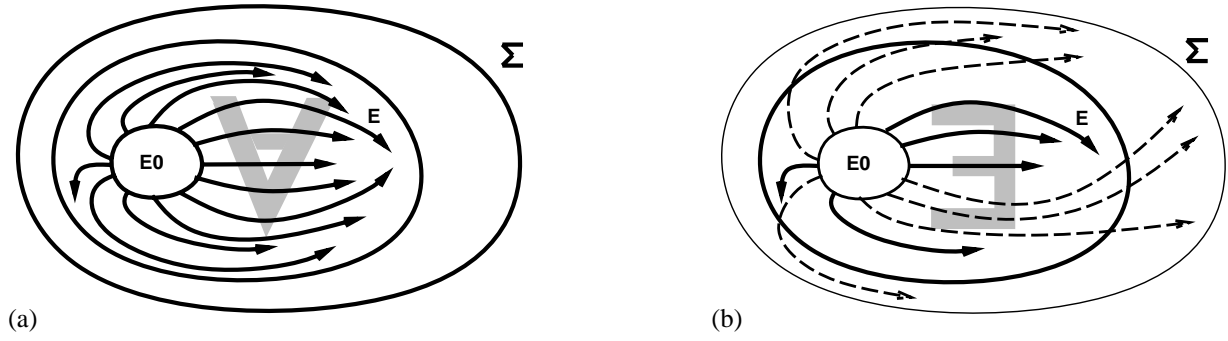


FIG. 2.4 – SIGNAL et SIGALI : invariance (a) et invariance sous contrôle (b) de E depuis E_0

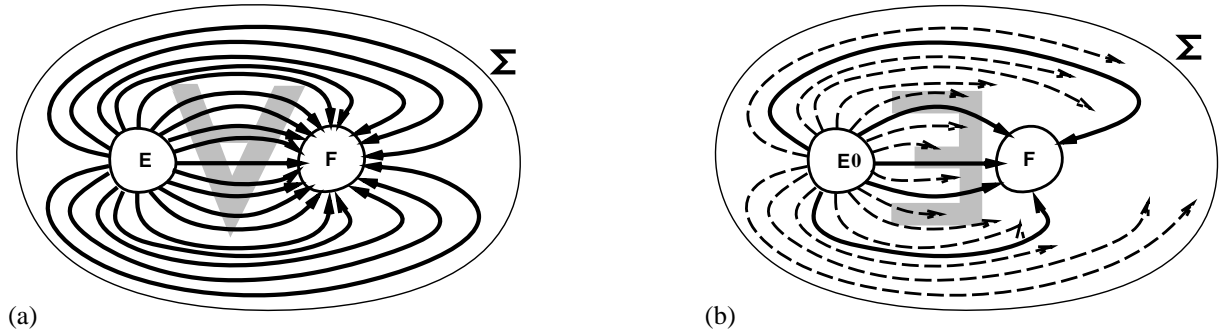


FIG. 2.5 – SIGNAL et SIGALI : Attractivité (a) et accessibilité (b) de F depuis E

originale sur une plate forme asynchrone [64]. Une collaboration avec la DER (direction des études et recherches) d'EDF concerne l'étude de faisabilité d'une intégration des fonctionnalités de distribution de code et des langages de programmation d'automatismes (voir Section 2.3.2).

La vérification et la synthèse de contrôleur. La partie dynamique donne lieu à une forme de traitement différente, eu égard à l'information différente qu'elle donne sur le programme : il s'y agit, plus que des états admissibles, des états accessibles ou atteignables. La vérification des propriétés des programmes se fait par un calcul dynamique s'appuyant sur la représentation équationnelle d'un automate : les automates, leurs états, événements et trajectoires sont manipulés au travers des équations qui les représentent. Ces équations se dérivent naturellement et automatiquement des processus primitifs du langage, sous forme de polynômes sur $\{-1, 0, 1\}$. Calculer des trajectoires d'états ou d'événements, des états atteignables, des projections de trajectoires, des états de blocage, s'effectue alors sur les coefficients des équations polynomiales. Les Figures 2.4 et 2.5 illustrent les propriétés de base calculées, les notions d'attractivité et d'accessibilité étant définissables en termes de plus grand invariant ou invariant sous contrôle. Ces principes se concrétisent dans des techniques, variantes de celles utilisant les BDD¹⁰, mises en œuvre dans l'outil SIGALI, qui a été utilisé dans le cadre d'applications [62]. De manière similaire, des techniques de synthèse de contrôle ont été développées, où un système dynamique est transformé de manière à ce que le système résultant satisfasse des propriétés données comme objectifs ; ces objectifs concernent la sécurité, mais aussi la qualité de service liée au comportement résultant [37, 79, 25].

Des approches alternatives explorées autour du modèle de SIGNAL concernent les systèmes hybrides et la démonstration de théorème, qui élargissent le champ des propriétés vérifiables.

Les outils

L'environnement de programmation SIGNAL. L'environnement de programmation SIGNAL regroupe les résultats des études variées qui ont lieu autour du langage, du modèle et de leurs utilisations, comme illustré en Figure 2.6. Il est doté d'une interface graphique à base de blocs-diagrammes [26]. Sa compilation consiste d'abord

10. Binary Decision Diagrams : diagrammes de décision binaires.

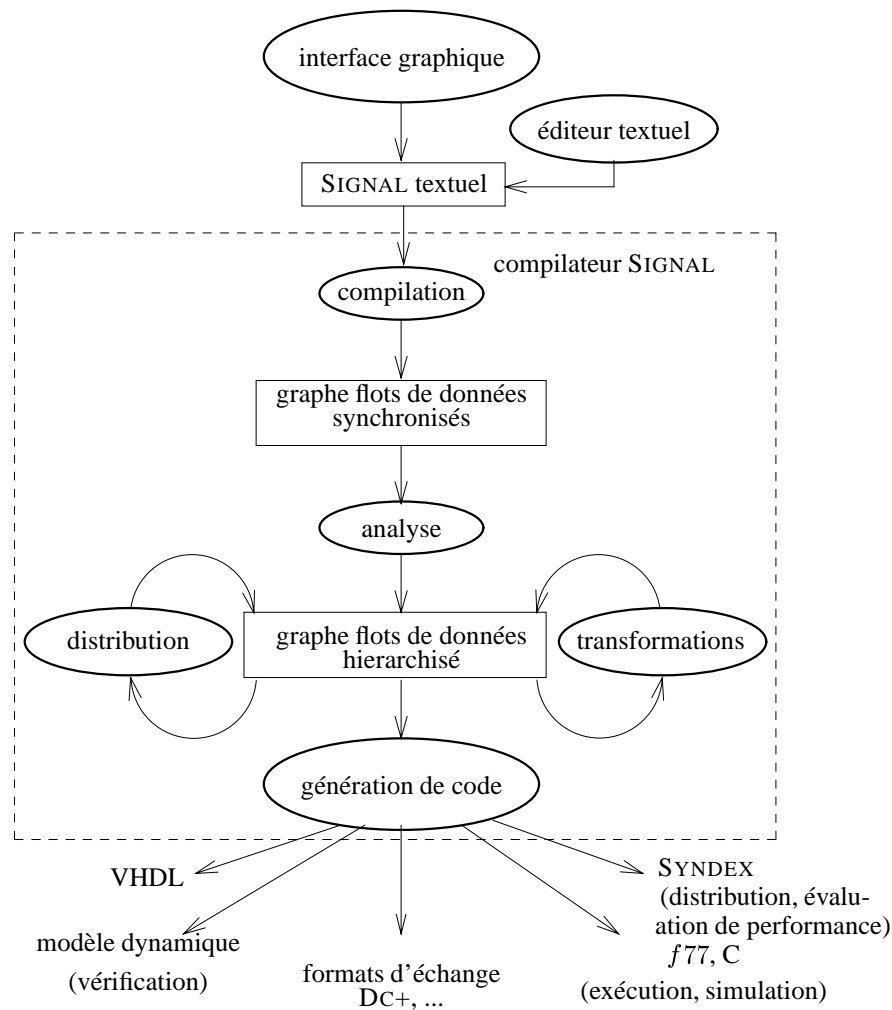


FIG. 2.6 – SIGNAL : architecture de l'environnement de programmation.

en la construction d'un graphe représentant les flots de données entre opérations primitives, avec des informations quant aux horloges des dépendances de données entre calculs, et aux instants auxquels ceux-ci sont définis. L'analyse de ce graphe, vis-à-vis de ces horloges, donne lieu à des transformations qui mènent à un graphe hiérarchisé aux dépendances conditionnées [3]. Ces transformations font intervenir différentes étapes de résolution des contraintes sur les horloges, hiérarchisation, Booléanisation relativement à des horloges progressivement plus globales [41]. À ces phases correspondent des définitions de formats, qui peuvent servir aux échanges avec d'autres outils ou à l'interopérabilité avec d'autres langages [23]. Sur ce graphe peuvent alors prendre place des manipulations dirigées vers des mises en œuvre, qu'on peut faire séquentielles (en C ou Fortran 77), ou dépendantes de l'architecture [12, 20, 64]. Des mises en œuvre liées à la conception conjointe logiciel/matériel (*co-design*) ont été étudiées aussi, donnant lieu à une génération de code VHDL [17], et à des fonctionnalités d'évaluation de performances [61, 60]. Une passerelle existe entre l'environnement SIGNAL et SYNDEX¹¹ avec lequel on peut réaliser des implémentations multi-processeurs. Concernant la vérification des propriétés sur la dynamique des comportements, les travaux fondés sur les systèmes dynamiques polynomiaux ont donné lieu à des résultats théoriques et un support concret sous la forme de l'outil SIGALI [36, 37] ; un autre débouché de ces travaux concerne la synthèse de contrôleurs à événements discrets [79, 25].

Une version industrielle de SIGNAL est développée et distribuée sous le nom de SILDEX par la société TNI¹².

11. Site web pour SYNDEX : <http://www-rocq.inria.fr/syndex/welcome.html>

12. Site web pour SILDEX : <http://www.tni.fr/>

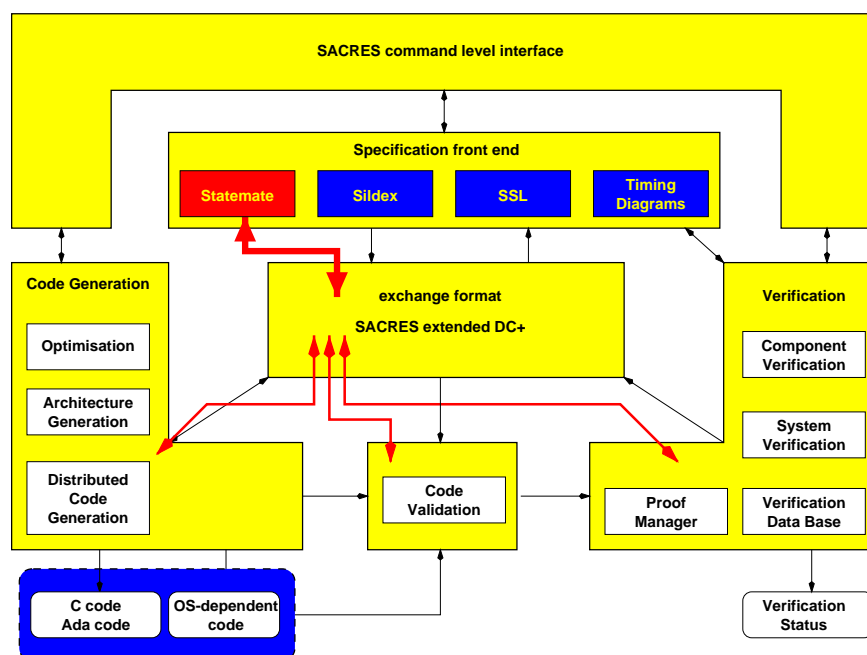


FIG. 2.7 – SIGNAL : l'architecture du projet SACRES, et la position de STM2DC+.

L'environnement SACRES L'environnement SACRES¹³ est le produit d'un projet ESPRIT de même nom, et est organisé autour du format synchrone DC+, constituant en cela une autre instance d'environnement de programmation similaire à SIGNAL. Ce projet regroupait les partenaires : Siemens (RFA), British Aerospace (Grande-Bretagne), i-Logix (Grande-Bretagne), Inria (France), Offis (RFA), Snecma (France), TNI (France) et le Weizmann Institute (Israël) (ainsi que Siemens/Nixdorf (SNI) (RFA), jusqu'en mars 1997).

Le but du projet est de fournir aux concepteurs de systèmes embarqués une meilleure méthodologie de conception permettant de réduire significativement tant le risque d'erreurs que le temps de conception [44]. Pour cela, la validation des spécifications initiales doit se faire à l'aide d'outils de vérification formelle intégrés et les phases de génération de code, réparti notamment, doivent être automatisées. L'approche proposée est multi-formalisme. Elle s'appuie sur un certain nombre d'outils existants : STATEMATE/STATECHARTS, Sildex/SIGNAL, Sildex/Grafset, Timing Diagrams, *model-checker* SVE. Ils sont regroupés dans une architecture logicielle construite autour du format d'échange DC+ et de sa machine virtuelle, et sont transformés et étendus pour répondre aux besoins des utilisateurs. Ceci comporte d'établir les passerelles nécessaires, comme celle de STATEMATE à DC+. Aux outils s'ajoute un travail sur leur méthodologie d'utilisation et sa place dans le processus de conception des utilisateurs. Les résultats du projet sont commercialisés par les partenaires industriels vendeurs (i-Logix et TNI) et utilisés par les partenaires industriels utilisateurs (British Aerospace, Siemens et Snecma)[13].

L'architecture de l'environnement SACRES est illustrée en Figure 2.7, où sont mis évidence les aspects liés à l'intégration de STATEMATE, dont il est question en Section 2.3.1. On y voit qu'au travers d'une interface de commande, on a accès aux langages de spécifications mentionnés plus haut, ainsi qu'à SSL (*System Specification Language*), qui permet d'assembler des composants dans ces langages. Une spécification peut être soumise à compilation en DC+, puis, du côté génération de code, on a des fonctionnalités d'optimisation prenant en compte le comportement des programmes, de modélisation de l'architecture, et de génération de code C ou Ada, séquentiel ou distribué, avec éventuellement des éléments dépendants du système d'exploitation [41]. Du côté vérification, la fonctionnalité de vérification de composant s'appuie sur le *model-checker* SVE de Siemens, et l'état du processus de vérification dans l'ensemble des composants est géré, à l'aide d'une base de données, de manière à ce que des versions successives de la spécification donnent lieu aux re-vérifications utiles. La vérification au niveau système concerne l'utilisation de techniques de vérification compositionnelles.

Activités générales autour de SIGNAL

publications : conférences (dont [13, 20, 64]), session de démonstration[38], rapports de contrat .

13. Site web pour SACRES : <http://www.tni.fr/sacres/>

Conférence invitée : 22st IFAC/IFIP Workshop on Real Time Programming, WRTP'97, Lyon, France, September 15 – 17, 1997.

Participation aux délégations de l'INRIA aux salons internationaux ENTERPRISE (en Juin 1993, Boston, USA) et CEBIT (en Mars 1994, Hanovre, Allemagne).

encadrement :

Co-encadrement, avec Paul Le Guernic, de la thèse de Doctorat en Informatique (IFSIC, Université de Rennes 1) de Sylvain Machard sur le sujet *Mises en œuvre distribuées de SIGNAL* (de fin 1996 à mi 1998).

collaborations :

- Projet Esprit R&D SACRES (EP 20897, 11/95 – 12/98).
En dehors de l'intégration STATEMATE-SIGNAL (voir Section 2.3.1) et des interventions sur la méthodologie et de la distribution, j'ai contribué à la gestion générale de la participation du projet EP-ATR, avec Yan-Mei Tang (ingénieur-expert). Par ailleurs nous avons participé à certains aspects de la gestion générale de SACRES (par exemple le site ftp, localisé à l'Irisa). Concernant le transfert de technologie et la diffusion et l'exploitation des résultats, nous avons participé à l'organisation et à la réalisation de rencontres avec un groupe d'industriels sensibilisés à l'approche SACRES, en vue d'établir des coopérations techniques autour de l'utilisation de l'environnement, entier ou par parties.
- J'ai contribué à des coopérations avec l'équipe de Mazen Samaan, du département CCC (Contrôle Commande Centrales) de la DER (direction des études et recherches) d'Électricité de France (EDF) sur le thème : *Méthodes de distribution de code SIGNAL et DC+*. Cette étude se déroule dans le contexte des études menées à EDF concernant l'application des méthodes formelles émergentes pour les systèmes de contrôle-commande des centrales. L'étude comporte une étude synthétique des méthodes de distribution de code synchrone, notamment celles qui ont été développées en lien avec le projet Esprit SACRES. Elle comporte aussi une étude sur leur application aux systèmes de contrôle-commande des centrales. Une autre collaboration à laquelle j'ai participé est détaillée en Section 3.2.2. L'ensemble de la coopération entre le projet EP-ATR et EDF a donné lieu à une présentation au séminaire général la DER d'EDF en mai 1997.

2.2.2 SIGNALGTi

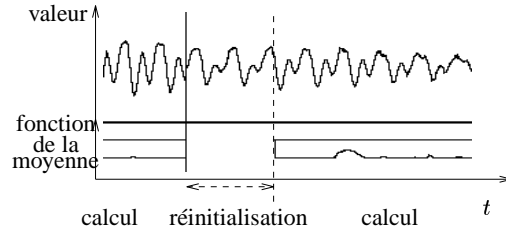
GTi est une extension au langage SIGNAL qui propose un notion d'intervalle de temps, et gère une structure de tâche associant un processus à un intervalle sur lequel il est actif [96, 98]. Ce travail fait usage du socle que constituent les travaux et l'environnement SIGNAL décrits précédemment, et s'inscrit comme couche par-dessus le noyau qu'ils forment. On définit ces constructeurs en leur donnant un modèle en terme de schémas de traduction en SIGNAL, l'ensemble formant SIGNALGTi. L'outil qui concrétise ceci est un préprocesseur de l'environnement SIGNAL [99, 82], qui a été appliqué dans divers contextes, décrits plus avant au Chapitre 3.

Motivations

La motivation de l'introduction de ces structures est de fournir le moyen adapté à la description de systèmes *commutant entre différents modes* d'interaction continue avec l'environnement. Ces modes sont indentifiés par des intervalles de temps délimités par des événements discrets de début et de fin, et durant lesquels les tâches sont actives. Le domaine d'application visé est le contrôle/commande de systèmes physiques, présentant à la fois des calculs sur des flots de données capteur, et des transitions discrètes dans un automate de contrôle.

Par exemple, dans un système de reconnaissance de la parole [63], le traitement du signal acoustique comporte un traitement de segmentation : les extrémités du segment sont déterminées par des changements dans les valeurs du signal. Ceux-ci sont détectés par comparaison avec une valeur moyenne, calculée sur une fenêtre de temps sur les valeurs passées. Une telle application présente des modes ou phases successifs : une phase d'initialisation doit calculer des valeurs de la moyenne, et ensuite le calcul nominal peut être effectué. On distingue donc deux phases (réinitialisation et calcul nominal) qui alternent sur des périodes de temps complémentaires comme l'illustre la Figure 2.8.

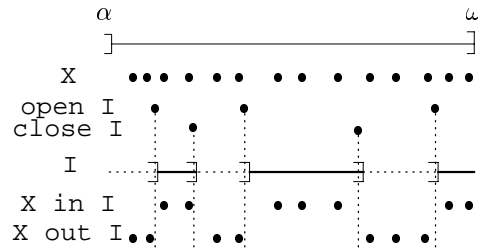
Chacune de ces phases correspond à des équations qui sont le domaine d'application privilégié de SIGNAL. Par contre, si la spécification des modes, et de l'association d'équations différentes à chacun d'eux, est possible en SIGNAL, elle oblige à la programmation explicite d'une structure de contrôle répartie dans les équations, difficile à

FIG. 2.8 – *GTi : phases dans le traitement de signal vocal.*

établir et à modifier, et dont la compilation ne peut guère tirer parti. Notre but est alors de proposer des constructeurs du langage qui soient particulièrement adaptés à la désignation de phases dans un processus. L'approche est de subdiviser son intervalle d'activité en sous-intervalles pour les différents modes, et d'associer des sous-activités à ceux-ci.

Langage

Principes. En *SIGNALGTi*, flot de données et séquencement sont tous deux réunis dans le même cadre de langage, et reposent sur les mêmes modèles pour leur exécution et analyse (pour la compilation et la vérification des programmes). De ce point de vue, une application flot de données est considérée comme s'exécutant à partir d'un état initial de sa mémoire à un instant initial α , qui se situe avant l'instant de la première réaction du programme. Une application flot de données n'a par ailleurs pas de terminaison spécifiée en elle-même : son instant de terminaison ω est imposé de l'extérieur, en lien avec un événement ou le dépassement d'une valeur. Il s'agit de l'instant de sa dernière réaction, donc ω fait partie de l'activité du programme, et l'intervalle d'activité est donc un intervalle de temps ouvert à gauche et fermé à droite : $] \alpha, \omega]$.

FIG. 2.9 – *GTi : subdivision de $] \alpha, \omega]$ en sous-intervalles.*

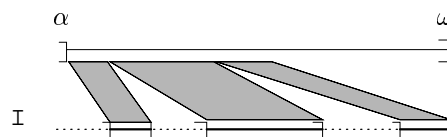
Les intervalles de temps sont introduits pour décomposer $] \alpha, \omega]$ en sous-intervalles comme illustré en Figure 2.9, et leur associer des processus.

Intervalles de temps. On définit un nouveau type : *interval*, qui a deux valeurs : *inside* et *outside*. La construction d'un intervalle de valeur initiale $I0$ est notée : $I :=]B, E]$ *init* $I0$, où B et E sont les événements causant l'ouverture et la fermeture de l'intervalle. De façon répétée, I ouvre à la première occurrence de B , et est *inside* jusqu'à ce qu'il se ferme à la prochaine occurrence de E , et il est alors *outside* jusqu'à la prochaine ouverture, et ainsi de suite. Les événements bornes sont donnés par, respectivement : $O := \text{open } I$ et $C := \text{close } I$. Les intervalles sont *ouverts à gauche* et *fermés à droite* : les transitions (entrantes et sortantes de l'intervalle) ont lieu en réaction à un événement, et dépendent de l'état courant, résultant dans le nouvel état seulement *après* l'instant de réaction (comme il est usuel dans les automates réactifs).

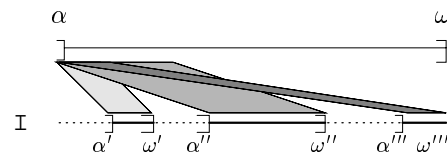
Les intervalles de temps peuvent être composés en expressions comme l'union $I := I1 \text{ union } I2$, le complémentaire $I := \text{comp } I$, ou l'intersection $I := I1 \text{ inter } I2$. La restriction d'un signal X à un intervalle de temps est notée : $XI := X \text{ in } I$, pour les occurrences de X à l'intérieur de I , et $XO := X \text{ out } I$ pour les occurrences à l'extérieur de I . On peut noter que $X \text{ out } I$ est équivalent à $X \text{ in } (\text{comp } I)$, et que $\text{open } I$ est $B \text{ out } I$, et $\text{close } I$ est $E \text{ in } I$.

Tâches et hiérarchies de préemption. Construire une tâche consiste à associer un sous-processus de l'application à un sous-intervalle de $[\alpha, \omega]$ sur lequel il est exécuté ou actif. Les processus usuels de SIGNAL sont des tâches actives depuis le début, et sans terminaison, c'est-à-dire sur tout $[\alpha, \omega]$: ils sont persistants sur toute l'application. Une tâche est active à l'intérieur de son intervalle, c'est-à-dire présente et contribuant au système d'équation global. En dehors de son intervalle, le processus est inexistant, les signaux qu'il définit absents, et la valeur de son état interne éventuel inaccessible : on lui a coupé l'horloge. Les tâches sont définies par le processus P à activer, l'intervalle I d'activation, et l'état (courant ou initial) dans lequel re-démarrer l'activité en ré-entrant, après une sortie de l'intervalle. Il s'agit donc bien d'une forme de préemption, contrôlant l'activité des processus.

Tâches suspensibles. Plus précisément, à la sortie de l'intervalle d'activité, un processus a atteint un état de sa mémoire interne (en SIGNAL : l'ensemble des délais (opérateur $\$$ et sous-intervalles dans le processus). À l'ouverture suivante de l'intervalle, on peut reprendre l'activité de P à cet état, c'est-à-dire là où il avait été *suspendu* : ceci est noté P on I . La Figure 2.10 illustre ce fonctionnement où le comportement que P aurait eu sur $[\alpha, \omega]$ est réparti sur les ouvertures successives de I .

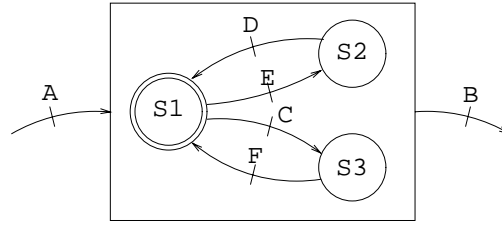

 FIG. 2.10 – GTi : tâche on l'intervalle I .

Tâches interruptibles. Une autre façon de re-démarrer l'activité d'un processus en ré-entrant dans intervalle d'activité est de choisir son état initial déclaré (par la déclaration de chacun de ses délais). Dans ce cas la tâche se comporte comme si la sortie de l'intervalle avait *interrompu* (de façon définitive) le processus. On note cette construction de tâche P each I . La Figure 2.11 illustre ce fonctionnement où le comportement que P aurait eu sur $[\alpha, \omega]$ est repris sur les ouvertures successives de I , par un préfixe à chaque fois. En ce sens, chaque ouverture successive de I est un nouvel $[\alpha', \omega']$, $[\alpha'', \omega'']$, $[\alpha''', \omega''']$, etc pour P .


 FIG. 2.11 – GTi : tâche each l'intervalle I .

Hiérarchies de préemption de tâche et séquençement. Les processus associés aux intervalles peuvent eux-mêmes être décomposés en sous-tâches : de cette manière on peut construire des hiérarchies de préemption pour décrire des comportements complexes. Le séquençement et la préemption de tâches est obtenue en contraignant les extrémités des intervalles, et en leur associant des activités en construisant des tâches hiérarchiques. Le parallélisme s'obtient naturellement quand les tâches partagent le même intervalles ou des intervalles se chevauchant. Le séquençement de tâches s'obtient simplement en contraignant les extrémités de leurs intervalles. En utilisant on et each on peut ainsi spécifier des systèmes de places/transitions parallèles hiérarchiques. Chaque intervalle de temps représente une information d'état, et les événements aux extrémités causent les transitions.

Par exemple, la Figure 2.12 illustre ceci avec une place initiale $S1$, d'où on peut transiter vers $S2$ sur l'occurrence de E , ou vers $S3$ sur celle de C . Si tous deux ont des occurrences simultanées, alors on entre dans les deux places. Les événements D et F font revenir vers $S1$. Ce comportement est un sous-comportement d'une place dans laquelle on entre par l'événement A et qu'on quitte par B . Ceci peut être codé en $SIGNALGTi$ par la hiérarchie de tâches de la Figure 2.13. Chacune des places $S1$, $S2$ et $S3$ y est encodée par un intervalle, avec ses événements d'entrée et de sortie ; par exemple l'entrée dans $S2$ a lieu à l'occurrence de E à l'intérieur de $S1$, c'est-à-dire E in $S1$. Ces trois intervalles constituent un processus qui, associé par each à l'intervalle $[A, B]$, forme une tâche codant l'ensemble.

FIG. 2.12 – *GTi : système place/transition hiérarchique.*

```
( | S1 := ]D, E default C] init inside
  | S2 := ]E in S1, D] init outside
  | S3 := ]C in S1, F] init outside
  | ) each ]A, B]
```

FIG. 2.13 – *GTi : hiérarchie de tâches codant le système de places/transitions.*

Exemple : le compteur transformé en chronomètre. Un autre exemple illustrant l'effet de la structure de tâche sur le comportement des processus flot de données est donné par un compteur transformé en chronomètre. La spécification en *SIGNALGTi* en est donnée en Figure 2.14. Dans la syntaxe *SIGNAL*, l'opérateur # *X* est un compteur entier, initialement à 0, du nombre d'occurrences du signal *X*. Ce compteur est associé à un intervalle *]R, S]* (pour *Resume* et *Suspend*) par *on*, constituant alors un compteur suspensible ou gelable. Associer celui-ci à l'intervalle *]B, E]* (pour *Begin* et *End*) par *each* en fait un compteur qu'on peut aussi arrêter et re-démarrer à sa valeur initiale 0.

```
( | I := ]B, E] init outside
  | ( | J := ]R, S] init inside
    | ( | Chrono := # Seconde | ) on J
    | ) each I | )
```

FIG. 2.14 – *GTi : spécification en SIGNALGTi du chronomètre.*

La trace montrée dans la Figure 2.15 illustre ceci : quand on entre dans *]B, E]* (qui est alors *inside*, noté \blacklozenge dans la Figure), on commence à compter le signal *Seconde*. Entre *S* et *R*, ce comptage est gelé. Sur l'occurrence de *E*, le chronomètre est arrêté ; à son re-démarrage, il repart de la valeur 0.

Modèle

Comme mentionné précédemment, la description de ce type de comportements est dans le pouvoir d'expression de *SIGNAL* ; il est possible de construire des programmes *SIGNAL* se comportant comme un programme *SIGNALGTi*. Une possibilité de définition de *GTi* est donc de définir une modélisation de ses structures en *SIGNAL*.

Les points essentiels à gérer sont la mémorisation de l'état, l'activation (ou l'inhibition) d'un sous-processus, et la réinitialisation de l'état du processus d'une tâche quand celle-ci est re-démarrée.

Mémorisation de l'état : les intervalles. Le codage d'un intervalle de temps peut être réalisé par un processus *SIGNAL* en utilisant le délai (opérateur de retard $\$$ de *SIGNAL*, seule forme de mémorisation) et en gérant adéquatement sa valeur [63], comme illustré en Figure 2.16.

Les valeurs *inside* et *outside* sont représentées par les Booléens *true* et *false* (type logique de *SIGNAL*). La valeur initiale de l'intervalle est donnée au retard codant son état dans sa déclaration. Le complément *comp* est la négation Booléenne. Les événements causant la transition entre les deux états sont les occurrences de *B* hors de *I : OPEN_I*, et de *E* dans *I : CLOSE_I*. Les intervalles sont ouverts à gauche et fermés à droite : leur valeur (en sortie de ce processus *SIGNAL*) est donc la valeur retardée, c'est-à-dire la valeur en mémoire : $I := \text{VAL } \$ 1$ et non la nouvelle valeur. Cette dernière est calculée dans le signal *NEW_VAL*, défini en fonction des événements *OPEN_I* et *CLOSE_I* (rappelons qu'un event de signal a toujours la valeur *true*, donc appliquer *not* rend *false* à la même horloge)

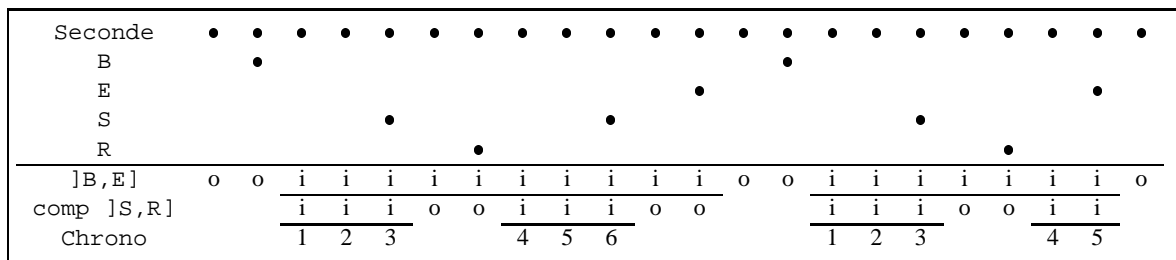


FIG. 2.15 – GTi : trace pour un tâche de comptage, contrôlée par les événements b, s, r et e.

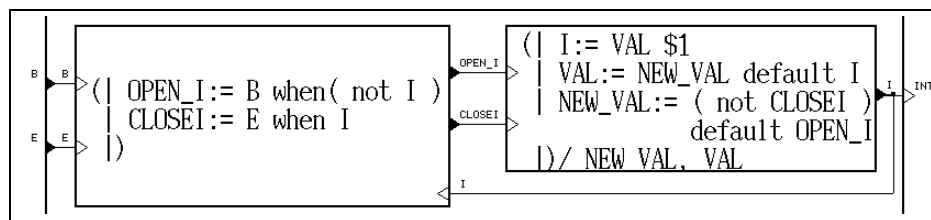


FIG. 2.16 – GTi : l'intervalle de temps en SIGNAL.

Activation : les tâches. L'activité des tâches doit être restreinte à l'intérieur de leur intervalle. Dans le cas de processus réactifs, dirigés par les entrées (où toute activité est causée par une entrée), une solution simple est de procéder à l'application d'un filtre sur ces entrées. De cette manière, on inhibe toute activité interne, la production de sorties comme l'évolution de l'état interne. Dans le cas non-réactif, où on a aussi un comportement interne en l'absence d'entrées (par exemple le suréchantillonnage de la Section 2.2.1), il faut avoir accès à l'horloge interne pour l'inhiber.

La Figure 2.17, correspondant à la hiérarchie de tâches de la Figure 2.14, illustre ceci en montrant sous forme de blocs diagramme, la structure des processus SIGNAL, avec l'insertion d'un filtrage (in), selon l'intervalle d'une tâche, sur les entrées du processus de cette tâche. En particulier, l'entrée X (qui correspond à Seconde) passe par deux filtrages (pour I et J) avant d'arriver au comptage (#).

Ré-initialisation : la hiérarchie de contrôle. Reste à assurer que dans le cas de tâches interruptibles, construites avec each, et redémarrées à chaque ré-entrée dans l'intervalle, on les fasse bien repartir de leur état initial. Ceci concerne toutes les variables d'état du processus de la tâche, c'est-à-dire tous les délais (\$ de SIGNAL) et donc les intervalles.

On applique une transformation, pour chaque équation $Y := X \ \$ \ 1 \ \text{init} \ V0$, qui consiste à faire en sorte que

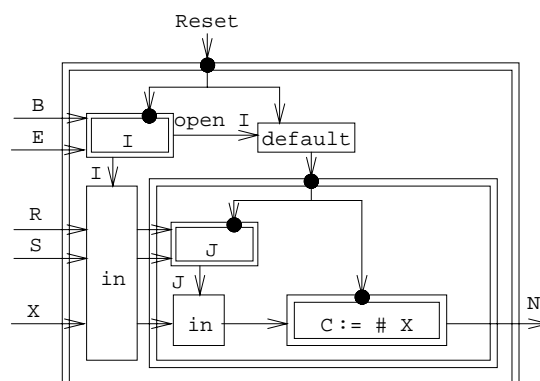


FIG. 2.17 – GTi : exemple de structuration hiérarchique du contrôle.

la valeur en sortie de Y soit, sur la présence d'un événement de ré-initialisation $Reset$, la valeur initiale déclarée $V0$:

```
Y_V := (V0 when Reset) default X
|   Y := (Y_V $ 1) when (event X)
```

Cet événement $Reset$ peut se définir, à chaque niveau de tâche $each$, comme l'union des ouvertures de l'intervalle de la tâche, et d'éventuelles réinitialisations causées par la ré-entrée dans la tâche englobante dans la hiérarchie. Au niveau le plus externe de la hiérarchie, ce dernier signal peut être considéré comme jamais présent. La Figure 2.17 illustre ceci, ainsi que le fait que les processus représentant les intervalles sont eux aussi soumis à réinitialisation.

Outil

L'outil qui concrétise tout ceci est un préprocesseur de l'environnement SIGNAL [99, 82]. Il a été appliqué dans divers contextes, décrits plus avant au Chapitre 3. Il est réalisé, avec Florent Martinez (stage de DEA en 1994), à partir des schémas de traduction de la modélisation. On a alors un outil construisant automatiquement, à partir de constructeurs adaptés, la structure de contrôle gérant l'activation de tâches.

Perspectives

Travaux liés. Ces travaux ont été accompagnés de travaux liés concernant les liens avec d'autres approches de la description de tâches dans les langages réactifs. Une étude a été menée en coopération avec Olivier Roux au LAN (IRCYN), avec Christine Sinoquet (stage de DEA en 1995), sur les relations entre ELECTRE et SIGNAL, doté des structures de *GTi*. ELECTRE comprend des structures de modules et des opérateurs de préemption, ayant donné lieu à une traduction en ESTEREL [91]. On a ici procédé à la modélisation en SIGNAL utilisant les intervalles de *GTi* [105].

D'autres travaux concernant la préemption dans les langages réactifs dans un cadre d'algèbre de processus sont évoqués en Section 2.2.3.

Suites. Les aspects principaux concernant la modélisation de hiérarchies de tâches préemptives en SIGNAL, que sont la mémorisation de l'état, la gestion de l'activation, et la gestion de la ré-initialisation, trouvent signification et usage dans les travaux ultérieurs concernant la modélisation des langages plus élaborés que sont STATEMATE et l'ensemble de langages de la norme IEC 1131 (voir Section 2.3).

Perspectives. Parmi les aspects intéressants qui poursuivraient ces travaux sur les intervalles de temps et les tâches en SIGNAL, se trouvent :

- l'élaboration d'une notion de tâches externe, avec la gestion de leur démarrage, suspension, reprise, interruption et terminaison
- la définition d'un modèle formel spécifique des intervalles, pour rendre compte de leur différence avec la notion d'instant en SIGNAL ; ceci pourrait faire intervenir la définition d'un calcul sur les compositions d'intervalles, éventuellement lié à des formalismes comme celui de Allen [2].

L'utilité de ce modèle serait de pouvoir intervenir dans les calculs de compilation (optimisation, vérification de consistance) comme représentation de la dynamique des programmes.

- l'utilisation de ces notions d'intervalle comme collection d'instant, et de contrôle de l'activité d'un processus, pour décrire la compilation de SIGNAL comme des raffinements successifs d'instant logique de la spécification vers des granularités plus fines (ce qui serait lié aussi aux perspectives de la Section 2.3.1).

Activités sur *GTi* et la gestion de tâches en SIGNAL

publications : journal (traitant aussi de l'application de vision robotique décrite en Section 3.2.1) [98], conférences (dont [96, 99]), rapport de recherche.

outil : Cette extension à SIGNAL, appelée SIGNAL*GTi*, a été mise en œuvre et intégrée à l'environnement SIGNAL, avec Florent Martinez (stagiaire de DEA), sous la forme d'un préprocesseur au compilateur : *GTi*. Il a été utilisé par les projets SIAMES et TEMIS à l'IRISA, le LAN (IRCYN) à Nantes, et dans le cadre d'une coopération avec EDF.

encadrement :

Stages de DEA en Informatique (IFSIC, Université de Rennes 1) de Florent Martinez : *Séquençement de tâches flot de données et intervalles de temps en SIGNAL* (1994) [82] et Christine Sinoquet : *Étude des liens entre les langages réactifs asynchrone et synchrone; application à ELECTRE et SIGNAL* (1995) [105] (co-encadré avec Olivier Roux, LAN (IRCYN), Nantes).

collaborations : Outre les collaborations avec Électricité de France (EDF), et les projets TEMIS (maintenant VISTA) et SIAMES, concernant les applications et détaillées plus loin (Chapitre 3) :

1995, avec Olivier Roux du LABORATOIRE D'AUTOMATIQUE DE NANTES (LAN, depuis renommé IRCYN), concernant les liens entre le langage ELECTRE et SIGNALGTi (voir stage de DEA de Christine Sinoquet [105]).

2.2.3 PAL : la préemption dans les langages réactifs

L'introduction, dans GTi, de structures d'interruption et de suspension de tâches en fait une forme de préemption. Parmi les travaux sur la préemption dans le cadre des langages synchrones, on trouve une algèbre définie à partir des opérateurs d'ESTEREL [21] qui situe les uns par rapport aux autres les différents opérateurs. L'étude brièvement résumée ici concerne la recherche d'un cadre unifié pour la spécification de la sémantique des primitives de préemption dans les langages réactifs. Elle est menée avec Sophie Pinchinat (projet EP-ATR) et R.K. Shyamasundar, du TATA INSTITUTE OF FUNDAMENTAL RESEARCH (TIFR) à Bombay (Inde), dans le cadre d'une coopération Franco-Indienne soutenue par le CEFIPRA.

Les opérateurs de préemption dans les langages réactifs

Ce travail a commencé par l'étude de la préemption dans les langages SIGNAL (et SIGNALGTi) et ESTEREL [88]. On se définit un algèbre d'opérateurs inspirée de MEIJE [34], et comportant des opérateurs de préfixage, composition parallèle, «ticking», restriction, et une primitive de préemption de base. Leur sémantique est définie en termes de règles de sémantique opérationnelle structurelle (SOS).

On y encode des opérateurs d'apparences différentes d'ESTEREL et de SIGNALGTi. Pour ESTEREL, il est rendu compte de la préemption forte (où le comportement du sous-processus préempté est inhibé dans l'instant de la préemption) et de la préemption faible (où le comportement du sous-processus préempté fait sa dernière transition dans l'instant de la préemption), ainsi que de la suspension (où le comportement du sous-processus préempté reprend depuis son état à l'instant de la préemption) et des opérateurs immédiats et retardés (où les événements ou actions sur l'occurrence desquels on préempte sont prise en compte respectivement dès l'instant de démarrage, ou à partir de l'instant suivant). Concernant SIGNALGTi (plus particulièrement les structures de GTi), on a décrit en règles SOS le comportement d'un intervalle (ses transitions en présence des événements d'extrémités), ainsi que celui des tâches susceptibles (on) et interruptibles (each). Ces différentes structures sont elles aussi encodées dans l'algèbre.

Ces travaux ont mené à la définition de l'algèbre de processus PAL (Process Algebra Language) pour servir de cadre commun dans lequel modéliser les différents opérateurs en présence. On s'attend à pouvoir intégrer à ce cadre des langages comme LUSTRE (d'une façon proche de SIGNAL) et ARGOS (d'une façon proche d'ESTEREL).

Expressivité des opérateurs de préemption

Le cadre de l'algèbre PAL conduit naturellement à étudier les propriétés de minimalité du jeu d'opérateurs qui y sont introduits, autrement dit les propriétés d'expressivité de ces opérateurs les uns par rapport aux autres. On propose un cadre formel basé sur les *systèmes de sémantiques opérationnelles structurelles* pour caractériser les opérateurs de préemption, et identifier les sous-classes d'opérateurs d'interruption et de suspension pour lesquelles nous avons établi des résultats d'expressivité comparée [89].

Perspectives

Des perspectives quant à ce travail concernent l'étude effective d'autres langages réactifs, la considération dans le cadre de GTi d'autres orientations d'intervalles $[, [,] , [, [,]$, avec préemption faible/forte pour les tâches, ou de structures de points de reprise (*checkpointing*).

collaborations : Collaboration Franco-Indienne (Convention IFCPAR/Inria n°195C0250031301005, 9/94 – 8/97) avec l'IISC (Bangalore) et le TIFR (Bombay) sous l'égide du Centre Franco-Indien pour la Promotion de la Recherche Avancée (CEFIPRA/IFCPAR). Il s'agit d'utiliser des variantes du paradigme synchrone pour modéliser et spécifier les systèmes de production automatisés.

Séjours au TIFR à Bombay, dans le groupe d'informatique théorique du Prof. R.K. Shyamasundar, en Mars-Avril 1995, Décembre 1995, Février-Mars 1997, Juillet-Août 1997.

2.3 Modélisation de langages de contrôle

Les structures proposées dans SIGNAL et *GTi* sont très simples dans l'objectif de favoriser leur étude et la définition de modèles et calculs associés. Il est possible de définir de constructions dérivées pour fournir à un utilisateur ou programmeur des supports à la structuration des spécifications et du confort d'écriture. Elles prennent la forme d'opérateurs macro-expansés, ou de déclarations de processus rassemblées en bibliothèques, ou encore d'extensions de langage traitées par pré-processeur comme c'est le cas de la mise en œuvre de *GTi*.

D'autres langages, comme ceux de STATEMATE ou de la norme IEC 1131 de programmation des automatismes industriels, sont définis d'emblée équipés d'ensembles de constructeurs beaucoup plus étendus et complexes. Ils sont plus élaborés, au sens où ils sont moins primitifs, et proposent des dérivations diverses des objets de base. Ils sont aussi plus répandus, dans les milieux industriels et académiques, correspondant de manière plus classique à la culture des utilisateurs. Ils cumulent éventuellement plusieurs formalismes différents, et manipulent des entités et concepts variés, de façon à convenir au mieux à leurs utilisateurs que sont les concepteurs de systèmes, à leurs besoins et à leur culture. Or ces langages n'en reposent pas moins sur les mêmes modèles sous-jacents, faisant intervenir des notions de réaction à un stimulus de l'environnement, d'états, de transition, d'action déclenchée, de détermination de l'ensemble des actions causées éventuellement indirectement par un événement en entrée. Ces langages sont définis de manière assez informelle, d'un point de vue essentiellement opérationnel, et en bloc, tous leurs éléments étant considérés au même niveau d'abstraction. Ceci ne favorise guère la clarté de leur compréhension (que ce soit par les utilisateurs ou par les concepteurs de leurs outils d'analyse ou de mise en œuvre), ni les possibilités d'analyser (*a fortiori* automatiquement) les spécifications.

Ceci motive qu'on s'attaque à construire des modélisations de ces langages qui soient à la fois formelles et exploitables dans des environnements de conception automatisés. En effet :

- cet effort de formalisation a l'intérêt de contribuer à l'étude des langages réactifs, en clarifiant certains aspects, et en fondant ces travaux sur des modèles inambigus : par exemple l'étude de l'interopérabilité dans le multi-formalisme ;
- cette modélisation présente l'avantage de permettre un accès effectif depuis ces langages de spécification, répandus chez les utilisateurs, vers les technologies de compilation, vérification et génération de mises en œuvre issues des travaux sur l'approche synchrone ; ceci est illustré par les travaux du projet SACRES et autour des formats d'échange comme DC+ (voir Section 2.2.1). De plus, cette base d'utilisation élargie peut occasionner un plus important retour expérimental.
- les travaux sur la préemption et le multi-formalisme dans les langages synchrones, parmi lesquels mes travaux autour de *GTi* présentés en Section 2.2.2, ont rendu disponibles les éléments de base pour se livrer à cette modélisation, qui peut prendre la forme d'une traduction encodant les langages considérés en SIGNAL en l'occurrence, en réutilisant les structures essentielles.

Dans cette Section, je présente synthétiquement mes contribution concernant la modélisation en SIGNAL des langages de STATEMATE (Section 2.3.1) et ceux, plus récemment démarrés, sur les langages de la norme IEC 1131 de programmation des automatismes industriels (Section 2.3.2). Ces travaux se situent dans l'approche mentionnée en état de l'art dans la Section 2.1.3.

2.3.1 STATEMATE : STATECHARTS et ACTIVITYCHARTS

Motivations

Ce travail s'est fait avec Jean-René Beauvais (thèse soutenue en février 1999 [14]), Thierry Gautier et Roland Houdebine et Yan-Mei Tang (ingénieurs-expert), en lien avec le projet Esprit SACRES. L'interopérabilité entre STATECHARTS et SIGNAL est étudiée sous l'angle d'une traduction des formalismes de STATEMATE en équations SIGNAL ou DC+. Dans STATEMATE, la structuration en sous-processus et le contrôle de leur activité se rattachent à une problématique de gestion de tâche et de préemption. Cette étude fournit aussi les bases d'un accès, à partir

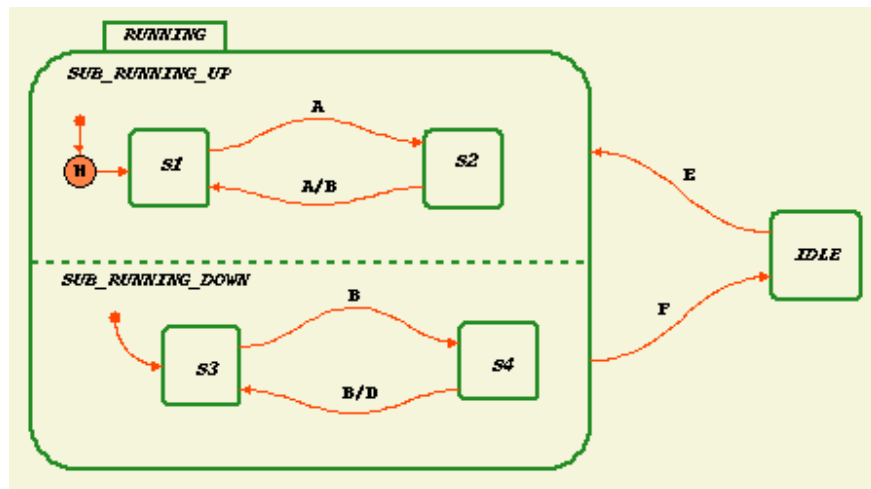


FIG. 2.18 – STATEMATE : un exemple de STATECHARTS dans MAGNUM.

de STATEMATE, aux fonctionnalités et outils (analyse, vérification, génération de code, ...) de l'environnement SIGNAL/DC+. Des schémas de traduction des STATECHARTS vers SIGNAL ont été définis pour la version des STATECHARTS telle que proposée par David Harel et supportée par l'outil commercial d'I-LOGIX qui en est issu [15, 16]. Ces schémas couvrent l'essentiel des fonctionnalités : concurrence et hiérarchie d'automates, gestion de l'historique des sous-automates suspendus, actions complexes, activités liées par des flots de données (dans ACTIVITYCHARTS). Ces schémas sont ensuite utilisés pour implémenter un traducteur automatique vers le format DC+, dans le cadre du projet Sacres.

Les langages de STATEMATE

Le formalisme des STATECHARTS a été introduit par Harel [50], et se définit en termes d'états, de transitions, de hiérarchie et de composition d'automates. Il s'est enrichi pour devenir un ensemble appelé STATEMATE [51], comprenant aussi les ACTIVITYCHARTS, un langage de structuration bloc-diagrammes, et un langage élaboré de spécification des actions associées aux transitions, proche des langages séquentiels impératifs. On en rappelle ici quelques traits essentiels, la description détaillée étant beaucoup plus longue, et sortant du cadre de ce document.

Les STATECHARTS. Comme l'illustre la Figure 2.18, un automate STATECHARTS fait une transition d'un état à l'autre (aussi appelée réaction) quand l'événement qui l'étiquette est présent : ici on alterne entre les états *Idle* (initial, marqué par une transition sans origine dite par défaut) et *Running* sur l'occurrence des événements *e* et *f*. On peut construire une hiérarchie en raffinant un état (ici *Running*) en un état-ET, dans lequel on réside concurremment dans les sous-états (*Sub_Running_Up* et *Sub_Running_Down*); c'est-à-dire que la réaction de l'état-ET est la conjonction simultanée des réactions des sous-états. On peut aussi utiliser le raffinement en état-OU, où on réside dans exactement un des états de l'automate (dans l'exemple : les sous-états *S1* et *S2* de *Sub_Running_Up*). On appelle configuration l'ensemble des états, structuré hiérarchiquement, où on réside.

Quand une transition fait quitter un état, tous ses sous-états sont également quittés. Quand on entre dans un état, il y a diverses façons d'aborder ses sous-états-OU. Si la transition par défaut a pour destination un état, le sous-automate est (ré-)initialisé dans celui-ci. Si elle pointe vers un connecteur d'historie *H*, on réside dans l'état où on résidait quand l'état-OU a été quitté (voir *Sub_Running_Up*). S'il s'agit d'un connecteur d'historie profonde *H**, la règle du connecteur d'historie est appliquée récursivement à tous les sous-états.

Les transitions entre états sont étiquetées, sous la forme $e[c]/a$, où *e* est un événement, *c* un condition booléenne, et *a* une action. La présence de l'événement et la satisfaction de la condition conditionnent la tirabilité de la transition. Si la transition est tirée, l'action est exécutée. L'exemple illustre l'émission d'événements (*b* dans *a/b* et *d* dans *b/d*). Ces événements sont diffusés et leur présence peut être ressentie simultanément dans tout le STATECHARTS. Cette diffusion est le support de la communication entre sous-états d'états-ET. Ces étiquettes peuvent aussi être associées aux états, sous le nom de réaction statique : leur tirabilité est évaluée à chaque fois que l'on réside dans l'état.

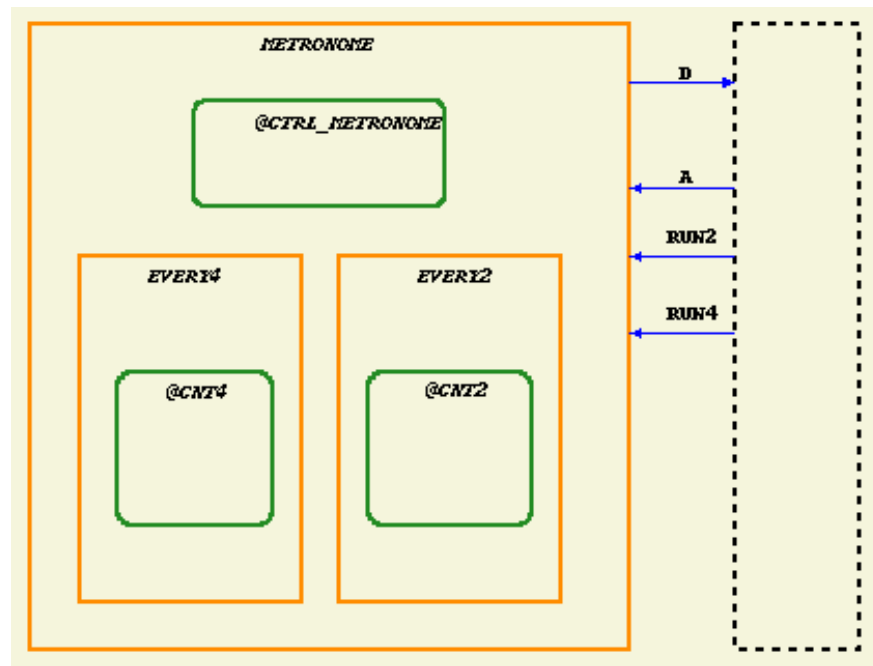


FIG. 2.19 – STATEMATE : un exemple d'ACTIVITYCHARTS dans MAGNUM.

Le langage des actions. Hormis l'émission d'événements, les actions peuvent être composées à partir d'un langage qui comprend l'affectation à une variable, le chien de garde `timeout`, le lancement différé d'action `schedule`, la conditionnelle Booléenne, la conditionnelle événementielle, et l'itération bornée (`for`). On y trouve aussi une itération conditionnelle non-bornée (`while`), et des variables dites «de contexte» qui, à la différence des variables dites «ordinaires», peuvent prendre plusieurs valeurs au cours d'un *step*.

Les ACTIVITYCHARTS. Les ACTIVITYCHARTS, dont un exemple est donné en Figure 2.19, consistent en une structure de blocs-diagrammes hiérarchique, où certaines des activités sont des activités de contrôle définies par un STATECHARTS, et les activités peuvent être démarrées, suspendues, redémarrées ou stoppées. On peut leur associer des actions conditionnées de la même forme que les étiquettes des transitions.

Les schémas d'exécution. Deux schémas d'exécution sont proposés pour STATEMATE, qui donnent pour une même spécification deux sémantiques différentes vis-à-vis des traces perçues en entrée :

- Le mode *step* : un pas (ou *step*) commence par acquérir les entrées venant de l'extérieur, et se fait sur la configuration et la valeur des variables en début de pas. À partir de là sont tirées les transitions et les actions correspondantes sont exécutées. Leur effet, avec la nouvelle configuration, constitue le résultat du pas. Ceci signifie qu'il y a un décalage d'un pas entre l'émission d'un événement et sa prise en compte pour déclenchement d'une transition.
- Le mode *superstep* : Il se différencie du mode *step* en ce qu'à la suite de l'acquisition des entrées, on enchaîne les *steps*, le premier prenant en compte ces entrées, les suivants les résultats internes du *step* précédent, ceci jusqu'à ce qu'un *step* ne produise plus aucun effet de nature à déclencher une transition. On se resynchronise alors avec l'environnement extérieur pour acquérir de nouvelles entrées. On peut noter que ce mode de fonctionnement s'apparente au GRAFCET en interprétation avec recherche de stabilité (voir Section 2.3.2), et présente des risques de non-terminaison, et donc de blocage vis-à-vis de l'environnement.

Les langages de STATEMATE sont distribués industriellement sous la forme de l'environnement MAGNUM par la société I-LOGIX¹⁴.

14. Site web pour MAGNUM : <http://www.ilogix.com>

Modèle

L'approche suivie pour construire un modèle formel des langages de STATEMATE est d'utiliser SIGNAL pour définir structurellement le comportement des éléments des langages, en bénéficiant des avantages de structuration et ré-utilisation dans le langage SIGNAL. On est aussi alors très près d'un support effectif de connection avec les outils synchrones, par l'intermédiaire du format DC+.

Nous ne donnons ici que les principes généraux de la modélisation, détaillée ailleurs [16, 14]. On peut remarquer le lien avec la structure du modèle SIGNAL de *GTi*, en ce qu'il s'agit dans les deux cas d'une construction hiérarchique où des signaux de contrôle gèrent l'activation et la ré-initialisation, au travers d'une propagation faisant intervenir état et transitions locaux.

On insiste un peu plus sur deux aspect plus particuliers de notre approche, utilisant des spécificités de SIGNAL : la représentation du non-déterminisme, et l'«accélération interne» par rapport à l'horloge des entrées (le sur-échantillonnage).

Éléments de base. Dans la traduction structurelle de STATEMATE vers SIGNAL, la hiérarchie des activités comme des STATECHARTS sera rendue par une hiérarchie de processus SIGNAL. Pour la construire, on réutilise quelques éléments de base. Ceux-ci comprennent, dans leur interface, des signaux spécifiques, dont la propagation dans la hiérarchie structurelle interviendra dans la gestion de l'activation et de la ré-initialisation :

- une horloge globale (celle du *step*) ;
- une horloge locale, qui est celle de l'activation d'une partie locale de la spécification ;
- l'événement de réinitialisation de l'état ;
- un signal propageant le temps, à destination de *timeout* et *schedule* ;
- et en sortie un signal Booléen indiquant si la stabilité a été atteinte localement (pour le mode *superstep*).

Mémorisation de l'état. L'état est en fait réparti dans le contrôle explicite, et dans les données, qui nécessitent aussi une mémorisation.

Contrôle. Un processus standard gère, pour chaque état de STATECHARTS, la mémorisation de la configuration, et la possibilité de le ré-initialiser.

En suivant la hiérarchie des états et sous-états :

- pour chaque état-OU, on instancie un processus de gestion de son état, un processus de calcul de réceptivité pour chaque transition, et un processus pour chaque sous-automate, récursivement.
- pour chaque état-ET, on procède à la simple composition des processus des sous-états.

Les ACTIVITYCHARTS sont traités comme des processus dotés chacun d'un automate contrôlant leur activation, en réaction à des actions de lancement, suspension, reprise et arrêt. L'horloge d'activité est sous-échantillonnée par la présence dans l'état actif.

Données. Par ailleurs, la sémantique de STATEMATE impose, à la différence de celle des langages synchrones, le report des effets des actions (émission d'événements, affectations à des variables, ...) au *step* suivant. Il faut donc encoder explicitement la mémorisation des effets d'une action, en utilisant le délai synchronisé à l'horloge globale ; en ce qui concerne les événements, la mémorisation se fait au travers d'un Booléen.

Activation. La Figure 2.20 résume la structure hiérarchique générale du modèle, et illustre particulièrement le contrôle de l'activation et de la ré-initialisation.

Contrôle. Le contrôle de l'activation de sous-comportement se fait en filtrant hiérarchiquement l'horloge : dans un état-OU, le sous-comportement associé à un état se verra transmettre une horloge qui est l'horloge locale sous-échantillonnée aux instants où on est dans l'état en question. Par exemple en Figure 2.20, on construit *clk1* dans le cas où l'état *s* vaut *S1* ; on peut remarquer que la structure de *GTi* illustrée en Figure 2.17 correspond à un cas où l'état peut prendre deux valeurs. C'est au travers de ceci qu'on pourra établir une hiérarchie d'horloges, qui correspondra à la hiérarchie de la spécification, et dont le compilateur SIGNAL peut faire usage. Dans un état-ET, l'horloge locale est propagée aux sous-états.

Dans les ACTIVITYCHARTS, l'horloge locale est sous-échantillonnée aux instants où elle est démarrée et non suspendue. Elle est propagée aux sous-activités.

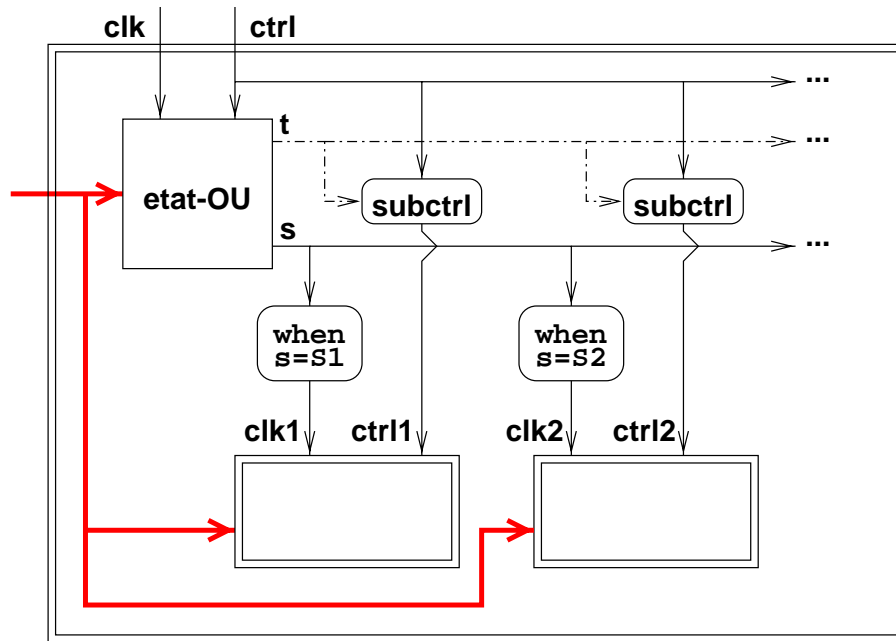


FIG. 2.20 – STATEMATE : structure hiérarchique générale du modèle.

Actions. La présence dans un état sert aussi à activer le calcul des conditions ou réceptivités des transitions qui le quittent. Celles-ci, quand elles débouchent sur la tirabilité de la transition, définissent l’horloge d’activation des actions de la transition. Le langage des actions fait l’objet d’une traduction dans l’instant [84], dans le sens où la séquence des instructions est traduite en style flot de données; les conditionnelles sont traitées par sous-échantillonnage des signaux passés aux calculs. Toutefois, les itérations, en particulier les boucles non-bornées (*while*) ne peuvent rentrer dans ce cadre. On donne plus loin des pistes pour pouvoir les modéliser.

Ré-initialisation. Le contrôle de la réinitialisation est lui aussi propagé aux sous-processus avec modification selon la présence d’un connecteur historique. De façon comparable à ce qui se passe dans *GTi*, on reçoit par l’interface de signaux spécifiques un éventuel signal de ré-initialisation venu d’un niveau supérieur dans la hiérarchie structurale. Celui-ci est propagé à toutes les sous-activités ou aux sous-automates.

C’est dans le cas des états-OU qu’on doit distinguer les cas en fonction du connecteur d’histoire, et donner à ce signal des valeurs différentes selon qu’il s’agit de ré-initialiser l’état du niveau courant et plus bas, ou seulement plus bas, ou pas du tout. La Figure 2.20 illustre le fait que le signal de contrôle transmis (par exemple *ctrl1*) dépend de celui reçu de niveaux supérieurs de la structure hiérarchique, et des transitions du niveau courant (*t*). Dans la de *GTi*, illustré en Figure 2.17, c’est la transition *open* qui influe sur la ré-initialisation d’une tâche construite par *each*.

Non-déterminisme. Il est possible d’utiliser SIGNAL pour modéliser le non-déterminisme dans le sens où l’on peut définir des processus ayant un ensemble possible de comportements. Cette possibilité est utilisée pour modéliser le non-déterminisme des STATECHARTS. Ici, on s’intéresse au cas où plus d’une transition est tirable à un instant donné. Si on se réfère à la sémantique de STATEMATE [51], un ensemble de transitions sans conflit est défini de la façon suivante :

- Deux transitions tirables sont *en conflit* s’il existe un état commun qui est quitté si l’une d’elles est tirée,
- Un ensemble de transitions est *sans conflit* si aucune paire de transitions de l’ensemble n’est en conflit,
- Etre *maximal* pour un ensemble de transitions sans conflit signifie que chaque transition non contenue dans l’ensemble est en conflit avec au moins une transition contenue dans l’ensemble. Sinon, cette transition pourrait être ajoutée à l’ensemble.

A chaque *step*, STATEMATE choisit un ensemble maximal de transitions sans conflits. Lorsqu’il y a plus d’un seul ensemble tirable, un choix non-déterministe est effectué.

Le non-déterminisme dans le choix de deux transitions peut être représenté explicitement en SIGNAL par l’ajout d’un Booléen κ que l’on appelle *oracle* à l’équation SIGNAL qui choisit entre les différentes transitions tirables. Si

t_1 et t_2 sont des signaux portant chacun la valeur de l'état destination de la transition correspondante, aux instant où sa réceptivité est satisfaite et où l'état courant est son état source, on peut définir t la transition choisie par :

$$t := t_1 \text{ when } K \text{ default } t_2 \\ | \wedge t_1 \text{ when } \wedge t_2 \wedge \wedge t_1 \text{ when } \wedge t_2 \text{ when } \wedge K$$

La première équation dit que t prend l'une des deux valeurs selon la valeur de l'«oracle» K . La deuxième équation de synchronisation, qui peut aussi s'écrire $\widehat{t}_1 \cap \widehat{t}_2 \subseteq \widehat{K}$, spécifie que l'oracle est présent au moins aussi souvent que nécessaire pour déterminer le choix. La simplification en $t_1 \text{ default } t_2$ a le comportement d'un processus déterministe faisant un choix arbitraire, au sens où l'une des transitions est privilégiée par la priorité de l'opérateur `default`.

A cette étape, nous avons donc un modèle exact du comportement non-déterministe de cette partie de la spécification. En l'absence de définition pour K , nous avons affaire à une variable libre dans le système d'équations d'horloges : dans les analyses du calcul d'horloge, de même que dans la vérification de propriétés dynamiques, l'ensemble des valeurs possibles sera pris en compte. Si on ne peut pas exécuter une telle spécification, on peut donc néanmoins lui appliquer les méthodes de vérification formelle. Par ailleurs, le non-déterminisme peut parfois être résolu : si t_1 et t_2 sont exclusifs, alors l'«oracle» est inutile et peut donner lieu à une simplification, ou bien on peut décider de connecter ces «oracles» sur des entrées, demandant ainsi à l'environnement de fournir leurs valeurs.

Granularités de temps : *step* et *superstep*. Une des spécificités de SIGNAL parmi les langages synchrones est de généraliser l'approche des systèmes purement réactifs (*fonctions* de leurs entrées et état interne) à des relations; notamment ceci signifie qu'on peut décrire des systèmes qui pour une entrée donnée peuvent avoir plusieurs comportements possibles, ou qui n'ont pas besoin d'entrée pour avoir un comportement, comme on l'a mentionné en Section 2.2.1. Ce point constitue la «pro-activité» de SIGNAL, qu'on appelle parfois le sur-échantillonnage. L'exemple classique d'un processus SIGNAL dont l'activité interne (et en l'occurrence les sorties) est plus fréquente (qualitativement parlant) que les entrées est celui du décompteur donné en Figure 2.1. L'horloge interne (et des sorties) inclut celle des entrées, comme le montre la trace de la Figure 2.2. C'est l'état interne du processus (dans l'exemple, la valeur de la variable d'état interne) qui décide de la synchronisation avec l'environnement pour acquérir les entrées. Il se trouve que dans certaines conditions, le compilateur SIGNAL peut même synthétiser un code exécutable pour de telles spécifications ; dans d'autres cas on a des spécifications présentant par exemple plusieurs «accélération internes», pour lesquelles on n'a pas de schéma d'exécution, mais qui sont accessibles aux outils d'analyse par ailleurs.

Dans le cadre de la modélisation de STATEMATE, il se trouve deux possibilités d'application de ce phénomène, à deux niveaux imbriqués :

- le mode *superstep*, comme on l'a dit, consiste en une acquisition d'entrées suivie d'un enchaînement de *steps* réagissant chacun aux actions du *step* précédent, jusqu'à stabilité, c'est-à-dire absence de transition tirable. Cette stabilité peut s'évaluer à chaque *step*, et décider de la continuation dans l'enchaînement ou bien de la synchronisation avec un nouveau *superstep*. La synchronisation ou non de l'avancement du temps (à destination des actions `timeout` et `schedule`) définit alors divers modes d'exécution de STATEMATE.
- dans le langage des actions : les itérations conditionnelles ont un caractère non-borné qui interdit leur «déplie» dans un flot de données dans l'instant. Elles font intervenir des variables dites «de contexte» qui peuvent prendre plusieurs valeurs dans un *step*. Une solution est de construire une horloge sur-échantillonnée par rapport à celle du *step*, tant que la condition de sortie n'est pas satisfaite. L'exécution des actions est alors répartie sur une série d'instant, définissant ainsi une horloge de *microstep*.

Le sur-échantillonnage de SIGNAL représente ainsi des niveaux d'imbrication de façon homogène : il pourrait en être imaginé une profondeur arbitraire.

Outil

Ces schémas de traduction brièvement résumés précédemment sont utilisés par Roland Houdebine et Yan-Mei Tang pour implémenter un traducteur automatique vers le format DC+, dans le cadre du projet Sacres (voir Section 2.2.1). Ce traducteur est interfacé avec la version commerciale de STATEMATE (plus précisément à ANAPORT, l'A.P.I. (*Application Programming Interface*) de génération de code C de l'environnement MAGNUM). De l'autre côté il est interfacé avec l'A.P.I. de manipulation d'arbres de syntaxe abstraite de la machine virtuelle DC+. Il produit du code DC+, qui peut alors être utilisé comme point d'entrée des outils de génération de code séquentiel ou distribué, de preuve, d'évaluation de performances.

Perspectives

Parmi les perspectives autour de ces travaux, on peut mentionner :

- des améliorations de ce modèle lui-même, pour l’adapter de façon optimisée aux traitements par les outils synchrones,
- la ré-utilisation de ces concepts pour la modélisation de langages de la famille des STATECHARTS : les variantes directes comme UML, ou des langages différents mais présentant des similarités de structure et de comportement, comme les langages de la norme IEC 1131-3 (voir Section 2.3.2).
- l’approfondissement sur les notions de granularités de temps, pour la modélisation des mises en œuvre de programmes synchrones : ayant un modèle formel d’un degré de granularité, et du suivant, on pourrait analyser la relation entre l’un et l’autre, et décrire comment décomposer un instant logique en séquence d’opérations non-simultanées.

Activités liées à STATEMATE

publications : conférences (dont [15]), rapport de recherche [16], rapports de contrat Européen.

Soumission à une revue.

outil : traducteur *stm2dc+* de StateMate en DC+ (Roland Houdebine et Yan-Mei Tang, ingénieurs-experts); développé dans le cadre du projet SACRES, en coopération avec la société I-LOGIX.

encadrement :

Co-encadrement, avec Paul Le Guernic, de la thèse de Doctorat en Informatique (IFSIC, Université de Rennes 1) de Jean-René Beauvais : *Modélisation de STATECHARTS en SIGNAL pour la conception de systèmes critiques temps-réel* (thèse soutenue en février 1999) [14].

Stage de DEA Informatique (IFSIC, Université de Rennes 1) de Mirabelle Nebut : *Modélisation de STATEMATE en SIGNAL : le langage impératif des actions* (1998) [84].

collaborations :

Projet Esprit R&D SACRES (EP 20897, convention n° 195C4170031307006, 11/95 – 12/98) (voir Section 2.2.1) notamment avec ILOGIX.

2.3.2 GRAFCET et la norme IEC 1131-3

Motivations

Dans le domaine des automatismes industriels, et en particulier de la conception de leurs contrôleurs, on rencontre des environnements, des normes et des langages de programmation spécifiques. Ceci est occasionné par les architectures sur lesquelles les contrôleurs s’exécutent, par exemple les automates programmables, et un aspect culturel, au sens où certains métiers s’expriment dans des langages reconnus comme standard, et ne sont pas prêts à en changer sans raison impérieuse. Ces langages, dont GRAFCET est un exemple, ainsi que le standard IEC 848 qui le normalise pour la spécification, ou la norme IEC 1131 qui en donne une interprétation opérationnelle, ont pour but de décrire et exécuter le contrôle de processus. Présentant des aspects réactifs et une notion de cycle ou instant logique, ils ont des similitudes avec l’approche synchrone.

Il s’agit ici d’un travail, en commun avec Fernando Jiménez-Fraustro (en thèse), pour étudier la sémantique de langages de spécification de tels automatismes industriels, en vue de leur intégration à des outils de conception comportant des outils d’analyse, de transformation et d’implémentation. Cette intégration se fera au travers de la traduction et compilation de ces langages dans un formalisme synchrone, suivant une approche similaire à celle pratiquée concernant STATEMATE dans le cadre de l’environnement SACRES. Parmi les langages de la norme, on s’intéresse notamment au SEQUENTIAL FUNCTION CHARTS (une interprétation du GRAFCET) et au langage impératif séquentiel, ST, dont le traitement [55] peut réutiliser certains résultats des travaux sur STATEMATE.

Les langages de la norme IEC 1131

La norme IEC 1131 définit un cadre pour la programmation des automates programmables industriels [29]. En cela elle diffère d’autres normes, telles la IEC 848, qui traite d’un langage de spécification, qui n’est pas lié à la mise en œuvre ; on y trouve donc une interprétation différente du GRAFCET, un langage à la syntaxe graphique essentiellement

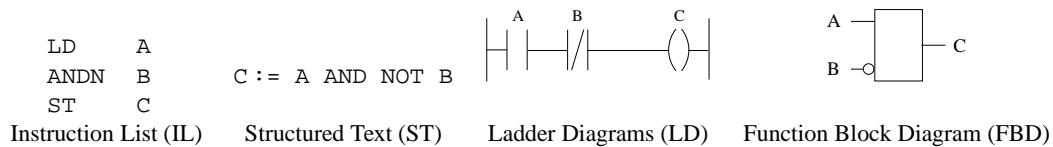


FIG. 2.21 – Norme IEC 1131-3 : les quatre langages.

similaire au SFC (SEQUENTIAL FUNCTION CHARTS ou DIAGRAMME SÉQUENTIEL DE FONCTION) de la norme IEC 1131. Cette norme couvre un certain nombre d'aspects du problème, et est organisée en cinq parties :

- l'introduction
- le matériel, décrivant les architectures sur lesquelles s'exécutent les programmes ;
- les langages de programmation, décrits chacun individuellement, avec aussi le contexte dans lequel ils interopèrent ;
- les préconisations aux utilisateurs, qui complètent la documentation des langages ;
- les communications, qui décrivent comment les automates programmables peuvent être mis en réseau, et comment les programmes les contrôlant peuvent communiquer.

La troisième partie, concernant les langages, est celle qui nous intéresse le plus : elle expose leur syntaxe et sémantique, sous une forme informelle. Les préconisations donnent des informations complémentaires. À plus long terme, dans une perspective de mises en œuvre dépendantes de l'architecture d'exécution, en l'occurrence des automates programmables, éventuellement connectés en réseaux, on s'intéresserait aussi aux parties décrivant le matériel et les communications.

Organisation des langages. Les langages de programmation sont organisés en deux parties : les éléments communs, et les quatre différents langages. Les éléments communs concernent tout ce qui est utilisé dans le cadre de tous les langages :

- les types de données élémentaires et les variables, avec des valeurs initiales par défaut, sont utilisés pour en construire des dérivés.
- les Unités d'Organisation de Programme sont les unités de structuration de base, qui peuvent être associées à des tâches dans l'environnement d'exécution. Elles peuvent être :
 - des fonctions (standard, ou définies par l'utilisateur), par exemple l'addition, la racine carrée, les fonctions logarithmiques. Elles n'ont pas d'état interne : toute invocation avec les mêmes paramètres donne les mêmes résultats.
 - les blocs-fonctions, qui contiennent dans la même entité les données et l'algorithme qui les manipule ; c'est-à-dire qu'elles ont un état interne, ce qui constitue la différence avec les fonctions.
 - les programmes sont construits avec des appels à des fonctions et des blocs-fonctions. Un programme peut être écrit dans l'un des quatre langages décrits plus loin.
- les Diagrammes Séquentiels de Fonction sont un langage graphique pour la modélisation des aspects fonctionnels et comportementaux des systèmes de contrôle à événements discrets.
- les éléments de configuration donnent le moyen de décrire l'implémentation du contrôleur à l'aide de variables globales, de ressources, de tâches et de chemins d'accès.

Une notion de «scan» définit le comportement de ces éléments dans un cycle de lecture des entrées, calcul de la réaction et émission des sorties.

Les quatre langages de programmation. La Figure 2.21 illustre les quatre langages de programmation, sur l'exemple d'un même programme simple. Il y a deux variables d'entrée A et B, de type Booléen, et une variable de sortie C du même type ; l'opération consiste à faire la conjonction de la valeur de A avec la négation de la valeur de B. Les quatre langages sont : *Instruction List* (IL), *Structured Text* (ST), *Function Bloc Diagram* (FBD), *Ladder Diagrams* (LD).

Les langages impératifs textuels : ST et IL. Deux des langages sont de forme textuelle, à structures de contrôle séquentielles et impératives :

Instruction List (IL) est de type langage d'assemblage, travaillant sur des variables, avec des instructions comme :

- le chargement en registre (LD), l'écriture en mémoire (ST), les opérateurs Booléens (AND, XOR, ...);
- les branchements, conditionnel ou non (JMP, ...);
- les appels à temporisations (CAL *timer*, ...).

Il existe aussi la possibilité d'appeler des blocs fonction (qui peuvent être écrits dans les autres langages).

Par exemple, la Figure 2.21 donne le programme chargeant la valeur de la variable A, procédant à sa conjonction avec celle de la négation de B, et enfin écriture dans C.

Structured Text (ST) est un langage structuré de la famille de Pascal ou Ada ; il travaille sur des variables éventuellement d'entrée/sortie, en appliquant en séquence les instructions dans une liste. Celles-ci peuvent être :

- des affectations $X := \text{expr}$ (la Figure 2.21 donne un exemple) ;
- la séquence, notée «;», où la première instruction est exécutée, puis la deuxième sur l'état de valuation des variables résultant de la première ;
- la conditionnelle IF *expr* THEN ... END_IF, qui exécute le sous-programme quand l'expression s'évalue à vrai ;
- l'itération bornée FOR *expr* DO ... END_FOR pour laquelle le sous-programme est répété en séquence un nombre de fois indiqué par *expr* ;
- l'itération non-bornée WHILE *expr* DO ... END_WHILE, où le corps est répété jusqu'à ce que l'expression *expr* s'évalue à la valeur *faux*, et sa variante REPEAT ... UNTIL *expr* END_REPEAT, où la condition est évaluée en fin de boucle. Cette expression est évaluée à chaque tour de boucle sur l'état courant des variables.

Les langages graphiques : FB (et FBD) et LD. Les deux autres langages sont graphiques :

Ladder Diagrams (LD) est fondé sur la notion de schéma à relais électro-mécanique. La Figure 2.21 en montre un exemple simple, où, entre les deux bornes représentées par des verticales, les éléments sur la gauche représentent les opérations (lecture/accès à la valeur de A, et celle de B avec négation, la mise en série correspondant à la conjonction) et à droite se trouve l'écriture ou production du résultat. Une connection en parallèle donne un connecteur logique disjonctif.

Function Bloc Diagram (FBD) fait la combinaison des fonctions et blocs-fonctions en diagrammes-blocs ou graphes à flots de données. Il comporte un certain nombre de blocs prédéfinis, tels qu'opérateurs Booléens, détecteurs de front, bascules. Dans la Figure 2.21, l'opérateur est une conjonction, et le cercle sur l'entrée B indique la négation.

En dehors des opérateurs prédéfinis, il est possible de définir des blocs-fonctions comprenant des données locales et une fonction d'évaluation écrite dans un des autres langages, qui à chaque appel calcule, en fonction des entrées et des variables locales la nouvelle valeur des sorties et des variables locales.

GRAFCET ou SEQUENTIAL FUNCTION CHARTS. Les SEQUENTIAL FUNCTION CHARTS sont dérivés du GRAFCET en en reprenant les aspects principaux de la syntaxe graphique et de règles définissant le comportement, mais en en donnant une interprétation distincte motivée par le domaine d'application et les spécificités de son utilisation pour la programmation et l'exécution.

Structures du langage. C'est un formalisme à base d'états lié aux réseaux de Petri [33, 32]. Il est constitué d'étapes (l'étape initiale étant distinguée par un contour en ligne double), liées par des transitions, et associées à des blocs d'action (voir Figure 2.22). Chaque étape représente un état particulier du système contrôlé. Une transition est associée à une condition (dans l'illustration : A, B, C) qui, quand elle est satisfaite, entraîne la désactivation de la place précédant la transition, et l'activation de la place qui suit. On peut utiliser des séquences présentant des alternatives ou des branches parallèles. Les étapes sont associées à des blocs d'action, dont le comportement ou calcul est décrit dans les langages de la norme : la Figure 2.22 montre une action écrite dans le langage ST, et comportant notamment une itération non-bornée. En effet, dans les applications, les SFC sont utilisés en conjonction avec les autres langages, car ils ne disposent pas de notation pour décrire les conditions ou les actions elles-mêmes.

Parmi les éléments disponibles pour structurer ces graphes, on dispose de branchements («divergences» et «convergences»), et aussi selon les variantes de formes de forçage de l'état d'un GRAFCET par un autre, de GRAFCET hiérarchique, de macro-étapes, d'encapsulation, ...

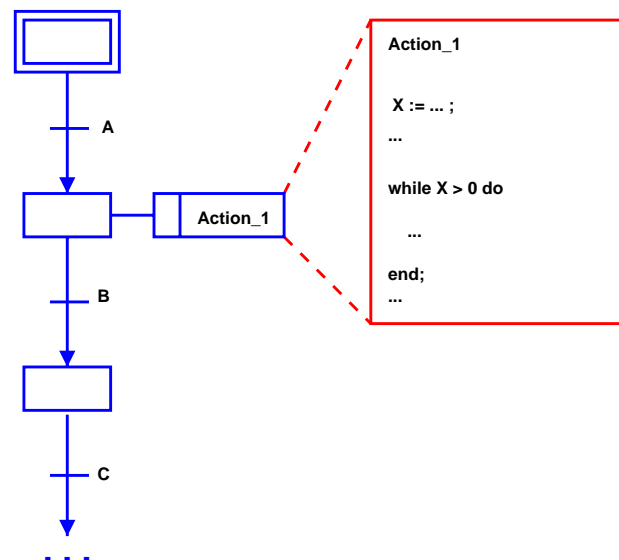


FIG. 2.22 – Norme IEC 1131-3 : un GRAFCET (SEQUENTIAL FUNCTION CHARTS) avec une action en ST.

Interprétations. Les évolutions de l'état d'un tel graphe sont définies par un ensemble de cinq règles, définissant en fonction de l'état d'activation des étapes, la tirabilité des transitions, et l'effet de leur tirage effectif vis-à-vis de l'activation des étapes comme des actions associées. Il existe plusieurs interprétations pour des graphes de type GRAFCET. Nous nous intéressons à la norme IEC 1131, qui concerne la programmation des contrôleurs en vue de leur exécution, mais aussi de façon plus large aux autres variantes du GRAFCET, notamment l'interprétation avec recherche de stabilité [66, 40].

Brièvement, on peut décrire ces deux interprétations comme :

sans recherche de stabilité où les entrées sont utilisées en combinaison avec l'état interne (étapes et variables) pour calculer la configuration immédiatement suivante, après quoi on attend l'entrée suivante. Dans ce sens, la stabilité est atteinte immédiatement.

avec recherche de stabilité où on enchaîne les transitions, la première prenant en compte les entrées, les suivantes consistant à tirer les transitions tirables jusqu'à épuisement.

Ces deux modes d'évolution (vis-à-vis de l'environnement extérieur) sont à mettre en parallèle avec les deux schémas d'exécution de STATEMATE (voir Section 2.3.1).

Le modèle

Les premiers résultats concernent la modélisation de ST, et des principes de représentation d'imbrications de granularités de temps réutilisant les résultats liés à STATEMATE. Le travail est en cours sur les autres aspects.

Modélisation de ST. Elle reprend des aspects classiques de la construction d'un graphe de dépendances de données à partir d'un texte impératif [55, 54]. L'idée, vis-à-vis de la modélisation dans un cadre synchrone, est que les calculs sont déroulés dans un instant logique, dans la mesure où ce déroulement est borné. Les variables de ST subissant des affectations au cours de la séquence devront alors être représentées par différents signaux, ceux-ci ne pouvant porter qu'une seule valeur par instant. Cette démultiplication des signaux représentant une même variable peut être appelée «expansion spatiale». Dans la cas, par contre, d'un déroulement non borné, qui est celui des boucles WHILE et REPEAT, on ne peut procéder de la sorte. Dans le cas des boucles bornées FOR, on peut par ailleurs mettre en question la pertinence de l'expansion spatiale dans le cas de bornes élevées. L'idée est alors, pour porter ces différentes valeurs affectées aux variables, de procéder à une «expansion temporelle», dans le sens où des instants logiques seront insérés de façon suffisante pour rendre du déroulement du programme. Cette technique reprend le phénomène d'«accélération interne» ou suréchantillonnage de SIGNAL, décrit en Section 2.2.1.

Plus précisément :

les variables sont mémorisées d'un *scan* à l'autre ; ceci donne lieu à des processus utilisant le délai \$, dont les sorties sont connectées en entrée du processus représentant le programme lui-même ;

l'affectation donne lieu à la définition d'un nouveau signal, qui prend la valeur produite par la traduction de l'expression ; le tout forme un processus dont l'entrée comporte les signaux représentant les variables apparaissant dans l'expression ;

la séquence consiste à établir entre les processus P_1 représentant le sous-programme avant l'opérateur $;$, et P_2 représentant la suite, les flots de données correspondant à leurs dépendances. Les entrées de P_2 qui ne sont pas produites par P_1 doivent être cherchées en amont dans la séquence, ou à la sortie de la mémorisation de variables.

la conditionnelle est modélisée par filtrage de ses entrées (utilisant `when`) par la valeur de la traduction de la condition ; les sorties sont le mélange (par `default`) des signaux sortant de la traduction du corps si la condition est vraie, et d'entrée si la condition est fausse.

l'itération bornée (déroulée) peut se faire, comme on l'a dit, par déroulement en séquence, ou en appliquant la technique rendue nécessaire par les itérations non-bornées.

les itérations non-bornées sont traitées par expansion temporelle ; elles sont traduites dans un processus qui s'enclenche quand il reçoit des signaux d'entrée, et ne produit de signaux en sortie pour une éventuelle séquence qu'à la sortie de boucle. Entre ces deux instants, il peut y en avoir où le processus prend en entrées supplémentaires les valeurs en mémoire des variables, évalue la traduction de la condition, et si elle est vraie, produit les sorties pour le passage en séquence, sinon, enclenche le calcul de la traduction de son corps, à la sortie duquel les mémorisations des variables sont mises à jour. L'état interne permet de déterminer quand l'itération doit continuer, et quand se «resynchronise» avec l'environnement, qui est ici le contexte du programme. Les schémas de traduction sont structurels, et peuvent traiter les boucles imbriquées ; l'instruction `EXIT`, qui fait sortir de la boucle courante, donne lieu à la gestion de signaux de contrôle supplémentaires.

Lien avec les interprétations du GRAFCET. Comme on l'a indiqué précédemment, le GRAFCET peut donner lieu à des interprétations diverses, avec ou sans recherche de stabilité. Même si la norme IEC 1131 ne le prévoit pas, on s'intéresse à la question, car un traitement pragmatique et clair de la recherche de stabilité peut offrir une fonctionnalité intéressante aux utilisateurs. Cette recherche de stabilité comporte de faire se succéder les pas jusqu'à trouver un état stable. Concernant le tirage de transitions et l'exécution d'actions, on s'expose donc à avoir à représenter l'exécution à chaque pas des actions correspondant aux transitions tirées et places activées.

Par ailleurs, si les actions elles-mêmes peuvent être définies par un bloc-fonction dont le corps est écrit dans un des langages de la norme, voire en GRAFCET, on voit que l'exécution d'une action peut donner lieu à une recherche de stabilité, introduisant un degré supplémentaire de raffinement d'échelle de temps. Enfin, ces actions, quand elles comportent des programmes en ST comprenant des boucles non bornées, introduisent elles aussi des niveaux supplémentaires de granularité.

On se trouve donc à envisager des hiérarchies de niveaux de granularité de profondeur arbitraire, et c'est là que l'aspect d'accélération relative offert par le mécanisme de sur-échantillonnage de SIGNAL permet d'envisager la construction structurelle, par des schémas de traductions définis localement aux constructions des langages à modéliser, d'un modèle de comportement rendant compte effectivement de la sémantique des programmes.

Perspectives

Comme on l'a dit, ces travaux sont encore en cours ; leur intérêt, vis-à-vis de l'étude de STATEMATE, est dans l'élargissement de la problématique multi-formalisme, et dans celui des domaines d'application. Les perspectives se trouvent dans :

- la couverture de la norme IEC 1131-3 : autres langages, individuellement : traitement effectif de SFC, étude de IL, FBD, LD, clarification de leur interopération. On envisage en fait de proposer des modèles des variantes, extérieures la norme IEC 1131 elle-même, comme les différentes interprétations des SFC/GRAFCET.
- la mise en œuvre d'un outil qui automatise la construction de modèle à partir de spécifications d'utilisateurs, éventuellement sous la forme d'un traducteur vers SIGNAL ou DC+ ; ceci pose la question de considérer la connection à un frontal graphique existant.
- l'utilisation effective des fonctionnalités des technologies synchrones comme la vérification ou les mises en œuvre distribuées, pour démontrer l'utilité de l'approche, et explorer les possibilités d'exploiter des spécificités des langages.
- l'application à une étude de cas, qui permette de valider l'utilisabilité des méthodes.

encadrement : Co-encadrement en cours, avec Paul Le Guernic, de la thèse de Doctorat en Informatique (IFSIC, Université de Rennes 1) de Fernando Jiménez-Fraustro, depuis la rentrée 1997, sur le sujet *Environnement pour la conception d'automatismes complexes*.

Stage de DEA Informatique (IFSIC, Université de Rennes 1) de Fernando Jiménez-Fraustro : *Traduction de blocs de diagrammes électroniques en format synchrone* (1997)

collaborations :

- en lien avec la collaboration avec la DER (Direction des Études et Recherches) d'Électricité de France (EDF) évoquée en Section 2.2.1, on travaille à la modélisation de langages de programmation d'automatismes en SIGNAL.
- Groupe de travail «téléproductique Bretagne»¹⁵.
Il rassemble des partenaires intéressés dans l'Ouest, à Brest (Université de Bretagne Occidentale, ENIB, IUT, ENSIETA) et à Rennes (Irisa, Insa, Supelec, ENS-Cachan à Bruz). Des intérêts communs avec l'UBO et Supelec se sont dégagés, concernant la modélisation de contrôleurs de cellule de production en SIGNAL, Grafcet ou STATECHARTS, ainsi que les architectures d'automates programmables.
- Groupe GRAFCET du club EEA¹⁶.
Il rassemble des partenaires industriels et académiques de France et de Belgique autour de la problématique de la définition, de l'évolution et de la diffusion du GRAFCET.

2.4 Modélisation du séquençement de tâches flot de données

2.4.1 Bilan

Ce chapitre fait la synthèse de mes contributions à la problématique de définition et modélisation formelle de structures mixtes de langage de programmation des systèmes de contrôle/commande, avec le support d'outils d'assistance. Il présente *GTi*, extension de SIGNAL avec des structures d'intervalle de temps et de tâches suspensibles ou interruptibles. Elles encapsulent les processus flot de données dans un contrôle d'activité, pour rendre compte de la mixité contrôle/commande, et permettre l'expression aisée de hiérarchies de systèmes de transition concurrents. Leur définition en termes de SIGNAL les intègre à l'environnement de programmation ; elle fait intervenir la gestion de la mémorisation d'état, du contrôle de l'activité, et de la réinitialisation de l'état.

Ces structures simples se retrouvent dans la modélisation synchrone de langages de contrôle plus élaborés et répandus chez les utilisateurs que sont STATEMATE et les langages de la norme IEC 1131. Il s'agit de donner pour ces langages une modélisation rigoureuse, et qui donne accès à des fonctionnalités d'analyse, de vérification et de mise en œuvre automatisées, en particulier ici celles de la technologie synchrone, au travers de SIGNAL et du format DC+. On s'intéresse aux langages de STATEMATE : STATECHARTS (automates à états, hiérarchiques et concurrents) et ACTIVITYCHARTS (activités décrites sous forme de bloc-diagrammes), ainsi que le langage des actions (textuel impératif). Leur encodage en SIGNAL fait une ré-utilisation des concepts de base de *GTi* pour la hiérarchisation de la gestion du contrôle.

Un travail en cours étend la problématique par une variété de langages correspondant à une culture et des domaines d'application différents (machinisme industriel), où se retrouvent les mêmes notions de combinaison contrôle/commande. Il s'agit des langages de la norme IEC 1131 de conception de systèmes à base d'automates programmables, parmi lesquels on trouve une variante du GRAFCET, et un langage impératif textuel. Leur modélisation en SIGNAL se fait en ré-utilisant les résultats précédents : ceux de *GTi* pour les mêmes aspects et le codage des actions de STATEMATE généralisé à la représentation de granularités de temps imbriquées.

2.4.2 Perspectives

Des perspectives spécifiques aux différents points présentées dans les sections précédentes, on peut en dégager de plus générales, concernant

- un modèle formel spécifique de la notion de tâche (éventuellement fondé sur un calcul sur les intervalles de temps ou d'activité),

15. Site web du groupe : <http://doelan-gw.univ-brest.fr:8080/teleproductique/teleprod.html>

16. Site web du groupe GRAFCET : <http://www.lurpa.ens-cachan.fr/grafcet/>

- la structuration des entités à offrir aux utilisateurs de langages de contrôle/commande (qui soient plus élaborées que les primitives, sans toutefois obscurcir la compréhension de leur fonctionnement par leur complexité),
- et aussi les liens entre les niveaux de granularité d’instantanés logiques (vis-à-vis de l’environnement) et la mise en œuvre de programmes synchrones par raffinements successifs du séquençage des actions d’un instant de réaction. Cela pourrait faire intervenir la modélisation de l’éventuelle préemption ou suspension de parties du code à l’intérieur de l’exécution d’une réaction.

2.4.3 Applications

Un autre aspect de la problématique était de se confronter à des applications, et de traiter des systèmes de contrôle/commande présentant les caractéristiques de mixité de comportement continu (échantillonné) et discret, de complexité de contrôle (préemption, concurrence et hiérarchie), et de besoin d’assistance à la conception, notamment en validation. C’est l’objet du Chapitre 3 suivant de faire la synthèse de mes travaux dans des expérimentations concernant une cellule de production, un système de vision active en robotique, un contrôleur de disjoncteur dans un transformateur électrique, et l’animation comportementale dans une plateforme de simulation.

Chapitre 3

Applications à des systèmes de contrôle/commande

Ce chapitre décrit les applications que j'ai faites des concepts décrits aux chapitres précédents à la programmation de systèmes de contrôle/commande. Les domaines d'applications de *SIGNAL* et *SIGNALGTi* comprennent particulièrement l'interaction du traitement continu échantillonné avec le contrôle discret au sens de commutation entre modes continus. Ces applications illustrent donc la spécification des systèmes en utilisant *GTi*, mais aussi l'utilisation de l'environnement *SIGNAL*, pour la génération de code exécutable, et pour la vérification. J'étudie particulièrement dans ces expérimentations le caractère intégré de l'alliance de processus et d'intervalle de temps pour la spécification de système hybrides.

Les applications concernent :

- une cellule productive, définie comme banc d'essai de méthodes formelles [69], traitée avec *SIGNAL* ;
- la vision active en robotique, où apparaît l'aspect hybride continu/discret traité par *GTi* ;
- le contrôle d'un automate de disjoncteur dans un transformateur électrique, illustrant les hiérarchies de préemption ;
- l'animation et la simulation en image de synthèse, où on trouve une combinaison différente de flots de données et d'automates.

Leurs traits communs peuvent se trouver dans l'aspect méthodologique associé à l'approche synchrone, et notamment l'utilisation dans la programmation :

- de l'hypothèse synchrone pour la structuration, c'est-à-dire notamment le caractère de la composition synchrone permettant d'utiliser un parallélisme de spécification indépendamment de la mise en œuvre,
- des structures de préemption hiérarchique,
- d'un alliage multi-formalisme pour la combinaison du contrôle et de la commande.

3.1 Cellule de productive du FZI

Cette expérimentation de *SIGNAL* s'est faite dans le cadre d'une étude de cas proposée par le FZI¹ de Karlsruhe, et traitée dans une vingtaine d'autres méthodes formelles, dont plusieurs langages synchrones [69]. Elle a été menée avec Tocheou Pascal, Amagbegnon, Paul Le Guernic et Hervé Marchand. Il s'est agi de la spécification en *SIGNAL* du contrôleur d'une cellule robotique, de la vérification de ses propriétés de sécurité, et de sa mise en œuvre sous forme de contrôleur d'un simulateur graphique fourni par le FZI [4, 5].

Cette application est représentative du contrôle dans les applications de productive en général, en ce qu'elle décrit un ensemble de machines inspiré d'un site industriel réel, et comportant des équipements courants, tant dans les actionneurs que dans les capteurs. Le simulateur graphique proposé par le FZI est accessible comme le serait la cellule, à un degré d'interfaçage où on peut lire les valeurs mesurées par les capteurs, et écrire des ordres de contrôle

1. *Forschungszentrum Informatik* : centre de recherche en informatique de l'Université de Karlsruhe, Allemagne ; sites web : général du laboratoire : <http://www.fzi.de>, sur l'étude de cas de la cellule de production dans sa première version (traitée ici) : http://www.fzi.de/prost/projects/production_cell/ProductionCell.html, et sur sa nouvelle version (concernant aussi la tolérance aux fautes) : <http://www.fzi.de/prost/projects/korsys/korsys.html>.

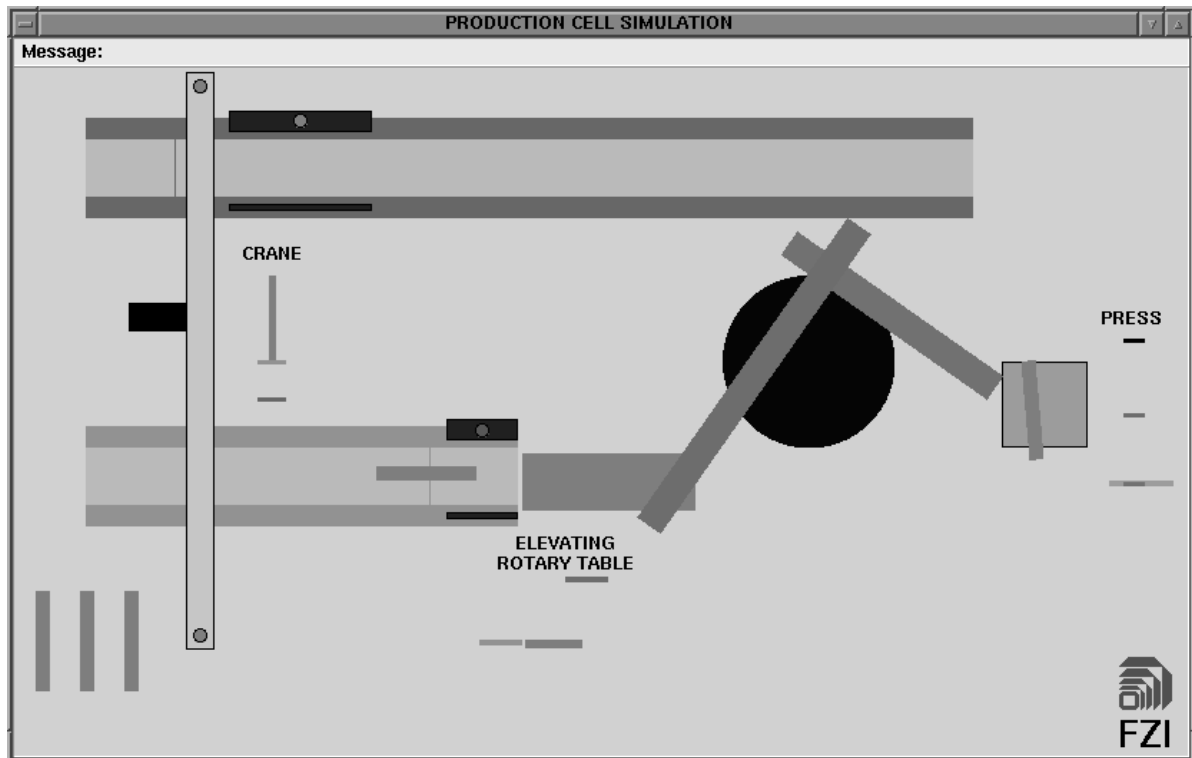


FIG. 3.1 – Cellule de production : le modèle du FZI.

des actionneurs. Ce qui est requis est de produire un contrôleur qui mette en œuvre le contrôle de ces éléments selon la fonctionnalité de la cellule, tout en établissant certaines propriétés de sécurité.

Ce cadre expérimental permet donc de mettre en pratique les différentes fonctionnalités d'un environnement comme SIGNAL :

- spécification du contrôleur,
- mise en œuvre et génération de code exécutable,
- vérification des propriétés.

3.1.1 La cellule de productique du FZI

Configuration

Le cellule robotique définie par le FZI est composée d'un ensemble de machines à contrôler, illustrées en Figure 3.1 qui est une copie d'écran du simulateur graphique fourni par le FZI :

- deux convoyeurs (tapis roulants) représentés horizontalement ; chacun est équipé d'un capteur de présence d'objet transporté, en l'occurrence une cellule photo-électrique ;
- une table rotative et élévatrice, qui doit transporter les objets arrivant du convoyeur vers un des bras manipulateurs, situé plus haut ; une représentation schématique de sa position verticale la montre en position basse ;
- une presse, elle aussi dotée dans l'interface graphique d'une vue de côté, indiquant sa position verticale par rapport à trois niveaux : haut (presse fermée), milieu, bas ;
- un portique avec grue, doté de deux capteurs photo-électriques en extrémités de course ; une vue de côté indique aussi la position verticale de la grue ;
- deux bras extensibles, sur une base rotative ; chacun est muni à son extrémité d'un électro-aimant, qui est le moyen de saisir les pièces métalliques véhiculées dans la cellule. L'un des bras est à la hauteur de la position basse de la presse, et l'autre à la hauteur de sa position moyenne. C'est en jouant sur leur longueur d'extension et l'angle de la base rotative qu'on décrit l'espace dans lequel ils manœuvrent.

Cet ensemble est accessible par des ordres (démarrage de mouvement, arrêt), et on peut consulter des capteurs (de position, comme la longueur d'extension de bras, l'angle de rotation, la position de la presse, ou de présence d'objet sur les convoyeurs)

Scénario

La fonctionnalité à remplir est de faire en sorte que des pièces métalliques arrivant sur un convoyeur (représenté en bas dans la Figure 3.1) soient déversées sur la table une par une. Celle-ci élève chacune en tournant pour qu'elle puisse être saisie par un des bras extensibles, dont la base doit alors tourner pour l'emmener vers la presse. La pièce y est posée, puis pressée, puis emportée par l'autre bras extensible vers l'autre convoyeur. Ce dernier amène la pièce vers le portique, dont la grue la saisit. De manière à mettre en place un comportement cyclique, à des fins de complétion de l'expérience, la grue transporte la pièce vers l'autre convoyeur, où elle repart vers la table et la presse.

Pour obtenir ce scénario, on doit contrôler les différents éléments de la cellule de manière à les faire coopérer ; par exemple on doit synchroniser le lâchage de pièce par l'électro-aimant du bras extensible avec sa position à l'intérieur de la presse, quand celle-ci est en position basse. On doit aussi gérer des risques de conflit ou de compétition vis-à-vis par exemple de l'espace ; par exemple, si la presse est en position moyenne, il y a risque de collision avec le bras qui est à la hauteur de la position basse, selon l'extension de celui-ci et l'angle de la base rotative.

Propriétés requises

Comme le suggèrent les spécifications ci-dessus, on est face à un système aux comportements complexes, parmi lesquels on veut s'assurer qu'on interdit ceux qui mèneraient à des accidents. L'étude de cas définit donc des propriétés qui doivent pouvoir être vérifiées formellement sur le contrôleur.

Sûreté. Il y a des propriétés de sûreté qui sont par exemple, concernant la table, que celle-ci ne doit pas être soumise à rotation plus loin dans le sens des aiguilles d'un montre quand elle est dans la position de livrer une pièce au bras ; réciproquement, elle ne doit pas être tournée dans l'autre sens quand elle est en position de recevoir des pièces du convoyeur. De même, on ne doit pas pouvoir la faire descendre quand elle est en position basse, ou monter au-delà de la position haute. Ces propriétés concernent la sauvegarde du matériel ; des restrictions similaires s'appliquent aux mouvements du portique et de la grue. Dans l'interaction entre le bras et la presse, il y a des propriétés de sûreté concernant l'accès à l'espace commun pour éviter les collisions.

Vivacité. Une propriété supplémentaire requise est la vivacité, au sens de l'absence de blocage qui mettrait le contrôle dans un sous-ensemble d'états d'où il ne pourrait plus sortir. Ici, on veut que toute pièce introduite dans le système *via* le convoyeur soit finalement posée par la grue du portique sur le convoyeur.

3.1.2 Spécification du contrôleur en SIGNAL

La spécification est structurée de façon modulaire, en utilisant deux principes de décomposition : l'un suivant l'architecture de la cellule de production, l'autre séparant clairement le modèle du système et son contrôleur. Ce dernier point est une originalité de l'approche, comparée aux approches impératives : le langage déclaratif est utilisé pour spécifier, sous forme d'équations sur des signaux, les comportements possibles d'un système, et un contrôleur les contraignant. De cette façon, on peut construire des hiérarchies de systèmes contrôlés imbriqués, comme l'illustre la Figure 3.2 : dans le cas de la cellule de production, le comportement ordonnancé est une instance contrôlée du comportement sûr, qui est lui-même une instance contrôlée du comportement naturel.

Modélisation de la cellule

Comportements naturels. Par exemple, le contrôleur de la presse a des entrées capteurs (dans la Figure 3.2 : `Ecapteur`) donnant la position de sa partie mobile (un signal Booléen pour chacune : haute (fermée), milieu, basse (ouverte)), et une sortie de contrôle (`Sact`) qui peut être l'ordre de monter, de descendre, ou d'arrêter. Le contrôleur des mouvements naturels de la presse est composé de règles comme : émettre un ordre d'arrêt quand la presse est en mouvement vers le haut et que le capteur de position haute est vrai.

Comportements sûrs. Les interactions entre les machines dans la cellule sont variées ; on a déjà mentionné les synchronisations à établir entre les bras et la presse, tant pour venir la charger avant qu'elle ne se ferme, que pour la décharger après (une collision peut avoir lieu si le bras s'étend quand sa position angulaire lui fait pointer vers la presse, et si celle-ci n'est pas ouverte). Pour réaliser la synchronisation (à des fins de coopération comme de gestion

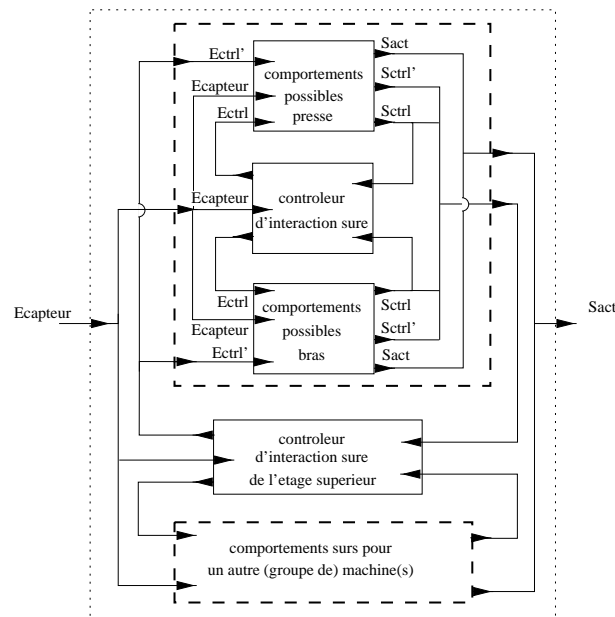


FIG. 3.2 – Cellule de production : spécification étagée, avec les comportements possibles, les comportements sûrs, et le scénario complet.

de la compétition) avec les autres éléments de la cellule, le contrôleur des comportements sûrs de la presse est doté d'entrées et de sorties supplémentaires pour la communication d'information de contrôle (dans la Figure 3.2 : *Ectrl* et *Sctrl*). Par exemple, les règles supplémentaires dans le cas de la presse définissent, en position ouverte ou basse, l'émission d'un événement de disponibilité, et l'attente d'un signal d'autorisation avant d'entamer un mouvement de fermeture, vers le haut.

Quand on groupe deux ou plus éléments, ces signaux de contrôle sont utilisés par un contrôleur additionnel (le contrôleur d'interaction sûre dans la Figure 3.2) qui se concentre sur les interactions, et définit les comportements sûrs de l'ensemble. Son rôle est de faire en sorte que les éléments se synchronisent de façon à gérer les compétitions possibles, et à coordonner leurs actions. Par exemple, quand un bras maintient avec son électro-aimant une pièce destinée à être pressée, alors son contrôleur doit attendre l'événement de disponibilité signalant l'ouverture de la presse et sa capacité à accepter cette pièce. Par ailleurs la presse doit attendre que le bras ait fait le mouvement nécessaire et l'action de lâcher la pièce au bon endroit, avant de pouvoir faire un mouvement vers le haut pour se fermer et exécuter le pressage. Ce type de contrôle d'interaction peut faire intervenir des entrées capteurs, lui aussi, et être dynamique, au sens où il présente un état interne.

De façon à être en mesure de se synchroniser avec encore un autre étage de contrôle, il est lui-même doté d'entrées et de sorties de synchronisation (dans la Figure 3.2 : *Ectrl'* et *Sctrl'*). Dans la Figure, ce système est délimité par la boîte en trait discontinu.

Le rôle de la composition est ici bien celui de la composition de systèmes d'équations : l'ensemble des comportements possibles de chaque élément de la cellule se trouve restreint par l'addition de contraintes supplémentaires par rapport aux traces des signaux additionnels de contrôle. Celles-ci sont définies par les contrôleurs d'interaction sûre, qui en définissent le sous-ensemble admis (*a priori* ils sont strictement inclus dans l'ensemble de tous les comportements, à moins d'avoir affaire à un contrôleur vide). Ainsi, la composition des ensembles de comportements possibles pour chaque élément avec les contrôleurs d'interactions sûres définit un ensemble de comportements qui en est l'intersection.

Spécification du scénario

Ce schéma peut être répété hiérarchiquement à plusieurs niveaux, comme illustré en Figure 3.2, où on compose la boîte en trait discontinu du bas avec le contrôleur d'interaction sûre de l'étage supérieur. On construit ainsi des sous-ensembles, comme l'illustre la Figure 3.3. On y voit les contrôleurs respectifs de la partie de transport de la cellule (portique et grue, table, convoyeurs) composée avec le sous-ensemble gérant le robot (la base rotative et les deux bras extensibles) et la presse. Ils échangent des signaux synthétisant leurs interactions : ils signalent par exemple le fait

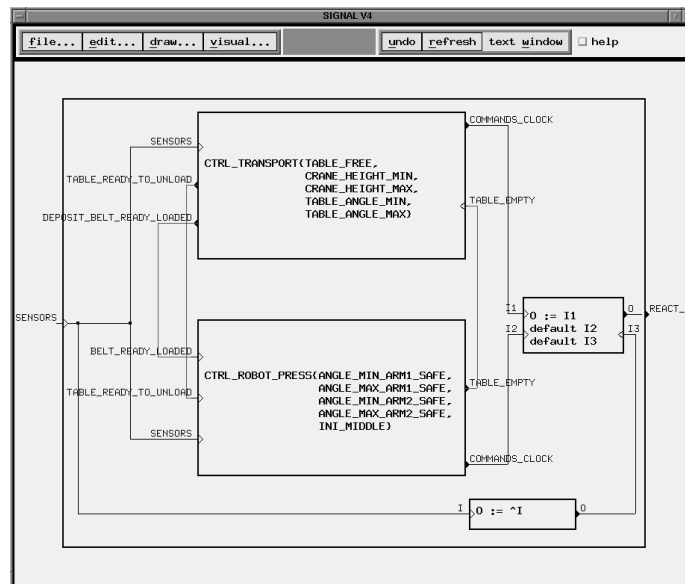


FIG. 3.3 – Cellule de production : spécification du contrôleur.

que la table est prête à livrer une pièce nouvelle, ou le fait qu'elle est vide. Ces contrôleurs sont dotés de paramètres quantifiant leurs caractéristiques (angles maximaux, longueurs de courses, états initiaux d'ouverture).

L'obtention du contrôleur final global de la cellule de production, regroupant tous les éléments, consiste à définir un niveau hiérarchique où toutes les interactions de contrôle sont gérées sans ressortir à des entrées de contrôle externes. En d'autres termes, les interactions sont résolues complètement de façon interne.

Dans ce cas, le comportement global est complètement déterminé par ses entrées capteur (boîte pointillée dans la Figure 3.2), et en ce sens constitue le scénario complet de l'application. Ceci complète une méthodologie de spécification progressive, étagée en couches, et en même temps décomposée selon les éléments qui constituent la cellule, au moyen de la composition synchrone. Une mise en œuvre distribuée du contrôleur pourrait alors être décomposée et structurée d'une façon toute différente, par exemple suivant non pas la topologie des éléments contrôlés, mais suivant la localisation des capteurs, où un même capteur peut être utilisé par des parties de plusieurs contrôleurs.

3.1.3 Spécification et vérification des propriétés

Le modèle de la cellule de production est fait en termes d'événements et de booléens. S'abstraire ainsi de la nature numérique de certains capteurs permet l'analyse formelle de propriétés du système. Le caractère équationnel de Signal mène naturellement à l'utilisation de méthodes fondées sur les systèmes polynomiaux dynamiques d'équations sur $\mathbb{Z}/3\mathbb{Z}$ pour la preuve formelle de la satisfaction des propriétés nécessaires à l'application. Comme on l'a mentionné en Section 2.2.1, c'est sur ces bases que repose la technique et la méthode de vérification de propriétés dynamiques de programmes SIGNAL.

On l'introduit ici sur l'exemple d'une propriété concernant la presse : la presse ne doit pas être descendue si elle est en position basse, ni montée si elle est en position haute.

Spécification

Modèle de l'environnement. La vérification de certaines propriétés de contrôleurs doit tenir compte d'hypothèses qui ont été faites sur l'environnement dans lequel ils agissent. Pour cela, il faut les rendre explicite, et construire un modèle global qui en rende compte. Ici encore la composition synchrone intervient : en effet on compose le contrôleur de la presse avec un processus décrivant une abstraction de l'environnement. Dans le cas de la presse, il s'agit de décrire la relation entre les trois capteurs logiques décrivant l'état de la presse :

- les signaux Booléens de position haute, milieu, et basse ne peuvent être à vrai qu'exclusivement,
- il y a toujours une occurrence de la position du milieu à vrai entre celles de la position haute et de la position basse.

Observateur de la propriété. Concernant la spécification de la propriété elle-même, on recourt ici à la technique des observateurs, qui consiste à composer le contrôleur et l'abstraction de son environnement, avec un processus qui calcule une sortie Booléenne qui est vraie quand la propriété est contredite. Ce calcul peut faire intervenir lui-même un état interne, et consister en la reconnaissance de séquences d'événements indésirables. Dans le cas de la presse, la propriété qui nous intéresse est contredite si la presse est toujours en mouvement vers le bas après qu'on soit passé par la position basse (et avant qu'on ne soit retourné vers le haut).

Un tel processus peut faire intervenir des éléments plus génériques, réutilisables ; par exemple ici, on peut se définir un processus qui reconnaît l'occurrence d'un événement entre deux autres.

Vérification

Dans le cadre de l'utilisation de SIGALI, la propriété à vérifier se formule alors comme : l'ensemble des états où le signal d'erreur est vrai n'est pas accessible depuis l'ensemble des états initiaux. Le calcul de l'accessibilité d'un sous-ensemble d'états à partir d'un autre fait partie des fonctionnalités de base de SIGALI.

Dans le cas de la presse étudiée ici, il a rendu comme résultat que les situations indésirables n'étaient pas accessibles ; ainsi est vérifiée la propriété que le contrôleur ne permet aucune trace menant au mouvement en butée de la presse.

Activités liées à l'application à la cellule de production

publications : chapitre de livre [4], rapport de recherche [5].

outil : démonstrateur : contrôleur connecté au simulateur graphique fourni par le FZI.

3.2 Applications de GTi

3.2.1 Vision active en robotique

Cette application s'est faite en collaboration avec le projet TEMIS (sa partie devenue VISTA²) de l'IRISA/INRIA Rennes (François Chaumette et Eric Marchand), avec le soutien du projet inter-PRC VIA (*Vision Intentionnelle et Action*) (en 1992-95), et avec Hervé Marchand. La programmation au niveau tâche en robotique fait intervenir l'enchaînement (sur des intervalles de temps) des fonctions de contrôle à comportement continu (tâches flot de données) de façon réactive à l'environnement. On utilise SIGNALGTi pour la programmation de stratégies perceptives d'un environnement statique dans un contexte de vision active. Cette expérimentation illustre le caractère intégré de l'environnement, de la spécification en SIGNALGTi à la mise en œuvre séquentielle et la vérification de propriétés du comportement du système. Elle s'inscrit dans une réflexion plus générale sur l'adéquation d'un environnement comme SIGNAL en tant qu'environnement intégré de conception pour le contrôle d'applications robotiques.

Application de reconstruction tridimensionnelle

Fonctionnalité. La tâche du système de vision robotique est de créer un modèle géométrique 3D de son environnement à partir de la séquence d'images acquises par une caméra en temps réel [75]. Une méthode générale a été établie permettant de localiser toute primitive géométrique paramétrable. Elle est basée sur l'utilisation de la matrice d'interaction qui permet de relier les informations visuelles sur la primitive considérée au mouvement de la caméra. Lorsque les mouvements sont quelconques les erreurs de localisation sont très importantes. Il a été cependant montré qu'un déplacement adéquat de la caméra par rapport à la primitive à localiser permettra de minimiser les erreurs sur l'estimation des paramètres de la primitive. De tels mouvements seront réalisés en utilisant l'approche commande référencée-vision ou asservissement visuel. Cependant, cette approche ne permet de reconstruire qu'une seule primitive à la fois ; de plus une connaissance *a priori* sur la nature de la primitive est nécessaire afin de générer les mouvements optimaux de la caméra. Il faut donc s'abstraire de ces contraintes en définissant des stratégies de perception de la scène afin de permettre une représentation 3D précise et complète de la zone à reconstruire. Le schéma de base proposé consiste à focaliser successivement la caméra sur les différents objets de la scène et à les reconstruire. La stratégie de reconstruction de la scène 3D est donc constituée de phases d'asservissement visuel durant lesquelles la primitive 3D sur laquelle la caméra est focalisée sera reconstruite, et de phases de recherche d'informations permettant de sélectionner les futures primitives à reconstruire.

2. Site web : <http://www.irisa.fr/vista/>

Programmation. Il est donc apparu clairement que l'une des difficultés de cette étude résidait, entre autres, dans la façon de gérer la dualité *continu/événementiel* dans la gestion du mouvement de la caméra et des stratégies de perception. Dans un tel système, la réactivité intervient donc à deux niveaux distincts : dans le calcul de la commande en boucle fermée où le système doit réagir à chaque instant en fonction des informations transmises par le capteur, et au niveau du contrôle de tâche où le système doit réagir à des événements extérieurs par des changements d'état. Dans cette optique, SIGNAL apporte une méthodologie de programmation unifiée permettant d'intégrer l'aspect "continu" des algorithmes de commande et d'estimation avec l'aspect "événementiel" des stratégies de perception au sein d'un langage temps réel de haut niveau.

Les méthodes de programmation classiques ne sont pas bien adaptées à la spécification et à l'implémentation de tels algorithmes. Un langage classique impératif requiert une prise en compte explicite des aspects bas niveau de la mise en œuvre (tel l'ordonnancement séquentiel des calculs) ainsi que des problèmes temporels pour lesquels ils ne fournissent pas de support ou de modèle bien établis. Les intérêts de SIGNAL pour cette application sont, d'une part, son caractère flot de données qui est parfaitement adapté aux tâches d'asservissement visuel et aux algorithmes de reconstruction 3D. Les opérateurs de SIGNAL permettent une programmation aisée des algorithmes utilisant des valeurs mesurées à des instants précédents (dans notre cas, mesure du déplacement de la caméra, mesure du déplacement des primitives entre plusieurs images successives, filtres moyennant, etc). D'autre part, son compilateur recalcule à chaque modification du programme un ordonnancement des calculs dans l'instant logique ou cycle, qui respecte les contraintes de dépendance de données. De plus, dans ce contexte où on est effectivement dans le cas d'une mixité séquençement/flot de données, les structures de *GTi* permettent une gestion simple du contrôle événementiel du lancement et de l'arrêt des tâches.

Pour ce qui est de la programmation d'applications robotiques en général, ces qualités se combinent aux fonctionnalités de l'environnement SIGNAL, pour offrir un ensemble très intégré, où tout repose sur un modèle commun, limitant les problèmes d'assemblage de composants logiciels trop divers, et permettant une vérification plus globale. D'autres approches, moins intégrées, existent comme CONTROL SHELL [103], KHEOPS [1], ORCCAD [24]; leur avantage est d'être plus dédiés à la programmation robotique, en présentant des structures de base plus proches des notions de tâche de commande ou d'événement d'exception, ou en traitant les problèmes décisionnels comme la planification de mission.

Algorithmes d'estimation et de contrôle, en SIGNAL

L'asservissement visuel : algorithme flot de données. L'asservissement visuel consiste à utiliser les informations fournies par une caméra mobile commandable afin de réaliser des tâches élémentaires (positionnement, suivi d'objet mobile, suivi de trajectoire) en contrôlant la situation ou le mouvement de la caméra par rapport à son environnement. L'approche consiste à spécifier le problème en termes de régulation dans l'image ce qui permet de compenser les imprécisions des modèles de la caméra et de l'effecteur.

La loi de commande en boucle fermée sur les informations capteurs utilisées en asservissement visuel consiste en la régulation d'une fonction de tâche qui peut, de manière schématique, s'écrire $c = f(s)$ où c est la vitesse de l'effecteur exprimée en fonction des signaux capteurs s . Le contrôle de l'effecteur est une fonction continue f plus ou moins complexe. L'implémentation d'une tel loi de commande est réalisée en «discrétisant» le flot d'information s en provenance de la caméra (cadence vidéo) en un flot de valeurs s_t , qui est ensuite utilisé pour calculer un flot de commande $c_t : \forall t, c_t = f(s_t)$. Ce type de calcul numérique et flot de donnée est le champ d'application privilégié des langages flot de données et de SIGNAL en particulier. De plus, comme le montre l'indice t dans l'équation schématique présentée, la présence simultanée des signaux mis en jeux est parfaitement gérée par l'hypothèse synchrone.

Une description modulaire du programme SIGNAL mettant en œuvre l'asservissement visuel est présentée sur la Figure 3.4. Au plus haut niveau de description on retrouve les trois processus principaux synchrones :

- le processus CAMERA_OUTPUT qui produit un flot d'information capteur à la cadence vidéo (horloge du système) ;
- ces informations sont reçues par le processus de calcul de la commande, qui est calculée en utilisant l'approche fonction de tâche (dans la boîte en trait discontinu) ;
- la commande de la caméra ainsi calculée est transmise à la commande numérique du robot par le processus ROBOT_CONTROL.

Le processus de calcul de la commande est lui même décomposé en sous processus permettant de calculer les erreurs entre les signaux capteurs courants et les signaux capteurs désirés, la valeur courante de la matrice d'interaction L , la tâche primaire (convergence vers un motif à atteindre dans l'image) et la tâche secondaire (suivi de trajectoire).

La reconstruction 3D : processus dynamique et parallélisme. L'objectif de la reconstruction est de remonter à la structure 3D de l'objet observé en utilisant les informations extraites d'une séquence d'image. Le mouvement

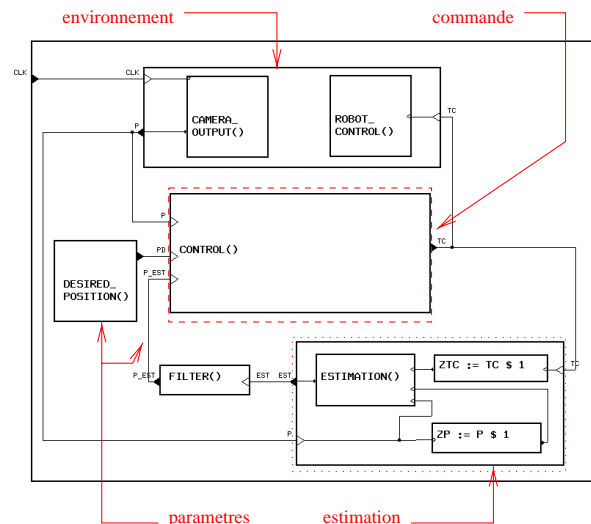


FIG. 3.5 – Vision robotique : processus SIGNAL d'estimation en parallèle de la commande.

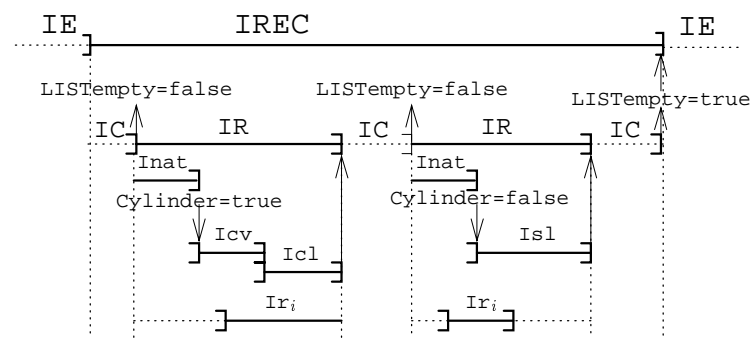


FIG. 3.6 – Vision robotique : une trace possible du séquençement de tâche.

SIGNALGTi autorise par exemple la combinaison de comportements (parallélisme entre tâches), la préemption ainsi que le séquençement de tâches. Il permet de spécifier la terminaison des tâches flot de données. La Figure 3.6 illustre ceci d'une trace possible, où on alterne, dans les intervalles IE et IREC, entre respectivement l'exploration (pour repérer des objets à reconnaître) et la reconnaissance d'un ensemble d'objets. Ce dernier intervalle est décomposé en IC (choix de l'objet à reconnaître) et IR (reconnaissance de cet objet). Si on n'en trouve pas (la liste est vide) on interrompt la tâche en terminant IREC. La reconnaissance d'un objet elle-même commence par la détermination de la nature de l'objet : cylindre ou segment. Dans le premier cas on enchaîne sur l'estimation de son axe, puis de sa longueur ; dans l'autre, on estime la longueur du segment. En parallèle, on peut lancer des tâches de reconnaissance secondaire, sur l'apparition d'objets traversant le champ de vision de la caméra, sachant qu'elles peuvent être interrompues par la tâche de reconnaissance principale.

La spécification de l'application en SIGNALGTi est donnée en Figure 3.7, d'une manière simplifiée en gardant uniquement les aspects essentiels. Par exemple, l'intervalle Icv d'estimation de l'axe du cylindre est ouvert par l'événement de reconnaissance d'objet cylindrique, et fermé par le fait que la précision atteinte $prec$ passe en dessous d'une valeur donnée ε_{cv} .

Un processus flot de données, comme nos tâche de vision, porte en lui la spécification complète de son comportement mais pas celle de sa terminaison. Cet aspect doit alors être défini séparément. Une façon de décider de la terminaison d'une tâche est de spécifier un critère dépendant des données capteurs ou des calculs effectués par le processus lui-même. L'évaluation de ce critère doit se faire à chaque instant, de ce fait, cette évaluation devient un autre processus flot de données. L'instant où la condition est vérifiée peut être marqué par un événement discret, qui, causant la terminaison d'une tâche, peut aussi être la cause d'une transition vers une autre tâche au même niveau ou à un niveau plus élevé dans l'automate hiérarchique. Dans cette optique ce type d'événement peut être utilisé pour

```

Primitive_estimation:
(| Optimal_estimation
 |(| Coarse_estimation1 | ... | Coarse_estimationn |)
 | Occlusion_avoidance | Joint_limits_avoidance |)

Coarse_estimationi:
(| Iri := ] New_Segment, Segment_Lost ] init outside
 | Coarse_estimation each Iri |)

Optimal_estimation:
(| Inat := ] close Isl default close Icl, Cyl] init inside
 | Nature each Inat
 | Icv := ] when Cyl, when (| prec | <  $\varepsilon_{cv}$ ) ] init outside
 | Cylinder_vertex each Icv
 | Icl := ] close Icv, when (| prec | <  $\varepsilon_{cl}$ ) ] init outside
 | Cylinder_length each Icl
 | Isl := ] when not Cyl, when (| prec | <  $\varepsilon_{sl}$ ) ] init outside
 | Segment_length each Isl |)

```

FIG. 3.7 – Vision robotique : spécification du séquençement de tâche.

marquer la fin de l'exécution interne d'une tâche.

La Figure 3.8 montre l'environnement d'exécution de l'application construit avec le langage SIGNAL. On y voit en haut à gauche une représentation en termes d'automate hiérarchique parallèle de la stratégie, au milieu la vue courante de la caméra, à droite une vue du modèle géométrique reconstruit. Les courbes sont les différentes grandeurs reconstruites, et en bas on voit le chronogramme des intervalles.

Vérification

Nous avons appliqué la méthode et les outils de vérification associés à SIGNAL (et avec lesquels SIGNALGTi est compatible) dans le cadre de ce système de vision robotique. Nous avons utilisé SIGNALI pour la vérification de diverses propriétés, comme par exemple :

- à l'intérieur d'une tâche de reconstruction, les processus d'estimation et d'évitement de butée articulaire ne peuvent pas être actifs en même temps. Cette propriété doit être vérifiée, faute de quoi les degrés de liberté du robot subiraient des commandes contradictoires. Il faut pour cela vérifier que les deux tâches (délimitées par leur intervalle de temps) ne peuvent être actives en même temps, c'est-à-dire, en termes d'intervalles, que l'état où I_joint et I_EST_limbs ont tous les deux la valeur vrai, ne soit pas accessible depuis les états initiaux du programme.
- un processus d'évitement de butée articulaire spécifique et un processus d'évitement d'occlusion ne sont actifs que quand une estimation l'est. L'estimation étant gérée par deux tâches différentes, à des niveaux différents de l'automate hiérarchique, cette propriété est moins locale, et par là même moins facile à vérifier.

Activités liées à l'application de vision robotique

publications : journaux (différents aspects : vision active [77], programmation en SIGNAL et GTi [98], environnement de programmation et vérification [78]), conférences (dont [76]), rapports de recherche.

outil : contrôleur et simulateur graphique développé par Florent Martinez (en stage de DEA) et Eric Marchand (en thèse)

encadrement : Stage d'été de Maîtrise Informatique, mini-projet 5^e année Informatique INSA de Rennes (co-encadré avec Éric Marchand, TEMIS).

Sur le thème de la programmation de systèmes robotiques, participation à des jurys de thèse de Doctorat en Informatique (IFSIC, Université de Rennes 1) : Erwan Le Rest : *PILOT : un langage pour la télérobotique* (UBO, Brest, Juin 1996) ; Jean-Luc Fleureau : *Vers une méthodologie de programmation en télérobotique : comparaison des approches Pilot et Grafcet* (UBO, Brest, Juillet 1998).

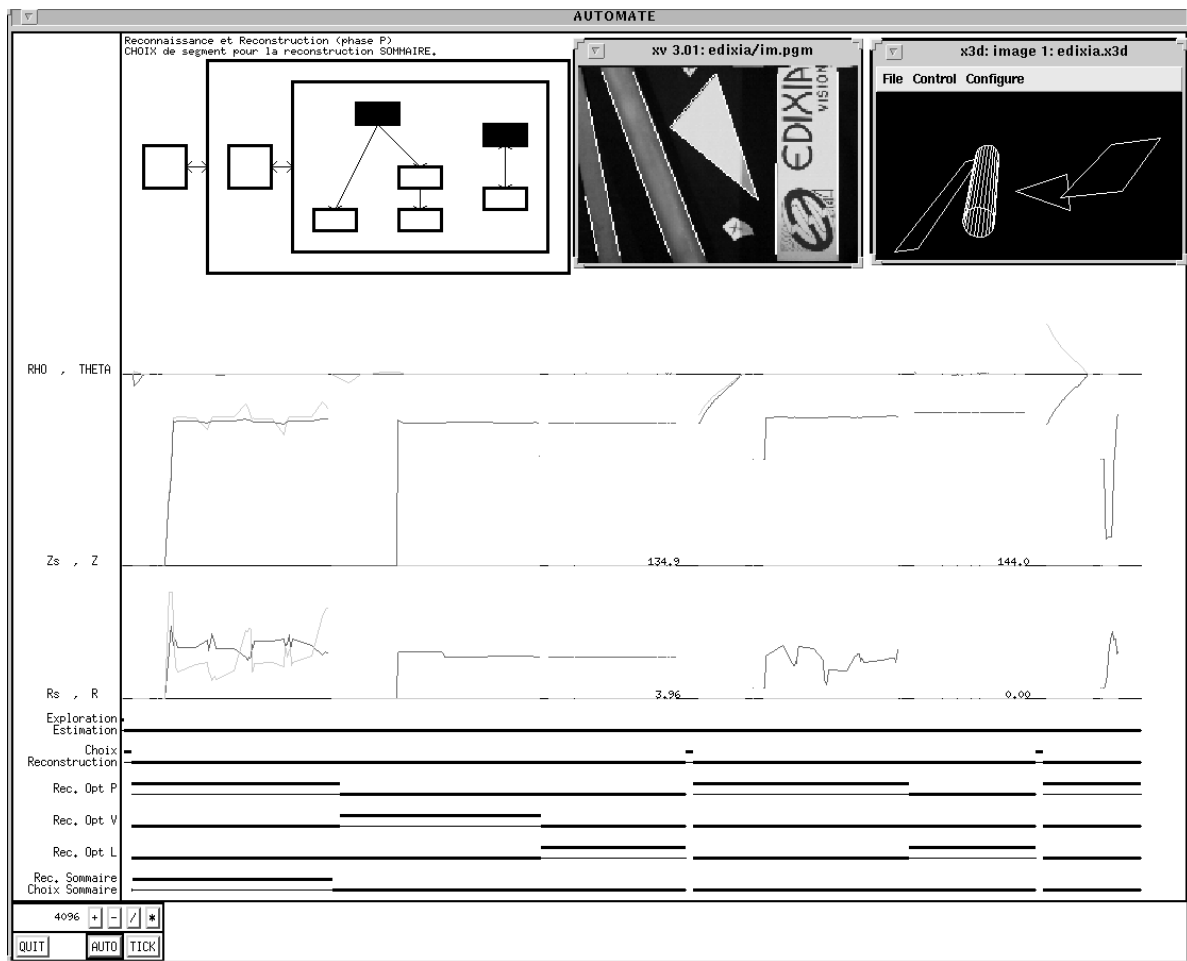


FIG. 3.8 – Vision robotique : l'environnement de reconstruction de scène 3D.

collaborations : avec le projet TEMIS, maintenant VISTA : François Chaumette et Éric Marchand (en stage de DEA, puis thèse) ; participation au Projet inter-PRC VIA (*Vision Intentionnelle et Action*).

3.2.2 Contrôle de transformateur

Cette étude s'est déroulée dans le cadre d'une coopération avec Mazen Samaan, du groupe Acsar puis du département Contrôle Commande Centrales de la DER (direction des études et recherches) d'EDF, et avec Hervé Marchand. La spécification du comportement de gestion de défauts électriques d'une cellule de transformateur a été réalisée en SIGNALGTi à partir de descriptions fournies par Électricité de France. Celles-ci étaient faites en termes d'automates parallèles communicants relativement complexes. Le comportement faisait apparaître un certain nombre de modes, eux-mêmes décomposés en sous-modes, et activés de façon cyclique. Leur spécification en termes d'une hiérarchie de tâches préemptives en SIGNALGTi a été mise en œuvre et simulée [80]. On a montré qu'elle satisfait un ensemble de propriétés au moyen de la méthode de vérification de SIGNAL [62].

Application au domaine de la production d'énergie

Dans le domaine de la production et de la distribution d'énergie, on est en présence de systèmes dont la sécurité est critique, pour diverses raisons : l'importance du service rendu (alimentation en électricité des hopitaux, par exemple), protection des matériels, risques pour l'environnement. Par ailleurs, ces systèmes sont souvent complexes, de par leur caractère distribué, comme c'est le cas pour le contrôle des réseaux de distribution, ou de par leur taille, en termes

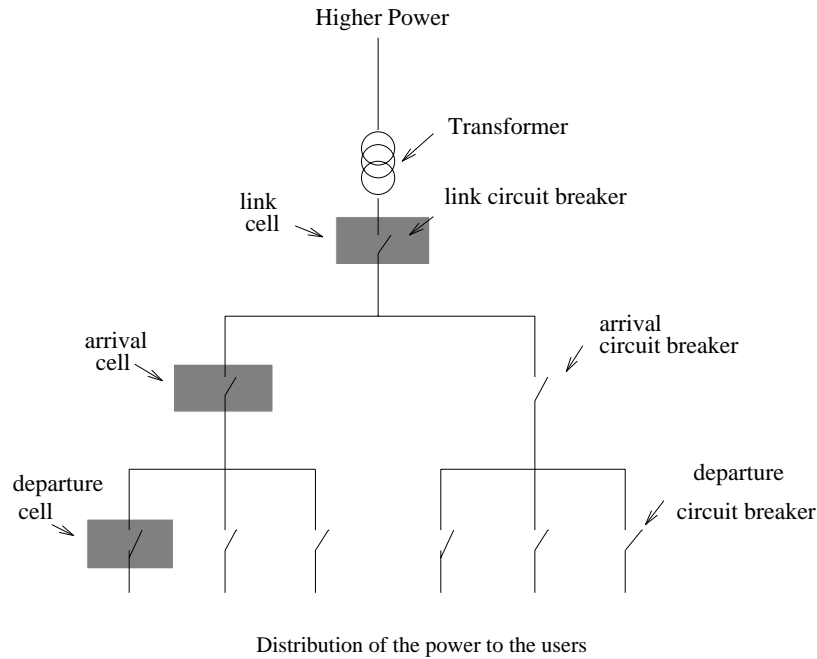


FIG. 3.9 – *Transformateur : topologie de la station.*

de nombre de capteurs (par centaines) ou d'automates de contrôle fonctionnant en parallèle ; leur conception en est particulièrement difficile, et semble pouvoir bénéficier d'environnements comme en propose l'approche synchrone, concernant la structuration des spécifications, comme leur vérification et leur mise en œuvre. Enfin, l'héritage des spécifications de contrôleurs obéit à une culture particulière : des langages « métier » de l'automatique logique sont utilisés depuis longtemps et largement, compromettant le passage à des langages plus modernes, indépendants de mises en œuvre par ailleurs datées (comme les relais électro-mécaniques), ou plus formels. Cet ensemble de caractéristiques alimente des travaux sur les mises en œuvre (voir Section 2.2.1) comme sur la vérification ou les techniques de synthèse de contrôleur [79], et motive aussi en partie les études sur les langages pour automates programmables industriels (voir Section 2.3.2). Il s'agit ici de l'aspect spécification, où le caractère événementiel du contrôleur, et sa structure étagée est traitée en termes de hiérarchie de préemption de tâches en *GTi*.

La station de transformation de courant

Les transformateurs dans le réseau de distribution. La méthodologie SIGNAL de spécification et de vérification a été utilisée pour l'application à un transformateur de tension du réseau EDF, comme il s'en trouve de très nombreux. Ce poste a pour but d'abaisser la tension du courant en vue de sa distribution dans les centres urbains. Durant l'exploitation d'un poste, plusieurs types de défauts peuvent apparaître sur le courant (dits : défaut de phase (PH), homopolaire (H), ou wattmétrique (W)). Pour protéger le matériel et l'environnement, plusieurs disjoncteurs ont été placés sur différentes parties du poste. Lors de l'apparition d'un défaut, des capteurs alertent les différents disjoncteurs, contrôlés par des contrôleurs locaux appelés *cellules* (cellule *liaison*, cellule *arrivée* et cellule *départ*) disposées selon la topologie illustrée en Figure 3.9

Fonctionnalité d'une cellule de départ. Cette cellule a deux activités : la phase de confirmation du défaut, suivie de la phase de traitement du défaut.

La *phase de confirmation* a pour but de faire disparaître les défauts fugitifs. Pour chacun des types de défauts un délai est déclenché, permettant de tester si ce défaut est permanent ou non. Ils sont testés en séquence, jusqu'à ce qu'un d'entre eux soit confirmé (c'est-à-dire présent à la fin du délai correspondant). De plus cette séquence est interrompue dès que le défaut disparaît, ou quand un des défauts préalablement examinés apparaît.

La *phase de traitement* commence dès que le défaut a été confirmé. On alterne alors entre une ouverture du disjoncteur durant un délai variable et sa fermeture afin de vérifier si le défaut n'a pas disparu. On ouvre le disjoncteur durant un délai donné, on le referme alors. Si le défaut est toujours présent, on répète cette opération pendant un

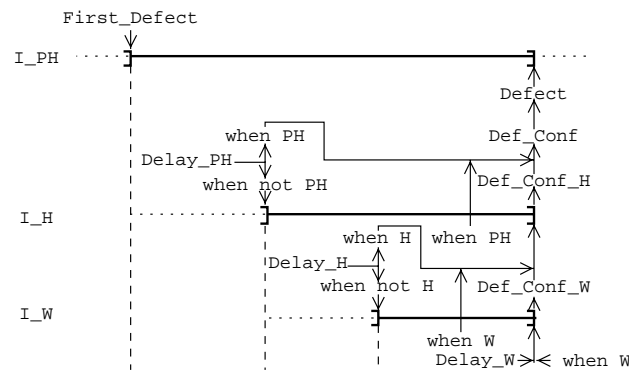


FIG. 3.10 – Transformateur : phase de confirmation : hiérarchie d'interruption.

certain nombre de cycles. Si le défaut persiste à la fin du dernier cycle, le disjoncteur est définitivement ouvert et son traitement est pris en charge par un intervenant extérieur.

La spécification de cette cellule (et des deux autres) a été réalisée en utilisant *SIGNALGTi* [80] et vérifiée en utilisant *SIGNAL* et *SIGALI* [62].

Spécification

Comme le suggèrent les paragraphes précédents, les comportements à décrire font intervenir différentes phases, décomposées en sous-phases. Or les intervalles et tâches de *GTi* se prêtent, comme on l'a vu en Section 2.2.2, à la construction de telles hiérarchies, qui vont trouver l'occasion de s'appliquer ici.

Hiérarchie de préemption. La Figure 3.10 montre comment la phase de confirmation peut être décrite comme une hiérarchie de tâche où chacune est interrompible par les tâches de plus haut niveau.

La tâche de confirmation, active sur l'intervalle I_{PH} , est démarrée (et l'intervalle ouvert) sur l'occurrence de la détection d'un premier défaut (*First_Defect*), et stoppée sur l'occurrence du signal *Defect* (qui a la valeur vrai en cas de confirmation (*Def_Conf*), et faux si le défaut a disparu entretemps), causant alors l'interruption de toutes les sous-tâches. La confirmation elle-même est décomposée en :

- attente d'un délai *Delay_PH*, à la fin duquel, si le défaut PH est détecté, alors *Def_Conf* est émis (ce qui cause la terminaison), sinon :
- la sous-tâche concernant le défaut H est démarrée, sur l'intervalle I_H , interrompible sur l'occurrence de PH. Cette sous-tâche se décompose elle-même de façon similaire en :
 - attente d'un délai *Delay_H*, à la fin duquel, si le défaut H est détecté, alors *Def_Conf_H* est émis (ce qui cause la terminaison), sinon :
 - la sous-tâche concernant le défaut W est démarrée, sur l'intervalle I_W , interrompible sur l'occurrence de H. Celle-ci procède à une attente d'un délai *Delay_W*, à la fin duquel, si le défaut W est détecté, alors *Def_Conf_W* est émis (ce qui cause la terminaison).

Validation

Cette spécification a donné lieu à validation par simulation graphique, en utilisant l'environnement *SIGNAL* pour la construction automatique de fenêtre d'acquisitions d'entrées et d'affichage de sorties sous forme de courbes, ainsi que par vérification, en utilisant les techniques déjà mentionnées précédemment.

Simulation. L'interface graphique de simulation est illustrée en Figure 3.11 La présence et la valeur de signaux, et les intervalles sont affichés sous forme de courbes ; pour les intervalles, les valeurs sont encodées par 1 pour *inside*, -1 pour *outside*, et 0 pour l'absence. Les événements sont codés par 1 pour présent et 0 pour absent.

La Figure 3.11 montre une simulation de la phase de confirmation.

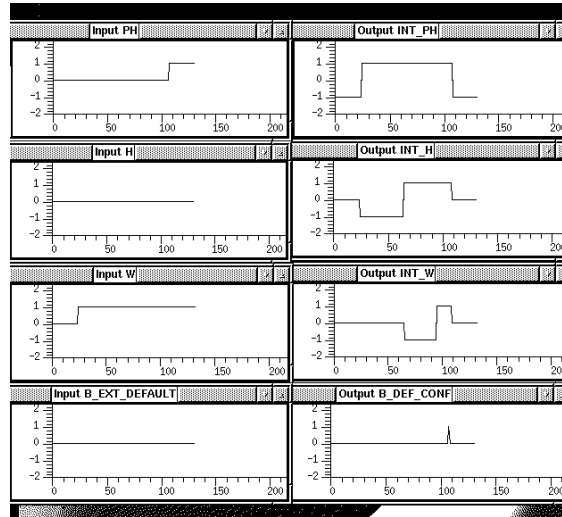


FIG. 3.11 – Transformateur : simulation de la phase de confirmation.

Vérification. On veut donc analyser les propriétés suivantes :

- (1) *La phase de confirmation et la phase de traitement ne sont jamais en cours aux mêmes instants*

Cette propriété peut être établie, en prouvant que l'ensemble des états correspondant à la situation où la phase de traitement et la phase de confirmation sont actives en même temps, n'est pas accessible depuis les états initiaux du système dynamique polynomial.

C'est ainsi, que l'on considère les deux intervalles I_Treat et I_PH , encodés par des booléens qui sont vrais quand le système est en phase de traitement, respectivement en phase de confirmation. Après la traduction du programme SIGNAL en système dynamique polynomial, on calcule l'ensemble des états $conf_and_treat$, où $I_Treat=1$ et $I_PH=1$. La méthode consiste alors à vérifier que l'ensemble des états $conf_and_treat$ n'est pas accessible depuis les états initiaux du système.

L'accessibilité de cet ensemble d'états peut être vérifiée en utilisant la fonction $reachable(prop)$ qui rend *vrai* si les états vérifiant la propriété sont accessibles depuis les états initiaux et *faux* sinon. Dans notre cas le résultat est faux.

- (2) *Quand un défaut apparaît, on aura nécessairement:*

- (a) *La confirmation du défaut,*
- (b) *ou la disparition du défaut,*
- (c) *ou l'apparition d'un défaut extérieur.*

ces trois possibilités ont été spécifiées en SIGNAL par un unique booléen DEFECT qui est *présent* quand l'une des trois possibilités est présente³.

Cette propriété peut être prouvée en vérifiant l'attractivité de l'ensemble des états F , où DEFECT est *présent*, à partir de l'ensemble E , le défaut apparaît (c'est-à-dire, il y a une occurrence de l'événement First_Defect). En appliquant la fonction $attractivity$, on peut prouver que F est attractif vis à vis de E (c'est-à-dire, chaque fois qu'un défaut apparaît, toutes les trajectoires du système amènent dans des états où DEFECT est présent).

- (3) *Quand un défaut a été confirmé, on aura nécessairement:*

- (a) *Soit le défaut ne disparaît pas et le signal Def_Break est émis,*
- (b) *soit le défaut disparaît avec le disjoncteur fermé*

L'événement Def_Break signale que le disjoncteur doit être ouvert définitivement (c'est-à-dire, la phase de traitement n'a pu faire disparaître le défaut). La méthode utilisée pour prouver cette propriété est la même que celle utilisée pour prouver la propriété (2). On calcule l'ensemble des états E , correspondant aux états où le défaut a été confirmé, et l'ensemble F correspondant à l'union des états où la condition (a) et la condition (b) sont vérifiées. Ainsi on montre que l'ensemble F est attractif vis à vis de E .

3. Ce signal est *présent* et *vrai* quand les conditions (a) et (c) sont vérifiées. il est *présent* et *faux* lorsque la condition (b) est vérifiée.

Activités liées à l'application de contrôle de transformateur

publications : conférences [80, 62], rapport de recherche, rapport de contrat.

Soumission à une revue.

outil : simulateur graphique utilisant l'environnement SIGNAL

encadrement : Stage d'été de Maîtrise Informatique (*co-encadré avec Hervé Marchand*)

collaborations : avec EDF : *Méthodes de synthèse de systèmes d'automatismes décrits par des automates à états finis*. (Convention INRIA – EdF n° M64/7C8321/E5/11, 12/93 – 12/96) Cette collaboration s'est faite avec Mazen Samaan, du groupe Acsar puis du département CCC (Contrôle Commande Centrales) de la DER (direction des études et recherches) d'EDF.

3.2.3 Animation comportementale

Une expérimentation concernant un environnement de simulation de systèmes physiques, animés en images de synthèse, a été effectuée en collaboration avec le projet SIAMES⁴ de l'IRISA/INRIA de Rennes (Stéphane Donikian), et Hervé Marchand pour la vérification. Dans l'aspect comportemental de l'animation, la spécification des comportements dynamiques a des aspects réactifs, hiérarchiques et préemptifs. SIGNAL et SIGNALGTi interviennent, dans le cadre d'une plateforme de simulation, pour la spécification de ces types de comportements, ainsi que dans la gestion des flots de données entre processus concurrents [35]. Des propriétés sur les comportements possibles ont été vérifiées. L'étude de cas traitée concerne la conduite de véhicules en lien avec le projet PRAXITELE (voir Figure 3.12). Plus récemment, une expérimentation de STATEMATE a été menée, pour évaluer son adéquation à la spécification de ces comportements ; les activités d'intégration avec le format DC+ ouvrent la perspective d'utiliser un langage comme STATEMATE en intégrant le code généré à la plate-forme de simulation.

Application à la simulation de systèmes dynamiques

Plateforme de simulation. Le contexte d'application envisagé ici est différent des autres, dans le sens où il est moins question de contrôle embarqué que de modélisation pour la simulation. Concrètement, il s'agit de s'intégrer à un environnement de synthèse d'images réalistes, à partir de modèles physiques, en éclairage comme en mouvement ; au sens plus large, cet environnement réalise des calculs de simulation et visualise éventuellement les résultats [10].

Dans cet environnement, la structure s'organise autour d'un noyau gérant le temps-réel et les communications et contrôlant l'ensemble des calculs. Ces derniers sont répartis en entités, chacune représentant un système individuel et indépendant (sans exclure naturellement des interactions). Une entité est composée de quatre modules : le modèle géométrique et mécanique (équations de mouvement) de l'entité, le contrôleur de «bas niveau» calculant les couples et forces à appliquer au modèle en fonction de commandes de plus haut niveau (vitesse, position), le comportement de «haut niveau» qui concerne les entités autonomes et le module capteur pour percevoir l'environnement.

Animation comportementale. Pour la simulation comportementale, et l'animation comportementale qui consiste à la visualiser, il s'agit de définir une animation en termes de perception, décision, communication et actions à effectuer. Elles s'adressent à la modélisation d'entités autonomes telles que des organismes vivants ou des personnages. Cette modélisation fait appel à des formes de systèmes de transition, et de systèmes réactifs qui en fonction d'événements et valeurs reçues de leur environnement changent d'état et émettent des commandes. L'état de ce système de transition est une représentation de l'environnement sur laquelle l'entité prend ses décisions.

La modélisation de comportements fait intervenir différentes lignes d'activité, pour la composition desquelles on a besoin de modularité et de concurrence. Le séquençement de comportements partiels en un comportement global peut faire intervenir des démarrages, arrêts, suspensions, reprises, qui peuvent bénéficier de structures hiérarchiques et préemptives pour sa description.

Dans le cas d'étude du système PRAXITELE de véhicules urbains partiellement automatisés, on a pris comme exemple celui de la modélisation de la conduite automobile (voir Figure 3.12).

4. Site web : <http://www.irisa.fr/siames/>



FIG. 3.12 – *Animation comportementale : simulation de conduite de véhicule*

Systèmes de transition parallèles hiérarchiques. Un formalisme a été proposé pour la description de comportements [10], fondé sur une notion de système de transition parallèles hiérarchiques (STPH). À un niveau hiérarchique, on a des états, liés par des transitions associées à des conditions de tirabilité. La hiérarchie permet de raffiner un état en lui associant un système de transition décrivant le sous-comportement qu'il abstrait (par exemple : mode de conduite avec dépassement, décomposé en déboîtement, accélération, et rabatement). Le parallélisme ou concurrence de spécification vise à mettre en parallèle les éléments relativement indépendants du comportement (par exemple : prise en compte de la signalisation routière, de l'état de la route, en parallèle de la gestion d'itinéraire et de la gestion de la circulation, et de la conduite elle-même).

La structure concurrente et hiérarchique est utilisée pour composer des sous-comportements qui contribuent diversement aux commandes qui seront émises à l'adresse de l'environnement (par exemple l'objectif de vitesse, qui peut être influencé par des considérations de géométrie ou état de la route (virage), manœuvre en cours (dépassement) ou autre). Il s'en suit la mise en place d'une forme de flot de données dans la remontée (synthèse) de valeurs calculées dans les différentes parties du comportement. Aux différents niveaux où plusieurs contributions doivent définir la sortie locale, une fonction particulière définit la valeur résultante.

Application de SIGNALGTi

C'est dans ce contexte que SIGNALGTi a été appliqué à la description de tels modèles à base de systèmes de transition hiérarchiques parallèles [35, 43]. Le recours à des langages réactifs et en particulier synchrones est naturel de par l'aspect réactif de cette simulation : chaque pas de simulation d'une entité va de la perception, en passant par la phase de calcul et décision, à l'action. Les structures mentionnées plus haut sont réminiscentes, pour la concurrence et la hiérarchie, d'ARGOS ou STATECHARTS, et pour la fonction de composition, d'ESTEREL.

Spécification. L'assemblage particulier de contrôle hiérarchique et de flot de données est modélisé en utilisant les structures de tâche de SIGNALGTi. Elles en constituent à la fois un modèle formel et un encodage concret en SIGNAL, donnant accès aux outils de l'environnement. Les intervalles de temps décomposent l'intervalle global du superviseur de conduite en sous-intervalles pour chacun des modes de conduite, et leurs événements d'ouverture et de fermeture sont conditionnés de façon à les synchroniser et séquencer.

L'aspect flot de données est représenté par la distribution au long de la structuration hiérarchique des données aux sous-comportements, en descendant, et en remontant par la fonction d'arbitrage composant les différentes contributions des sous-processus.

Le code C généré par l'environnement de programmation SIGNAL a été connecté à la plateforme de simulation en adaptant les fonctions gérant les entrées-sorties, et en intégrant l'appel de la fonction de transition au fonctionnement du noyau de la plateforme.

Vérification. On trouve dans ce type d'application une variante de la motivation usuelle de la vérification, qui est le caractère critique de la sûreté du contrôleur embarqué. Les méthodes et outils de vérification peuvent être vus ici comme des moyens d'analyser le modèle du comportement d'une entité dans le cadre de la simulation, comme par exemple les conséquences d'un certain processus de conduite automobile, même s'il n'est pas mis en œuvre de façon automatisée.

Ici comme ailleurs, un mode de vérification est l'utilisation, pour les propriétés statique, du calcul d'horloge du compilateur. Par exemple, si on s'intéresse à l'exclusivité des phases de déboîtement, dépassement et rabattement, on peut la vérifier en calculant l'horloge d'un signal présent dans l'intersection des trois intervalles correspondants, et en constatant sa nullité.

Pour les propriétés dynamiques, l'utilisation du *model-checking* et de l'outil SIGALI permet de se prononcer par exemple sur l'exclusion de la phase dépassement dans le module de gestion de la circulation, et de la phase de prise de virage dans le module de prise en compte de l'état de la route.

Spécification en STATEMATE

Les STPH ont une nature graphique et une structuration qui, comme on le mentionnait précédemment, rappelle les STATECHARTS. Les langages de STATEMATE sont donc plus proches des STPH que SIGNALGTi, et les utiliser pour les modéliser semble naturellement plus adéquat. Dans la perspectives des travaux d'intégration de STATEMATE et SIGNAL (voir Section 2.3.1), il est potentiellement possible de conserver l'accès à l'environnement de mise en œuvre et vérification de SIGNAL tout en utilisant STATEMATE. C'est dans cette optique qu'une étude a évalué à quel point et de quelle manière STATEMATE correspondait aux STPH [46]. En dehors des correspondances immédiates dues à la structure de système de transition parallèle hiérarchique, ont été mises en évidence des questions ayant trait notamment au caractère non-strictement synchrone de la sémantique de STATEMATE.

En effet, la transition dans STATEMATE s'accompagne d'un report des effets de toutes les actions au *step* suivant. Ce choix sémantique se traduit par une analyse de consistance simplifiée des programmes, puisqu'il exclut la possibilité de mettre en place un cycle de causalité dans l'instant comme c'est le cas dans les langages synchrones. Par contre dans notre cas, on souhaite spécifier la fonction d'arbitrage, produisant une valeur en fonction des contributions des sous-comportements, localement à chaque niveau hiérarchique ou composant parallèle. Or utiliser le mécanisme d'actions de STATEMATE introduit donc un délai supplémentaire à chaque composition hiérarchique ou parallèle, ce qui rend la sémantique de l'ensemble particulièrement sensible à la profondeur ou largeur de la structure, et donc peu lisible.

L'existence de *combinational assignments* en STATEMATE, dont les effets sont pris en compte à l'intérieur du *step*, peut toutefois répondre à ceci, et permettre de définir une fonction d'arbitrage qui synthétise à l'intérieur du pas de simulation la valeur à émettre vers l'extérieur.

Activités liées à l'application en animation

publications : conférence [35]

outil : démonstrateur : contrôleur connecté à la plateforme de simulation

encadrement : Co-encadrement avec Stéphane Donikian (SIAMES) des stages de DEA Informatique (IF-SIC, Université de Rennes 1) de Sébastien Gicquel : *Modélisation comportementale et programmation réactive synchrone* (1995) [43] et Antoine Hahusseau : *Spécification d'automates parallèles hiérarchisés à l'aide de langages réactifs synchrones pour la modélisation comportementale* (1997) [46]

collaborations : avec le projet SIAMES : Stéphane Donikian, Bruno Arnaldi

3.3 Enseignements tirés et perspectives

Ces applications explorent divers aspects de la problématique : mixité de la commande et du contrôle, phases et commutations, et hiérarchies de contrôle. Elles utilisent dans des contextes différents les structures de programmation, les spécificités de la composition synchrone, les outils d'assistance.

3.3.1 Utilisation plus approfondie des technologies synchrones

Les perspectives et continuations possibles, vis-à-vis de ces applications, vont dans le sens de l'utilisation plus approfondie des fonctionnalités de la technologie synchrone (telles que génération de mises en œuvre distribuées, évaluation de performances, vérification temporisée ou synthèse de contrôleur).

Elles concernent aussi le traitement plus élaboré, s'étendant à plus d'aspects de, par exemple :

- la nouvelle version de l'étude de cas de cellule de production diffusée par le FZI (concernant aussi des questions de tolérance aux fautes),
- la plateforme de simulation, où l'animation comportementale se fait en lien avec d'autres composants tels que gestion du temps-réel ou flux d'informations entre entités,
- la programmation robotique où l'application spécifique de *GTi* décrite plus haut pourrait être intégrée à une approche plus générale, comme celle d'ORCCAD [24], pour viser un environnement de programmation dédié à la robotique.

3.3.2 Langages dédiés

Par ailleurs, dans ces applications apparaissent des entités, qui sont celles effectivement manipulées par les utilisateurs dans ces domaines, pour la spécification des comportements comme pour la formulation des propriétés. Elles ne correspondent pas nécessairement au niveau de description donné par les primitives de langages comme SIGNAL, même si ces derniers ont l'expressivité suffisante. Ceci constitue un argument en faveur de langages dédiés, ou spécifiques à des domaines d'application, qui masquent une partie de la complexité des programmes en s'en abstrayant, et en fournissant un niveau de description indépendant du langage de modélisation sous-jacent.

Un exemple lié à ORCCAD en est MAESTRO [31, 106], langage spécifique au domaine de la programmation d'applications robotiques, qui fournit à un utilisateur des éléments comme les lois de commandes, événements d'exception ou tâches-robot, en s'abstrayant des protocoles de séquençement de lois de commandes. Ces derniers sont traités en détail dans la définition dans un formalisme plus général, en l'occurrence ESTEREL, qui est la base de la génération de code et vérification pour les applications. Il s'agit alors d'un niveau de spécificité de langage qui se construit sur le fondement des langages synchrones, là où ceux-ci sont définis comme des langages spécifiques au domaine des systèmes réactifs.

Chapitre 4

Bilan et perspectives

4.1 Bilan

Ce document fait la synthèse de mes travaux sur la programmation sûre des systèmes de contrôle/commande. Mes contributions ont eu pour cadre les modèles des systèmes réactifs, et plus spécifiquement la programmation synchrone. Elles ont porté plus particulièrement sur le séquençement, sur l'occurrence d'événements discrets, de tâches mettant en œuvre la commande continue échantillonnée.

Le spectre parcouru a été assez varié, du point de vue des techniques utilisées comme de celui des contextes dans lesquels se situaient mes travaux : débutant dans la modélisation logique pour la robotique téléopérée, j'ai poursuivi par l'application de la programmation synchrone à la robotique, la génération de plans en intelligence artificielle, et l'étude de langage de programmation flot de données. J'ai ensuite travaillé dans le contexte de la programmation synchrone, où s'est déroulée l'essentiel de l'activité que je présente ici. Le fil de conducteur de cette activité a toujours été la problématique de la spécification de systèmes de contrôle/commande, ainsi que la programmation suivant des méthodes avancées dans les domaines d'application dont la robotique est un exemple typique.

Cette activité a donné des résultats, synthétisés au long des chapitres précédents. Le besoin, et le gain auquel on peut s'attendre, concernant la structuration des spécifications par rapport à l'état actuel où on utilise directement des langages comme C ou l'assembleur pour programmer à la fois les lois de commande et leur séquençement ; ils concernent aussi les possibilités d'analyse, de validation et de mises en œuvre diverses des contrôleurs, pour lesquels la complexité même des systèmes de transition concernés requiert le support d'outils automatisés, eux-mêmes fondés sur des modèles formels qui permettent de définir les calculs utiles. Mes travaux ont contribué à cette problématique en explorant des formes de combinaisons multi-formalismes, et particulièrement celle du séquençement de tâches flot de données dans les langages réactifs. C'est ainsi que se situent mes propositions concernant l'introduction de tâches et d'intervalles de temps dans SIGNAL, les modélisations de STATEMATE et des langages de programmation des automatismes industriels comme ceux de la norme IEC 1131, et ma participation à des expérimentations en contrôle de production, vision robotique, ou contrôle d'automatisme.

Ces résultats ont donné lieu

- à des publications (dont cinq articles de revue et une quinzaine de conférences internationales avec comité de sélection),
- au développement d'outils prototypes, et de composants dans des environnements plus vastes, ou de démonstrateurs d'expérimentation,
- à un travail en équipe, notamment :
 - au co-encadrement de deux thèses,
 - au travail avec deux ingénieurs-experts,
 - à l'encadrement de six stages de DEA, et d'autres stages,

ainsi qu'à un enseignement en option *Programmation d'Applications Temps-Réel* du DEA d'Informatique de l'Université de Rennes 1,

- à des coopérations
 - internes au laboratoire de l'IRISA / INRIA-Rennes (projets SIAMES et TEMIS/VISTA),
 - académiques (dans le cadre de mon post-doctorat, dans la communauté synchrone, ou avec le TIFR en Inde via le CEFIPRA)

- ou industrielles (avec Électricité de France, ou dans le projet Européen Esprit SACRES),
- à diverses formes d'activité de diffusion ou d'animation comme
 - la participation à l'organisation de journées thématiques,
 - les relectures pour les conférences ou les journaux,
 - la participation à des jurys,
 - ou la représentation de l'INRIA.

4.2 Perspectives

Des perspectives assez détaillées émaillent déjà le document, aux endroits idoines. On peut rappeler brièvement les principales de celles qui émergent des travaux passés :

- une voie concerne les principes fondamentaux du séquençement de tâches, par la définition de modèles spécifiques sur lesquels puissent se faire des calculs, par exemple sur des intervalles de temps en lien avec le calcul d'horloges de SIGNAL, et par des études comme celle de la mise en perspective des différentes formes de préemption dans les langages réactifs,
- un travail peut se poursuivre sur l'intégration multi-formalisme, particulièrement flot de données et impératif :
 - au niveau des langages eux-mêmes et de la structure de leurs opérateurs, comme les automates de mode en sont une illustration, en s'inspirant éventuellement de langages «métier»,
 - à celui de la réalisation, où on peut intervenir au niveau source ou au niveau cible, avec des implications quant à l'intérêt (en quantité d'information) et au coût (en volume) d'avoir un modèle formel complet, global de l'application, et les liens de ces problèmes avec les approches de la modularité dans les langages synchrones.
- les instanciations de la problématique, où transférer les résultats, peuvent se trouver dans le prolongement des langages étudiés :
 - la modélisation de STATEMATE en SIGNAL peut donner lieu à des suites dans les variantes de STATEMATE, non pas tant pour décliner des nuances, mais pour entrer en contact avec des domaines d'applications et des problématiques éventuellement différentes, comme par exemple ce serait le cas de la variante de STATECHARTS présente dans le langage de spécification UML répandu dans le domaine des télécommunications ;
 - les travaux concernant la modélisation en SIGNAL du GRAFCET et des langages de la norme IEC 1131-3 sont encore en cours, et figurent donc dans les perspectives ; au-delà de cela, ils ouvrent des pistes quant aux domaines d'applications, comme aux architectures d'exécution à base d'automates programmables.

D'autres perspectives, toujours dans le domaine de la programmation de systèmes de contrôle/commande, s'inscrivent dans le contexte de mon arrivée dans le projet BIP de l'INRIA Rhône-Alpes, où l'accent est mis sur la programmation de systèmes de contrôle/commande, notamment en lien avec l'environnement ORCCAD. Elles concernent :

- l'exploration de la problématique de langages spécifiques au domaine, ou langages «métier», dans le cas de la robotique et des systèmes électro-mécaniques, des différents points de vue :
 - des langages de spécification d'applications ou missions et de tâches, notamment dans la combinaison entre les niveaux commande (flot de données, éventuellement multi-cadence), tâche et application, où une forme de «compatibilité» avec la norme IEC 1131 peut être étudiée ;
 - des langages pour la vérification, où on peut souhaiter une spécification de propriétés en termes du domaine d'application (tâches, lancement de loi de commande), plutôt qu'intrinsèques au formalisme de modélisation ; on peut y répondre par la génération automatique d'observateurs ou bien de formules de logique temporelle à partir de la spécification,
- la synthèse de contrôleur, en tant que méthode d'obtention de contrôleurs sûrs à partir de spécifications très déclaratives, par expression des objectifs à atteindre en terme de propriétés temporelles ; on s'intéresserait à son applicabilité dans le cadre de systèmes robotiques, à la façon dont on peut l'utiliser sur les structures de base en présence : les tâches et leur événements de contrôle et d'exceptions. Un parallèle avec la robotique de téléopération, où on pose des contraintes sur les degrés de liberté laissés à l'opérateur, au sens géométrique, pourrait être tiré au sens des évolutions permises dans un automate où sont interdites les transitions ou trajectoires considérées dangereuses. Plus généralement, la problématique serait celle des méthodes d'utilisation concrète de cette technique encore neuve qu'est la synthèse de contrôleur à événements discrets.

-
- l'exploitation du caractère spécifique des structures manipulées et des modèles synchrones que l'on construit, pour réaliser une modélisation et un codage formel favorisant l'utilisation des outils (compilation, analyse, vérification) ; en particulier on manipule des entités dont une partie du comportement est prédéfinie, d'une façon qui peut être exploitée plus avant que dans le cas général par les outils synchrones de vérification ou de mise en œuvre. Un exemple en serait le fait qu'on peut désigner en ORCAD quelles parties des calculs peuvent être désynchronisées des autres dans le flot de données de la tâche de commande, de par leur cadence moins grande.
 - d'autres domaines d'application, comme :
 - la programmation d'expériences spatiales, où les procédés contrôlés dans le cadre des expériences scientifiques sont variés, et où les possibilités d'intervention manuelle sont réduites,
 - la robotique médicale, qui présente des caractéristiques de finesse du contrôle à réaliser et de criticité de la sécurité de son automatisation.

Bibliographie

- [1] Alami (Rachid), Chatila (Raja), Fleury (Sara), Ghallab (Malik) et Ingrand (François-Félix). – An architecture for autonomy. *International Journal of Robotics Research* (special issue on Integrated Architecture for Robot Control and Programming), vol. 17, n4, avril 1998, pp. 315–337.
- [2] Allen (James F.). – Maintaining knowledge about temporal intervals. *Communications of the ACM*, vol. 26, n 11, 1983, pp. 832–843.
- [3] Amagbegnon (Tochéou), Besnard (Loïc) et Le Guernic (Paul). – Implementation of the data-flow synchronous language signal. In : *Proceedings of the ACM International Conference on Programming Languages Design and Implementation, PLDI'95*, pp. 163–173.
- [4] Amagbegnon (Tocheou Pascal), Le Guernic (Paul), Marchand (Hervé) et Rutten (Eric). – SIGNAL – the specification of a generic, verified production cell controller, chap. VII, pp. 115–129. – Springer Verlag, janvier 1995, *Lecture Notes in Computer Science (LNCS)*. nr. 891.
- [5] Amagbegnon (Tocheou Pascal), Le Guernic (Paul), Marchand (Hervé) et Rutten (Eric). – *The SIGNAL data flow methodology applied to a production cell*. – Rapport de Recherche n2522, INRIA, mars 1995. <http://www.inria.fr/RRRT/RR-2522.html> (Publication Interne IRISA, Rennes, no. 917, Mars 1995; <http://www.irisa.fr/EXTERNE/bibli/pi/pi917.html>).
- [6] André (Charles). – SYNCCHARTS. In : *Proceedings of IEEE SMC'96*, Lille, France, Juillet 1996.
- [7] André (Charles), Marmorat (Jean-Paul) et Paris (Jean-Pierre). – Execution machines for ESTEREL. In : *Proceedings of the European Control Conference, ECC'91*, Grenoble, France, juillet 1991.
- [8] Arbab (Farhad), Herman (Ivan) et Spilling (Per). – An overview of Manifold and its implementation. *Concurrency: Practice and Experience*, vol. 5, n1, février 1993, pp. 23–70.
- [9] Arbab (Farhad) et Rutten (Eric). – MANIFOLD: a programming model for massive parallelism. In : *Proceedings of the Working Conference on Massively Parallel Programming Models, MPPM'93*, Berlin, Germany, September 20-23. pp. 151–159. – (IEEE Publ.).
- [10] Arnaldi (Bruno), Donikian (Stéphane), Chauffaut (Alain), Cozot (Rémi) et Thomas (Gwenola). – Real-time simulation platform for dynamic systems. In : *Proceedings of the IROS'97 (IEEE/RSJ International Conference on Intelligent Robots and Systems) Workshop on Dynamic Simulation: Methods and Applications*, Grenoble, France, September 1997, pp. 32–41.
- [11] Arnold (André). – *Systèmes de transitions finis et sémantique des processus communicants*. – Masson, 1992.
- [12] Aubry (Pascal). – *Mises en oeuvre distribuées de programmes synchrones*. – Thèse de Doctorat en Informatique, Université de Rennes 1, IFSIC, octobre 1997.
- [13] Baufreton (Philippe), Granier (Hugues), Méhaut (Xavier) et Rutten (Eric). – The SACRES approach to embedded systems applied to aircraft engine controllers. In : *Proceedings of The 22nd IFAC/IFIP Workshop on Real Time Programming, WRTF'97*, Lyon, France, September 15–17, 1997. – Elsevier. (Invited presentation).
- [14] Beauvais (Jean-René). – *Modélisation de STATECHARTS en SIGNAL pour la conception de systèmes critiques temps-réel*. – Thèse de Doctorat en Informatique, IFSIC, Université de Rennes 1, février 1999.
- [15] Beauvais (Jean-René), Gautier (Thierry), Le Guernic (Paul), Houdebine (Roland) et Rutten (Eric). – A translation of STATECHARTS into SIGNAL. In : *Proceedings of the International Conference on Application of Concurrency to System Design, CSD'98*, Aizu-Wakamatsu, Japan, March 23–26. pp. 52–62. – IEEE.
- [16] Beauvais (Jean-René), Houdebine (Roland), Le Guernic (Paul), Rutten (Eric) et Gautier (Thierry). – *A translation of Statecharts and Activitycharts into Signal equations*. – Rapport de Recherche n3397, INRIA, avril 1998. <http://www.inria.fr/RRRT/RR-3397.html> (Publication Interne IRISA, Rennes, no. 1182, Avril 1998; <http://www.irisa.fr/EXTERNE/bibli/pi/1182/1182.html>).

- [17] Belhadj (Mohammed). – *Conception d'architectures en utilisant Signal et VHDL*. – Thèse de Doctorat en Informatique, Université de Rennes I, IFSIC, décembre 1994.
- [18] Benveniste (Albert) et Berry (Gérard). – The synchronous approach to reactive and real-time systems. *Proceedings of the IEEE*, special issue on *Another Look at Real Time Programming*, vol. 79, n9, sep 1991.
- [19] Benveniste (Albert), Caspi (Paul), Guernic (Paul Le) et Halbwachs (Nicolas). – *Data-flow Synchronous Languages*. – Rapport de Recherche n2089, INRIA, octobre 1993. <http://www.inria.fr/RRRT/RR-2089.html> (Publication Interne IRISA, Rennes, no. 768, Octobre 1993 ; <http://www.irisa.fr/EXTERNE/bibli/pi/pi768.html>).
- [20] Benveniste (Albert), Gautier (Thierry), Le Guernic (Paul) et Rutten (Eric). – Distributed code generation of dataflow synchronous programs: the SACRES approach. In : *Proceedings of the Eleventh International Symposium on Languages for Intensional Programming, ISLIP'98*, Menlo Park, California, May 7-9, 1998. pp. 26–54. – UVIC Faculty of Engineering LACIR Report, University of Victoria (Canada).
- [21] Berry (Gérard). – Preemption in concurrent systems. In : *Proceedings of the Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS'93*, Bombay, India. – Springer-Verlag. LNCS nr. 761.
- [22] Berry (Gerard) et Gonthier (Georges). – The ESTEREL synchronous programming language: design, semantics, implementation. *Science of Computer Programming*, vol. 19, 1992, pp. 87–152.
- [23] Besnard (Loïc), Bournai (Patricia), Gautier (Thierry), Halbwachs (Nicolas), Nadjm-Tehrani (Simin) et Res-souche (Annie). – Design of a multi-formalism application and distribution in a data-flow context: an example. In : *Proceedings of The 12th International Symposium on Languages for Intensional Programming, ISLIP'99*. – NCSR Demokritos, Athens, Greece, juin 1999.
- [24] Borelly (Jean-Jacques), Coste-Manière (Ève), Espiau (Bernard), Kapellos (Konstantinos), Pissard-Gibollet (Roger), Simon (Daniel) et Turro (Nicolas). – The ORCCAD architecture. *International Journal of Robotics Research* (special issue on Integrated Architecture for Robot Control and Programming), vol. 17, n4, avril 1998, pp. 338–359.
- [25] Bournai (Patricia), Le Borgne (Michel) et Marchand (Hervé). – *Environnement de conception d'automatismes discrets basé sur le langage SIGNAL*. – Rapport de Recherche n 3254, INRIA, septembre 1997. <http://www.inria.fr/RRRT/RR-3254.html> (Publication Interne IRISA, Rennes, no. 1124, Septembre 1997 ; <http://www.irisa.fr/EXTERNE/bibli/pi/pi1124.html>).
- [26] Bournai (Patricia) et Le Guernic (Paul). – *Un environnement graphique pour le langage SIGNAL*. – Rapport de Recherche n2040, INRIA, septembre 1993. <http://www.inria.fr/RRRT/RR-2040.html> (Publication Interne IRISA, Rennes, no. 741, Septembre 1993 ; <http://www.irisa.fr/EXTERNE/bibli/pi/pi741.html>).
- [27] Boussinot (Frédéric) et de Simone (Robert). – The ESTEREL language. *Proceedings of the IEEE*, special issue on *Another Look at Real Time Programming*, vol. 79, n9, septembre 1991, pp. 1293–1304.
- [28] Boussinot (Frédéric) et Susini (Jean-Ferdinand). – The sugarcubes tool box: A reactive java framework. *Software-Practice and Experience*, vol. 28, n14, décembre 1998, pp. 1531–1550.
- [29] CEI/IEC. – *International Standard for Programmable Controllers: Programming Languages*. – Rapport technique nIEC 1131 partie 3, CEI (Commission Électrotechnique Internationale)/IEC (International Electrotechnical Commission), 1993.
- [30] Coste-Manière (Ève), Espiau (Bernard) et Rutten (Eric). – A task-level robot programming language and its reactive execution. In : *Proceedings of the 1992 IEEE International Conference on Robotics and Automation*, Nice, France, May 12–14. pp. 2751–2756. – IEEE.
- [31] Coste-Manière (Ève) et Turro (Nicolas). – The MAESTRO language and its environment : Specification, validation and control of robotic missions. In : *Proceedings of the 10th IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS'97*, Grenoble, 1997.
- [32] David (René). – Grafset: a powerful tool for specification of logic controllers. *IEEE Transactions on Control Systems Technology*, vol. 3, n3, septembre 1995, pp. 253–268.
- [33] David (René) et Alla (Hassane). – *Du GRAFCET aux réseaux de Petri*. – Hermès, Paris, 1989.
- [34] de Simone (Robert). – Higher-level synchronizing devices in MEIJE-SCCS. *Theoretical Computer Science*, vol. 35, 1985, pp. 245–267.
- [35] Donikian (Stéphane) et Rutten (Eric). – Reactivity, concurrency, data-flow and hierarchical preemption for behavioral animation. In : *Programming Paradigms in Graphics*. pp. 137–153, 169. – Springer.

- [36] Dutertre (Bruno). – *Spécification et preuve de systèmes dynamiques*. – Thèse de Doctorat en Informatique, Université de Rennes I, IFSIC, décembre 1992.
- [37] Dutertre (Bruno), Le Borgne (Michel) et Le Guernic (Paul). – The cat and mouse in the synchronous paradigm. In: *Joint Workshop on Discrete Event Systems (WODES'92)*, pp. 117–120.
- [38] EP-ATR Project. – SIGNAL: A formal design environment for real-time systems. In: *Proceedings of the Sixth International Joint Conference on Theory and Practice of Software Development, TAPSOFT '95*, Århus, Denmark, May, 1995. pp. 789–790. – Springer Verlag. LNCS nr. 915.
- [39] Fleureau (Jean-Luc). – *Vers une méthodologie de programmation en télérobotique : comparaison des approches Pilot et Grafset*. – Thèse de Doctorat en Informatique, IFSIC, Université de Rennes 1, juillet 1998.
- [40] Gaffé (Daniel). – *Le modèle GRAFCET : réflexion et intégration dans une plate-forme multi-formalisme synchrone*. – Thèse de Doctorat en Informatique, Université de Nice-Sophia-Antipolis, janvier 1996.
- [41] Gautier (Thierry) et Le Guernic (Paul). – Code generation in the sacres project. In: *Towards System Safety, Proceedings of the Safety-critical Systems Symposium, SSS'99*. – Huntingdon, UK, février 1999.
- [42] Gelernter (D.) et Carriero (N.). – Coordination languages and their significance. *Communications of the ACM*, vol. 35, février 1992, pp. 97–107.
- [43] Gicquel (Sébastien). – *Modélisation comportementale et programmation réactive synchrone*. – Rapport de DEA en informatique, IFSIC, Université de Rennes 1, septembre 1995.
- [44] Grazebrook (Alvery). – Sacres - formalism for real projects. In: *Safer Systems*, éd. par F. Redmill et T. Anderson. – Springer-Verlag.
- [45] Guéguen (Hervé). – Mixing statecharts and signal for the specification of control. In: *Proceedings of the IFAC Workshop on Algorithm and Architecture for real time control, AARTC97*, Vilamoura, Spain, éd. par IFAC.
- [46] Hahusseau (Antoine). – *Spécification d'automates parallèles hiérarchisés à l'aide de langages réactifs synchrones pour la modélisation comportementale*. – Rapport de DEA en informatique, IFSIC, Université de Rennes 1, septembre 1997.
- [47] Halbwachs (Nicolas). – *Synchronous programming of reactive systems*. – Kluwer, 1993.
- [48] Halbwachs (Nicolas). – Synchronous programming of reactive systems – a tutorial and commented bibliography. In: *Proceedings of the International Conference on Computer-Aided Verification, CAV'98*, Vancouver, Canada, june 1998. – Springer-Verlag. LNCS nr. 1427.
- [49] Halbwachs (Nicolas), Caspi (Paul), Raymond (Pascal) et Pilaud (Daniel). – The synchronous dataflow programming language LUSTRE. *Proceedings of the IEEE*, special issue on *Another Look at Real Time Programming*, vol. 79, n9, septembre 1991, pp. 1305–1320.
- [50] Harel (David). – STATECHARTS: A visual formalism for complex systems. *Science of Computer Programming*, vol. 8, n3, juin 1987, pp. 231–274.
- [51] Harel (David) et Naamad (Amnon). – The STATEMATE semantics of STATECHARTS. *ACM Transactions on Software Engineering and Methodology*, vol. 5, n4, octobre 1996, pp. 293–333.
- [52] Harel (David) et Pnueli (Amir). – On the development of reactive systems. In: *Logic and Models of Concurrent Systems, NATO Advanced Study Institute on Logics and Models for Verification and Specification of Concurrent Systems*. – Springer-Verlag.
- [53] Hertzberg (Joachim) et Horz (Alexander). – Towards a theory of conflict detection and resolution in nonlinear plans. In: *Proceedings of the International Joint Conference on Artificial Intelligence, IJCAI'89*, Detroit, Michigan, August 1989.
- [54] Jiménez-Fraustro (Fernando) et Rutten (Eric). – Modélisation synchrone de standards de programmation de systèmes de contrôle : le langage ST de la norme CEI 1131-3. In: *Actes de la Journée d'études sur les Nouvelles Percées dans les Langages pour l'Automatique*, Amiens, 25 novembre 1999.
- [55] Jiménez-Fraustro (Fernando) et Rutten (Eric). – A synchronous model of the PLC programming language ST. In: *Proceedings of the Work In Progress session, 1st Euromicro Conference on Real Time Systems, ERTS'99*, York, England, June 9–11, 1999, pp. 21–24. – Disponible sur la page web : www.idt.mdh.se/ecrts99/wip.htm.
- [56] Jourdan (Muriel), Lagnier (Fabienne), Raymond (Pascal) et Maraninchi (Florence). – A multiparadigm language for reactive systems. In: *Proceedings of the 5th IEEE International Conference on Computer Languages, ICCL'94*, Toulouse, France, Mai 1994.

- [57] Jourdan (Muriel), Layaïda (Nabil) et Roisin (Cécile). – A survey on authoring techniques for temporal scenarios of multimedia documents. In : *Handbook of Internet and Multimedia Systems and Applications, part 1 : Tools and Standards*, éd. par Fuhr (B.). – CRC Press.
ftp://ftp.inrialpes.fr/pub/opera/publications/hand_ps.ps.gz.
- [58] Kapellos (Konstantinos). – *Environnement de programmation des applications robotiques réactives*. – Thèse de Doctorat en Informatique, l'Ecole des Mines de Paris, Sophia-Antipolis, novembre 1994.
- [59] Klint (Paul). – A meta-environment for generating programming environments. In : *Proceedings of the ME-TEOR Workshop on Methods Based on Formal Specification*. pp. 105–124. – Springer Verlag, LNCS nr. 490.
- [60] Kountouris (Apostolos). – *Outils pour la validation temporelle et l'optimisation de programmes synchrones*. – Thèse de Doctorat en Informatique, Université de Rennes 1, IFSIC, octobre 1998.
- [61] Kountouris (Apostolos) et Wolinski (Christophe). – False path analysis based on a hierarchical control representation. In : *Proceedings of ISSS'98*. – Hsinchu, Taiwan, R.O.C., décembre 1998.
- [62] Le Borgne (Michel), Marchand (Hervé), Rutten (Eric) et Samaa (Mazen). – Formal verification of SIGNAL programs: application to a power transformer station controller. In : *Proceedings of the Fifth International Conference on Algebraic Methodology and Software Technology, AMAST '96*, Munich, July 1–5. pp. 271–285. – Springer Verlag, LNCS nr. 1101.
- [63] Le Guernic (Paul), Gautier (Thierry), Le Borgne (Michel) et Le Maire (Claude). – Programming real-time applications with SIGNAL. *Proceedings of the IEEE*, special issue on *Another Look at Real Time Programming*, vol. 79, n9, sep 1991, pp. 1321–1336.
- [64] Le Guernic (Paul), Machard (Sylvain) et Rutten (Eric). – Répartition de programmes SIGNAL. In : *Actes des Rencontres Francophones du Parallélisme des Architectures et des Systèmes, RenPar'10*, Strasbourg, 9 – 12 juin 1998.
- [65] Le Guernic (Paul) et Rutten (Eric). – Experiments with the synchronous methodology illustrating its support of predictability. In : *Proceedings of the 21st IFAC/IFIP Workshop on Real Time Programming, WRTF'96*, Gramado, RS, Brazil, November 4–6, 1996. pp. 81–86. – Elsevier.
- [66] Le Parc (Philippe). – *Apports de la méthodologie synchrone pour la définition et l'utilisation du langage GRAFCET*. – France, Thèse de Doctorat en Informatique, Université de Rennes 1, janvier 1994.
- [67] Le Parc (Philippe), L'her (Dominique), Scharbarg (Jean-Luc) et Marcé (Lionel). – Grafcet revisited with a synchronous data-flow language. *IEEE Transactions on Systems, Man and Cybernetics - Part A: Systems and Human*, vol. 29, n3, mai 1999.
- [68] Le Rest (Erwan). – *PILOT : un langage pour la télérobotique*. – Thèse de Doctorat en Informatique, IFSIC, Université de Rennes 1, juin 1996.
- [69] Lewerentz (C.) et Lindner (T.) (édité par). – *Formal Development of Reactive Systems – Case Study Production Cell*. – Springer Verlag, janvier 1995, *Lecture Notes in Computer Science (LNCS)*. nr. 891.
- [70] Maler (Oded) (édité par). – *Hybrid and Real-Time Systems – Proceeding of the International Workshop HART'97*, Grenoble, France, March 1997. – Springer Verlag, mars 1997, *Lecture Notes in Computer Science (LNCS)*. nr. 1201.
- [71] Maraninchi (Florence). – Operational and compositional semantics of synchronous automaton compositions. In : *Proceedings of CONCUR'92*. – Springer Verlag, LNCS nr. 630.
- [72] Maraninchi (Florence). – *Modélisation et validation des systèmes réactifs : un langage synchrone à base d'automates*. – Habilitation à diriger des recherches, Université Joseph Fourier - Grenoble 1, mai 1997.
- [73] Maraninchi (Florence) et Halbwachs (Nicolas). – Compiling ARGOS into Boolean equations. In : *Formal Techniques in Real-Time and Fault-Tolerant Systems, FTRTFT'96*, Uppsala, Sweden, 1996. pp. 72–90. – Springer-Verlag, LNCS nr. 1135.
- [74] Maraninchi (Florence) et Rémond (Yann). – Mode-automata: About modes and states for reactive systems. In : *Proceedings of the European Symposium On Programming, ESOP'98*, Lisbon, Portugal. – Springer-verlag, LNCS.
- [75] Marchand (Éric). – *Stratégies de perception par vision active pour la reconstruction et l'exploration de scènes statiques*. – Thèse de Doctorat en Informatique, IFSIC, Université de Rennes 1, juin 1996.
- [76] Marchand (Eric), Chaumette (François) et Rutten (Eric). – Real time active visual reconstruction using the synchronous paradigm. In : *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS'95*, Pittsburgh, Pennsylvania, USA, August 5–9, 1995, pp. 96–102.

- [77] Marchand (Eric), Rutten (Eric) et Chaumette (François). – Applying the synchronous approach to real time active visual reconstruction. *IEEE Transactions on Control Systems Technology*, vol. 5, n2, mars 1997, pp. 200–216.
- [78] Marchand (Eric), Rutten (Eric), Marchand (Hervé) et Chaumette (François). – Specifying and verifying active vision-based robotic systems with the SIGNAL environment. *International Journal of Robotics Research* (special issue on Integrated Architecture for Robot Control and Programming), vol. 17, n4, avril 1998, pp. 418–432.
- [79] Marchand (Hervé), Bournai (Patricia), Le Borgne (Michel) et Le Guernic (Paul). – A design environment for discrete-event controllers based on the signal language. In : *1998 IEEE International Conf. On Systems, Man, And Cybernetics*, pp. 770–775. – San Diego, California, USA, octobre 1998.
- [80] Marchand (Hervé), Rutten (Eric) et Samaan (Mazen). – Synchronous design of a transformer station controller with SIGNAL. In : *Proceedings of the 4th IEEE Conference on Control Applications, CCA'95*, Albany, New York, USA, September 28–29. pp. 754–759. – IEEE.
- [81] Marcos (Mar), Moisan (Sabine) et del Pobil (Angel P.). – Knowledge modeling of program supervision task. In : *Proceedings of the 11th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems, IEA-98-AIE*, Benicassim, Spain, June 1998. pp. 124–133, Vol I. – Springer Verlag. LNCS nr. 1415.
- [82] Martinez (Florent). – *Séquençement de tâches flot de données et intervalles de temps en SIGNAL*. – Rapport de DEA en informatique, IFSIC, Université de Rennes 1, septembre 1994.
- [83] Materne (Stephan) et Hertzberg (Joachim). – MTMM - correcting and extending time map management. In : *Proceedings of the European Workshop on Planning, EWSP'91*, Sankt Augustin, Germany, March. – Springer-Verlag. LNAI (LNCS) nr. 522.
- [84] Nebut (Mirabelle). – *Modélisation de STATEMATE en SIGNAL : le langage impératif des actions*. – Rapport de DEA en informatique, IFSIC, Université de Rennes 1, septembre 1998.
- [85] Paoletti (Jean-Christophe). – *Spécification et sémantique opérationnelle d'un langage de contrôle d'exécution de plans d'actions pour la télérobotique*. – Thèse de Doctorat en Informatique, IFSIC, Université de Rennes 1, 1991.
- [86] Parmentier (Thibault), Ziébelin (Danielle) et Rechenmann (François). – Environnement de résolution de problèmes distribué. In : *Actes du 11^{ème} Congrès de Reconnaissance des Formes et Intelligence Artificielle, RFIA'98*, Clermont-Ferrand, France, 20–22 janvier 98, pp. II-265–274. – <ftp://ftp.inrialpes.fr/pub/sherpa/publications/parmentier98a.ps.gz>.
- [87] Perraud (Jean), Roux (Olivier) et Huon (Marc). – Operational semantics of a kernel of the language ELECTRE. *Theoretical Computer Science*, vol. 97, n1, avril 1992, pp. 83–104.
- [88] Pinchinat (Sophie), Rutten (Eric) et Shyamasundar (R.K.). – Preemption primitives in reactive languages (a preliminary report). In : *Algorithms, Concurrency and Knowledge — Proceedings of the Asian Computing Science Conference, ACSC '95*, Pathumthani, Thailand, December 11–13. pp. 111–125. – Springer Verlag. LNCS nr. 1023.
- [89] Pinchinat (Sophie), Rutten (Éric) et Shyamasundar (R.K.). – Taxonomy and expressiveness of preemption: a syntactic approach. In : *Proceedings of the Asian Computing Science Conference, ASIAN'98*, Manila, The Philippines, December 8–10, 1998. pp. 125–141. – Springer Verlag. LNCS nr. 1583.
- [90] Qhénec'hdu (Yves) et Villerman-Lecolier (Gérard) (édité par). – *Hybrid Dynamical Systems / Les Systèmes Dynamiques Hybrides — Proceeding of the 3rd International Conference on Automation of Mixed Processes, ADPM'98*, Reims, France, 19–20 mars 1998. – Presses Universitaires de Reims, mars 1998.
- [91] Richard (Martin) et Roux (Olivier). – Conjunction of synchronous and asynchronous languages for reactive programming. In : *Proceedings of the 8th Euromicro Workshop on Real Time Systems, ERTS'95*, l'Aquila, Italy, June 12 - 14, 1996.
- [92] Roux (Olivier). – *Une approche de programmation des systèmes réactifs asynchrones*. – Habilitation à diriger des recherches, LAN / Ecole Centrale de Nantes, septembre 1992.
- [93] Rutten (E.P.B.M.) et Thiébaux (Sylvie). – *Semantics of MANIFOLD: specification in ASF+SDF and extension*. – Research Report nCS-R 9269, CWI, Department of Interactive Systems, décembre 1992.
- [94] Rutten (Eric). – A temporal representation for imperatively structured plans of actions. In : *Proceedings of the 5th Portuguese Conference on Artificial Intelligence, EPIA '91*, Albufeira, Portugal, October 1–3. pp. 165–179. – Springer-Verlag. LNAI (LNCS) nr. 541.

-
- [95] Rutten (Eric) et Hertzberg (Joachim). – Temporal planner = nonlinear planner + time map manager. *A.I. Communications*, vol. 6, n1, mars 1993, pp. 18–26.
 - [96] Rutten (Eric) et Le Guernic (Paul). – Sequencing data flow tasks in signal. In: *Proceedings of the ACM SIGPLAN Workshop on Language, Compiler and Tool Support for Real-Time Systems*, Orlando, Florida, June 21, 1994. – Disponible sur la page web: http://www.cs.umd.edu/users/pugh/sigplan_realtime_workshop/lct-rts94/.
 - [97] Rutten (Eric) et Marcé (Lionel). – An imperative language for task-level planning: definition in temporal logic. *Artificial Intelligence in Engineering*, vol. 8, n4, 1993, pp. 235–251.
 - [98] Rutten (Eric), Marchand (Eric) et Chaumette (François). – An experiment with reactive data-flow tasking in active robot vision. *Software – Practice & Experience*, vol. 27, n5, mai 1997, pp. 599–621.
 - [99] Rutten (Eric) et Martinez (Florent). – SIGNALGTi: implementing task preemption and time intervals in the synchronous data flow language SIGNAL. In: *Proceedings of the 7th Euromicro Workshop on Real Time Systems, ERTS'95*, Odense, Denmark, June 14 - 16, 1995. pp. 176–183. – (IEEE Publ.).
 - [100] Rutten (Eric), Paoletti (Jean-Christophe), André (Guy) et Marcé (Lionel). – A task-level language for operator assistance in teleoperation. In: *Proceedings of the International Conference on Human Machine Interaction and Artificial Intelligence in Aeronautics and Space*, Toulouse, France, September 26–28, pp. 393–411.
 - [101] Rutten (Eric-Paul). – *Représentation en logique temporelle de plans d'actions dotés d'une structure de contrôle impérative ; application à l'assistance à l'opérateur en téléopération*. – Thèse de Doctorat en Informatique, IFSIC, Université de Rennes 1, 13 Juillet 1990.
 - [102] Sacres consortium. – *The common format of synchronous languages - The declarative code DC+ Version 1.4*. – Rapport technique, Esprit Project SACRES EP 20897, November 1997.
 - [103] Schneider (Stanley A.), Chen (Vincent W.), Pardo-Castellote (Gerardo) et Wang (Howard H.). – CONTROL-SHELL: a software architecture for complex electromechanical systems. *International Journal of Robotics Research* (special issue on Integrated Architecture for Robot Control and Programming), vol. 17, n4, avril 1998, pp. 360–380.
 - [104] Seshia (S.A.), Shyamasundar (R.K.), Bhattacharjee (A.K.) et Dhodapkar (S.D.). – A translation of statecharts to esterel. In: *Proceedings of the World Congress on Formal Methods, FM'99, Volume II, Toulouse, France, September 20-24, 1999*. pp. 983–1007. – Springer Verlag. LNCS nr. 1709.
 - [105] Sinoquet (Christine). – *Étude des liens entre les langages réactifs asynchrone et synchrone; application à ELECTRE et SIGNAL*. – Rapport de DEA en informatique, IFSIC, Université de Rennes 1, septembre 1995.
 - [106] Turro (Nicolas). – *MAESTRO: une approche formelle pour la programmation d'applications robotiques*. – Thèse de Doctorat en Informatique, Université de Nice-Sophia-Antipolis, septembre 1999.

Table des matières

1	Problématique	7
1.1	Contexte et motivation	7
1.2	Problématique	8
1.3	Approche suivie et premières contributions	11
1.3.1	Approche suivie	11
1.3.2	Séquencement de tâches externes	12
	<u>Activités en Doctorat</u>	15
	<u>Activités en programmation robotique</u>	16
	<u>Activités en planification</u>	18
1.3.3	Réseaux flots de données dynamiques.	18
	<u>Activités sur les flots de données dynamiques</u>	19
1.4	Vers le séquençement de tâches flot de données	21
2	Séquencement de tâches flot de données	23
2.1	Langages réactifs et synchrones	23
2.1.1	Principes	23
2.1.2	Langages synchrones	25
2.1.3	Approches du multi-formalisme dans les langages réactifs	26
	<u>Activités générales sur les langages réactifs et synchrones</u>	27
2.2	SIGNAL, GTi et la préemption	27
2.2.1	SIGNAL	27
	<u>Activités générales autour de SIGNAL</u>	33
2.2.2	SIGNALGTi	33
	<u>Activités sur GTi et la gestion de tâches en SIGNAL</u>	39
2.2.3	PAL : la préemption dans les langages réactifs	39
	<u>Activités sur la préemption</u>	40
2.3	Modélisation de langages de contrôle	40
2.3.1	STATEMATE : STATECHARTS et ACTIVITYCHARTS	40
	<u>Activités liées à STATEMATE</u>	46
2.3.2	GRAFCET et la norme IEC 1131-3	46
	<u>Activités liées à la norme IEC 1131</u>	51
2.4	Modélisation du séquençement de tâches flot de données	51
2.4.1	Bilan	51
2.4.2	Perspectives	51
2.4.3	Applications	52
3	Applications à des systèmes de contrôle/commande	53
3.1	Cellule de productique du FZI	53
3.1.1	La cellule de productique du FZI	54
3.1.2	Spécification du contrôleur en SIGNAL	55
3.1.3	Spécification et vérification des propriétés	57
	<u>Activités liées à l'application à la cellule de production</u>	58
3.2	Applications de GTi	58
3.2.1	Vision active en robotique	58
	<u>Activités liées à l'application de vision robotique</u>	63
3.2.2	Contrôle de transformateur	63

<i>Activités liées à l'application de contrôle de transformateur</i>	67
3.2.3 Animation comportementale	67
<i>Activités liées à l'application en animation</i>	69
3.3 Enseignements tirés et perspectives	69
3.3.1 Utilisation plus approfondie des technologies synchrones	70
3.3.2 Langages dédiés	70
4 Bilan et perspectives	71
4.1 Bilan	71
4.2 Perspectives	72
Bibliographie	74

Table des figures

1.1	Langage de planification : les treize relations temporelles entre intervalles de Allen	13
1.2	Langage de planification : une action primitive dans le temps.	14
1.3	Langage de planification : la séquence dans le temps.	14
1.4	Planification temporelle : les contraintes temporelles pour une action.	17
1.5	Planification temporelle : la contrainte pour les conditions, à la fois post- et précondition.	17
1.6	Flots de données dynamiques : code MINIFOLD pour un exemple à deux coordinateurs.	19
1.7	Flots de données dynamiques : sous-réseaux des coordinateurs C1 et C2.	20
1.8	Flots de données dynamiques : états globaux du réseau de l'application.	20
1.9	Flots de données dynamiques : automates pour les coordinateurs C1 et C2, et l'application.	20
2.1	SIGNAL : exemple de sur-échantillonnage : le processus décrémenteur d'entrées.	28
2.2	SIGNAL : trace du processus décrémenteur d'entrées.	28
2.3	SIGNAL : exemple de sur-échantillonnage : décompteurs en cascade.	29
2.4	SIGNAL et SIGALI : invariance (a) et invariance sous contrôle (b) de E depuis E_0	30
2.5	SIGNAL et SIGALI : Attractivité (a) et accessibilité (b) de F depuis E	30
2.6	SIGNAL : architecture de l'environnement de programmation.	31
2.7	SIGNAL : l'architecture du projet SACRES, et la position de STM2DC+.	32
2.8	GTi : phases dans le traitement de signal vocal.	34
2.9	GTi : subdivision de $[\alpha, \omega]$ en sous-intervalles.	34
2.10	GTi : tâche on l'intervalle I .	35
2.11	GTi : tâche each l'intervalle I .	35
2.12	GTi : système place/transition hiérarchique.	36
2.13	GTi : hiérarchie de tâches codant le système de places/transitions.	36
2.14	GTi : spécification en SIGNAL GTi du chronomètre.	36
2.15	GTi : trace pour un tâche de comptage, contrôlée par les événements b, s, r et e.	37
2.16	GTi : l'intervalle de temps en SIGNAL.	37
2.17	GTi : exemple de structuration hiérarchique du contrôle.	37
2.18	STATEMATE : un exemple de STATECHARTS dans MAGNUM.	41
2.19	STATEMATE : un exemple d'ACTIVITYCHARTS dans MAGNUM.	42
2.20	STATEMATE : structure hiérarchique générale du modèle.	44
2.21	Norme IEC 1131-3 : les quatre langages.	47
2.22	Norme IEC 1131-3 : un GRAFCET (SEQUENTIAL FUNCTION CHARTS) avec une action en ST.	49
3.1	Cellule de production : le modèle du FZI.	54
3.2	Cellule de production : spécification étagée, avec les comportements possibles, les comportements sûrs, et le scénario complet.	56
3.3	Cellule de production : spécification du contrôleur.	57
3.4	Vision robotique : processus SIGNAL de commande référencée vision.	60
3.5	Vision robotique : processus SIGNAL d'estimation en parallèle de la commande.	61
3.6	Vision robotique : une trace possible du séquençement de tâche.	61
3.7	Vision robotique : spécification du séquençement de tâche.	62
3.8	Vision robotique : l'environnement de reconstruction de scène 3D.	63
3.9	Transformateur : topologie de la station.	64
3.10	Transformateur : phase de confirmation : hiérarchie d'interruption.	65
3.11	Transformateur : simulation de la phase de confirmation.	66
3.12	Animation comportementale : simulation de conduite de véhicule	68