Spécification et implantation des systèmes distribués temps réel de contrôle et traitement de données

Nicolas Pernet, Yves Sorel
Projet AOSTE
INRIA Rocquencourt
8153 Le Chesnay, FRANCE
Email: Nicolas.Pernet, Yves.Sorel@inria.fr

***Abstract*— La combinaison des fonctionnalités de contrôle et de traitement de données dans les systèmes temps réel force les développeurs de tels systèmes à utiliser différents langages de spécification, chacun ayant diverses possibilités de vérification et de simulation. La plupart de ces systèmes étant aujourd'hui distribués, se pose ensuite le problème d'obtenir une implantation distribuée à partir d'un ensemble de spécifications. L'approche usuelle consiste à obtenir à partir d'une compilation mono-processeur un code issu de chaque spécification, à distribuer et ordonnancer "à la main" ces codes, puis à distribuer et ordonnancer les communications nécessaires à chaque code distribué. Cette approche est source d'erreur, rallonge la durée de développement du système et ne permet d'obtenir, au mieux, qu'une implantation distribuée correcte, c'est à dire satisfaisant les spécifications, à défaut d'être efficace, c'est à dire minimisant les ressources. Pour y remédier, nous proposons de distribuer efficacement toutes les fonctionnalités, qu'elles soient de contrôle ou de traitement de données, à l'aide de la méthodologie AAA et du logiciel SynDEx qui permet d'aider à la distribution et à l'ordonnancement puis d'obtenir des exécutifs distribués à parti de spécifications d'algorithme (fonctionnalités) et d'architecture. Pour ce faire, nous présentons comment différent types de langages peuvent être traduits dans le langage SynDEx.**

I. Introduction

Nos travaux de recherche sont dédiés aux systèmes temps réel distribués. Nous appelons "système" l'implantation de fonctionnalités, ou algorithmes, sur une architecture physique. De tels systèmes sont avant tout réactifs dans le sens où ils intéragissent continuellement avec leur environnement en acquérant des signaux d'entrée, en effectuant des calculs sur ces signaux et en produisant des signaux de sortie. Les systèmes temps réel sont, quant à eux, des systèmes réactifs pour lesquels des contraintes temporelles sont imposées entre deux occurrences successives d'un signal d'entrée (période) ou entre un signal d'entrée et un signal de sortie (latence). Les systèmes temps réels nous intéressant, sont souvent distribués et leurs algorithmes consistent à réaliser à la fois du contrôle et du traitement de données. De tels algorithmes sont habituellement décrits à l'aide de langages de spécification graphique.

Cet article présente les caractéristiques des algorithmes de contrôle et de traitement de données et comment ces caractéristiques influent sur le choix des langages de spécification. En effet, la partie contrôle et la partie traitement de données nécessitent des langages de spécification de types différents. Nous verrons ensuite que l'approche habituelle concernant la spécification de tels algorithmes, et consistant à combiner plusieurs langages, pose problème lors d'une implantation sur architecture distribuée.

Nous présenterons ensuite notre méthodologie AAA concrétisée par notre logiciel SynDEx. Nous expliquerons comment celui-ci constitue une solution pour le problème posé par l'approche usuelle, avant de donner des exemples de transformations de différents types de langages de spécification en langage de spécification SynDEx permettant une implantation distribuée efficace d'algorithmes de contrôle et de traitement de données.

II. Les systèmes de contrôle et de traitement de données

A. Aspects contrôle, aspects traitement e données

Les fonctionnalités des systèmes temps réel embarqués combinent les aspects contrôle et les aspects traitement de données. Les fonctionnalités de traitement de données consistent à du calcul massif sur des données, alors que le contrôle consiste à séquencer ces calculs en choisissant lequel effectuer parmi différentes alternatives. Il est important de noter que le contrôle n'implique pas la notion d'état. En effet les choix effectués par le contrôle peuvent l'être sans connaissance des calculs antérieurs. Le cas échéant, c'est l'obligation de mémoriser des résultats de calculs précédents qui introduit la notion d'état.

B. Langages de spécification

Les langages de spécification utilisés pour spécifier les systèmes de contrôle et de traitement de données sous forme graphique sont de deux types. Dans les langages flot de contrôle tout d'abord, un sommet est un état et un arc décrit une transition entre deux états, c'est à dire la condition nécessaire pour passer de l'un à l'autre. Les calculs peuvent être déclenchés par une transition ou un état.

Dans les langages flot de données, un sommet est une opération de calcul et un arc décrit une dépendance de données entre deux opérations. La façon dont le contrôle y est représentée diffère d'un langage flot de données à l'autre. Si la plupart utilise un sommet "retard"pour stocker l'état, certains utilisent une entrée booléenne sur une opération pour pouvoir choisir ou pas de l'exécuter, d'autres permettent de spécifier différents sous-graphes pour une même opération. Une entrée spéciale permet alors de choisir, à chaque occurrence de l'opération, le sous-graphe qui lui sera substitué.

Il est important de noter que dans le flot de contrôle, les arcs définissent un ordre total entre les états et donc les calculs qui leurs sont associés. D'autre part, les états et les données utilisées par les calculs sont des variables globales, accessibles par l'ensemble des états et des transitions du graphe. Or, ce concept de variable globale est impossible à maintenir sur une architecture distribuée. Enfin, les calculs n'y sont pas représentés sous forme de graphe.

Au contraire, dans un graphe flot de données, les arcs ne définissent qu'un ordre partiel entre les opérations. Tous les échanges de données entre opérations sont explicités, y compris les données servant au contrôle. Enfin le contrôle y est représenté comme les calculs sous forme de graphe.

Lorsqu'une implantation distribuée est souhaitée, il est donc préférable de partir d'une spécification flot de données :

- l'ordre partiel des graphes flot de données fait apparaître du parallélisme potentiel. En effet, deux opérations sans dépendances de données entre elles peuvent s'exécuter en même temps sur des opérateurs (processeurs) différents si l'architecture physique le permet :
- les dépendances de données étant exprimées, les problèmes d'accès concurrents aux données n'existent pas.

Néanmoins, le flot de contrôle garde l'avantage de permettre d'effectuer de la vérification en termes d'accessibilité d'un état ou de vivacité du graphe, par exemple.

C. Implantation distribuée : l'approche usuelle

La combinaison des fonctionnalités de contrôle et de traitement de données dans un même système conduisent les développeurs de tels système à utiliser plusieurs langages pour spécifier différentes fonctionnalités d'un même système. C'est le cas par exemple lorsque le langage flot de données Simulink est utilisé avec le langage flot de contrôle Stateflow [1]. Une fois les vérifications et simulations mono-processeur effectuées se pose le problème de l'implantation distribuée. En effet, bien que la spécification hétérogène [1] soit répandue, peu de langages s'intéresse à la distribution, ce que nous avions expliqué dans [2]. La plupart de ces langages permettent de générer du code mono-processeur fidèle à la spécification. Une spécification devient alors un code séquentiel, allouée par les développeurs à un opérateur ou processeur. Puisque certaines données sont communes ou échangées par les différentes spécifications, les développeurs doivent ensuite distribuer et ordonnancer les communications entre des opérateurs où des codes doivent échanger des données. De plus, dans les systèmes de contrôle et de traitement de données, les entrés sont des capteurs et les sorties des actionneurs. Or, avant même l'implantation, l'architecture des capteurs et des actionneurs est connue, les opérateurs auxquels chacun d'entre eux est connecté aussi. Le choix de placer un des code mono-processeur obtenus sur un des opérateurs est donc difficile, car il peut manipuler des entrées et des sorties dont les actionneurs et capteurs sont disséminés sur l'architecture. De plus, dans le cas de code obtenue à partir d'une spécification flot de contrôle, toutes les entrées ne sont pas nécessaires à chaque évaluation du graphe. En effet, en connaissant l'état courant, on peut réduire l'ensemble des entrées nécessaires à celle utiles pour évaluer les transitions sortantes par exemple. Un code mono-processeur centralisant le contrôle implique donc un surplus de communications inutiles provenant des différents capteurs.

L'implantation distribuée des fonctionnalités de contrôle et de traitement de données gagneraient donc à :

- utiliser la capacité du flot de données à exprimer le parallélisme potentiel des fonctionnalités d'un système pour en déduire automatiquement une implantation distribuée ;
- distribuer le contrôle de manière à l'effectuer au plus près de ces entrées et sorties pour minimiser les communications inutiles ;
- être déduite d'une spécification flot de données unique, comportant à la fois la description des fonctionnalités de contrôle et de traitement de données ;
- utiliser des transformation de langages, flot de contrôle ou non, décrivant le contrôle vers un langage flot de données

Afin de pallier à ces problèmes, nous présentons SynDEx un logiciel de spécification flot de données reposant sur notre méthodologie AAA. En effet celui-ci permet d'obtenir une implantation distribuée efficace à partir d'une spécification flot de données. Nous proposons aussi des exemples de transformation de langages permettant d'unifier les spécifications de différents langages incluant du contrôle en spécification SynDEx.

III. AAA/SynDEx

A. Méthodologie AAA

La méthodologie AAA [3] vise le prototypage rapide et l'implantation optimisée d'applications distribuées temps réel embarquées, telles celles rencontrées en contrôle-commande de systèmes complexes comprenant aussi du traitement du signal et des images. Elle est fondée sur des modèles de graphes, autant pour spécifier les Algorithmes applicatifs et les Architectures matérielles distribuée, que pour déduire les implantations possibles en termes de transformations de graphes. L'Adéquation revient à résoudre un problème d'optimisation consistant à choisir une implantation dont les performances, déduites des caractéristiques des composants matériels, respectent les contraintes temps réel et d'embarquabilité. Dans le cas du temps réel critique nous privilégions les approches statiques en introduisant le minium de décisions dynamiques, i.e. uniquement quand elles sont inévitables. Tout cela permet de générer automatiquement d'une part des exécutifs distribués temps réel à faible surcoût pour les composants processeurs, et d'autre part des "net-list" pour les composants circuits intégrés spécifiques. SynDEx est le logiciel concrétisant cette méthodologie. Afin de réduire le fossé entre la phase de spécification/simulation des automaticiens et la phase d'implantation en temps réel des informaticiens, et afin de minimiser la durée du cycle de développement des applications distribuées temps réel embarquées, nous étudions les liens entre des langages orientés métier et notre méthodologie AAA/SynDEx.

1. http://www.mathworks.com

B. Logiciel syndex

Grâce à SynDEx[2] , l'utilisateur spécifie deux graphes. Le graphe d'algorithme est un graphe flot de données représentant les fonctionnalités du système. Ce graphe est infiniment répété de part l'aspect réactif du système réactif qu'il représente. Toutes les opérations, les noeuds du graphe, doivent être exécutées avant que ne débute la prochaine répétition infinie du graphe. Chaque opération possède des ports d'entrée et/ou des ports de sortie et chaque arc relie un port de sortie à un ou plusieurs (diffusion) port d'entrée. Chaque opération ne peut être exécutée que lorsque toutes ses entrées sont disponibles. Un sommet spécifique appelé "delay" est nécessaire si une opération a besoin d'une donnée produite lors d'une précédente répétition infinie du graphe. Un sommet peut être décrit par un sous-graphe, rendant la spécification hiérarchique possible.

Le graphe d'architecture est un graphe orienté où les sommet sont des opérateurs (microcontrôleur, DSP, FPGA) ou des média de communication (ethernet, CAN, RS232) et les arcs sont des connections entre eux. Après avoir spécifier ces deux graphes, l'utilisateur doit définir la durée de chaque opération (respectivement de chaque dépendance de donnée) sur chaque opérateur (respectivement sur chaque media). De plus, l'utilisateur peut spécifier des contraintes de distribution, par exemple qu'une opération d'entrée doit s'exécuter sur tel opérateur car la réalité physique est que le capteur y est relié.

Finalement l'adéquation, basée sur des heuristiques d'optimisation [4] de distribution et d'ordonnancement, peut être réalisée et permet d'obtenir un diagramme temporel simulant l'exécution temps réel. Ce diagramme est un graphe représentant le résultat de la distribution et de l'ordonnancement temps réel de l'algorithme sur l'architecture. Ce résultat peut être utilisé pour générer des exécutifs multi-processeur. Chaque exécutif permet de réaliser l'ordonnancement des opérations allouées à un processeur par l'adéquation, mais initialise aussi les média, gère les communications et les synchronisations entre les opérateurs. Ces exécutifs ne nécessitent donc pas de système d'exploitation temps réel résident sur les opérateurs. Néanmoins, à partir du résultat de l'adéquation il est aussi possible de configurer automatiquement des systèmes d'exploitation temps réel comme OSEK, RTAI ou autres.

Des utilisations industrielles de SynDEx existent dans l'aéronautique, l'automobile et la robotique.

C. Contrôle dans SynDEx

Pour permettre la spécification du contrôle, le conditionnement a été introduit dans SynDEx. Un sommet peut ainsi être décrit par plusieurs sous-graphes. Un tel sommet possède un port d'entrée, dit de conditionnement, qui lors de chaque répétition infinie, consomme une donnée et, en fonction de sa valeur, choisit lequel des sous-graphes alternatifs elle exécutera. Le conditionnement permet donc d'expliciter la notion de choix nécessaire au contrôle. Pour représenter la notion d'état, le conditionnement doit être combiné à un sommet "delay". Cela permet ainsi de représenter l'équivalent flot de données d'une spécification flot de contrôle.

2. http://www.syndex.org

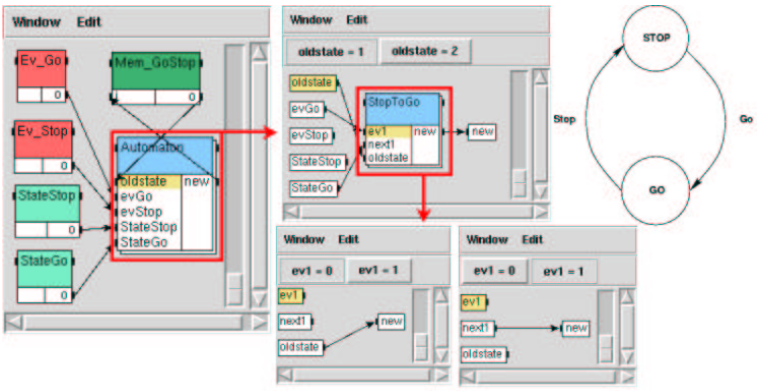

FIG. 1 –. *Graphe flot de contrôle avec son équivalent SynDEx*

La figure 1 montre, complétement à droite, un graphe flot de contrôle comportant 2 états et 2 transitions. A sa gauche, l'équivalent en flot de données SynDEx est donnée à titre comparatif. Les états y sont représentés par un seul sommet delay et ce, quelque soit leur nombre (dans le cas d'un graphe flot de contrôle non-hiérarchique). Par contre à chaque état correspond un sous-graphe différent d'un premier niveau de conditionnement. Ensuite pour un état et donc un sous-graphe donné, chaque transition sortante implique un niveau de conditionnement supplémentaire. Ainsi, le graphe flot de données comporte plus de sommets et d'arcs que son équivalent en flot de contrôle. Mais cela vient essentiellement du fait que toutes les dépendances de données sont explicitées. Néanmoins, dire que les langages flot de contrôle sont inutiles parce qu'un langage flot de données comme SynDEx permet de spécifier l'équivalent n'est pas raisonnable tant le surplus d'arcs et de sommets est important.. Par contre, si une transformation automatique permet de passer d'une spécification flot de contrôle où l'utilisateur ne décrit que le contrôle, les dépendances de données restant implicites, à une spécification flot de données où ces dernières sont rendus explicites, l'efficacité de l'implantation distribuée n'en sera que meilleure.

IV. TRANSFORMATIONS DE SPÉCIFICATION INCLUANT DU CONTRÔLE EN GRAPHE SYNDEX

Nous présentons ici deux transformations de langages permettant de spécifier du contrôle en langage flot de données SynDEx. L'un des langages de départ est flot de contrôle, l'autre est flot de données.

A. Exemple SyncCharts

SyncCharts [5] est un langage flot de contrôle, proche des Statechart [6] de David Harel, mais qui en corrige les lacunes, notamment au niveau du déterminisme. Ce langage permet la description hiérarchique, un état pouvant contenir un sous-graphe, et la composition, plusieurs graphes pouvant s'exécuter en parallèle.

La figure 2 présente un graphe SyncCharts. Le graphe ABRO comporte un état ABO qui est décrit par un graphe qui comporte deux états dont un, AB, est décrit par la composition de deux graphes possédant chacun deux états. Son comportement

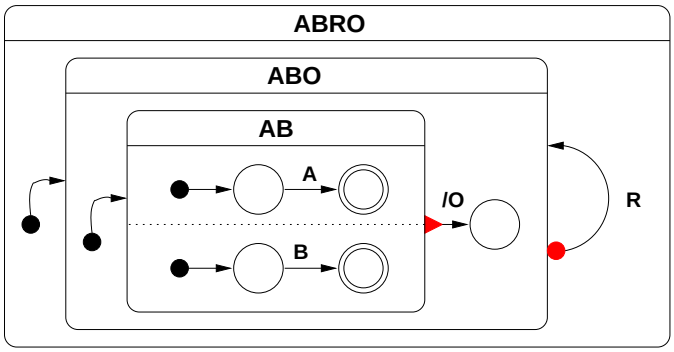

FIG. 2 –. *Exemple de graphe SyncCharts : ABRO*

est le suivant. Le système attend une occurrence de A et de B. Lorsque celles-ci ont eu lieu, O est émit. Toute occurrence de R ré-initialise le système.

La figure 3 montre la spécification SynDEx obtenue par transformation automatique du graphe SyncCharts ABRO. Les quatre sommets “delay” correspondent aux état de chacun des quatre graphes (ABRO, ABO, A et B), quatre sommets conditionnés décrivent, pour chaque graphe, les transitions concernant chacun des états du graphe. Enfin les deux sommets de droite spécifient l’émission de O.

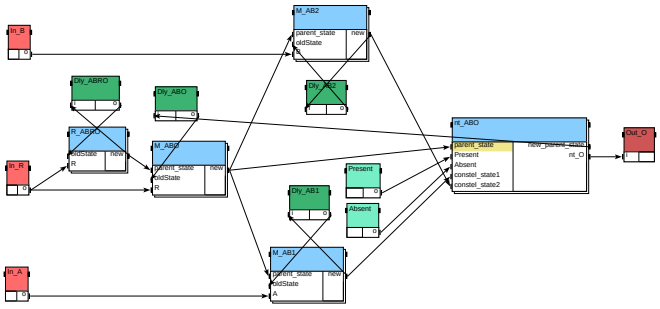

FIG. 3 –. *ABRO après transformation en graphe SynDEx*

B. *Exemple Scicos*

Scicos[3] [7] est une boîte à outil de Scilab qui permet de spécifier sous forme de flot de données et d’en faire la simulation. Ce langage est une alternative à Simulink. Dans les graphes flot de données Scicos, le contrôle est spécifié à l’aide de port d’entrée permettant, via une donnée booléenne, d’activer ou pas l’opération. Cette forme de contrôle est différente de notre approche dans SynDEx mais on peut, avec un sommet conditionné, représenter dans SynDEx l’équivalent d’un sommet de Scicos avec entrée d’activation. La difficulté est d’avantage de représenter les sommets scicos manipulant et générant les données booléennes servant aux activations.

La figure 4 montre une spécification Scicos comportant des sommets à entrées d’activation et des sommets manipulant et générant des données booléennes servant à ces activations.

3. http://www.scicos.org

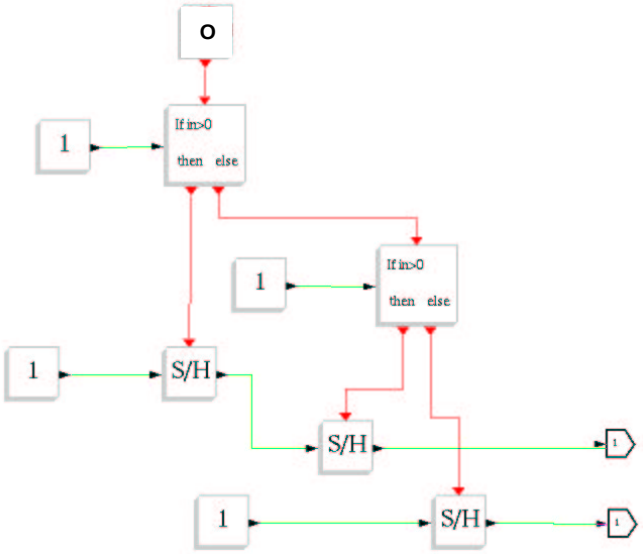

FIG. 4 –. *Spécification Scicos*

Afin de montrer que plusieurs transformations peuvent coopérer, cette spécification manipule une donnée d’entrée O qui une sortie de notre graphe SyncCharts ABRO. On peut voir l’usage de SyncCharts avec Scicos comme une alternative “logiciel libre” à l’utilisation de Stateflow avec Simulink.

La figure 5 montre le graphe SynDEx obtenu après la transformation du graphe SyncCharts ABRO et de la spécification Scicos. L’utilisateur peut alors, après avoir connecté la sortie O de la partie SyncCharts à l’entrée O de la partie Scicos :

- spécifier une architecture ;
- spécifier des contraintes de distribution inhérentes à la réalité physique de l’architecture ;
- réaliser différentes adéquation avec différentes architecture ;
- choisir de générer des exécutifs pour une implantation obtenue par l’adéquation.

Il n’a plus aucune décision à prendre sur l’ordonnancement et la distribution des communications. De plus il est sûr que tout le parallélisme potentiel de ses spécifications a pu été exploité.

V. CONCLUSION

Nous avons vu combien obtenir une implantation distribuée posait problème dans le cas de système de contrôle et de traitement de données. En effet, l’utilisation de plusieurs langages de spécification, amène les développeurs à devoir eux-même choisir les opérateurs devant exécuter chaque code obtenu par génération de code mono-processeur. Ensuite, les développeurs doivent distribuer et ordonnancer les communications entre ces opérateurs, sans aucune garantie de la cohérence de l’ensemble. Cette approche est source d’erreur, donc d’un allongement de la durée de développement de tels systèmes. De plus, elle ne permet d’obtenir, au mieux, qu’une implantation distribuée correcte, c’est à dire satisfaisant les spécifications, à défaut d’être efficace, c’est à dire minimisant les ressources (taux d’utilisation des media, nombre d’opérateur). Pour que

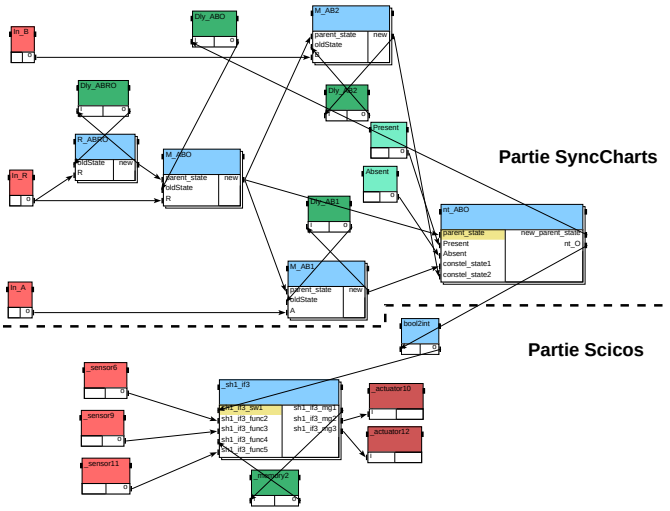

FIG. 5 –. *Graphe SynDEx obtenu par transformation des graphes SyncCharts et Scicos*

l'implantation soit efficace il faut d'une part que l'ensemble des dépendances de données soient explicitées, d'autre part que les spécifications expriment tout le parallélisme potentiel des fonctionnalités du système. Cela ne peut être réaliser qu'à partir d'une spécification flot de données. Nous avons donc présentés SynDEx, un logiciel issu de la méthodologie AAA. Celui-ci permet d'obtenir, à partir d'un graphe flot de données des fonctionnalités du système et d'un graphe d'architecture, une distribution et un ordonnancement optimisé des fonctionnalités sur l'architectures. Afin d'unifier les spécifications en une seuls spécification SynDEx, nous recommandons d'utiliser des transformations automatiques comme celles de SyncCharts vers SynDEx et Scicos vers SynDEx que nous avons proposées. Enfin, l'emploi de SyncCharts, Scicos et SynDEx permet de profiter de la vérification sur les aspects contrôle (SyncCharts), de faire de la simulation sur la partie traitement de données (Scicos) et d'obtenir une implantation distribuée efficace de l'ensemble (SynDEx). Ceci raccourci la durée de développement et minimise les ressources nécessaires.

REFERENCES

[1] X. Liu, J. Liu, J. Eker, and E. A. Lee. Heterogeneous modeling and design of control systems. In *Software-Enabled Control: Information Technology for Dynamical Systems*. Wiley-IEEE Press, Tariq Samad and Gary Balas edition, April 2003.

[2] N. Pernet and Y. Sorel. Optimized implementation of distributed real-time embedded systems mixing control and data processing. In *Proceedings of the ISCA 16th International Conference: Computer Applications in Industry and Engineering (CAINE-2003)*, Las Vegas Nv, Nov. 11-13 2003.

[3] Y. Sorel. Massively parallel systems with real time constraints, the algorithm architecture adequation methodology. In *Proceedings of Conf. on Massively Parallel Computing Systems*, Ischia, Italy, May 1994.

[4] T. Grandpierre, C. Lavarenne, and Y. Sorel. Optimized rapid prototyping for real-time embedded heterogeneous multiprocessors. In *CODES'99 7th International Workshop on Hardware/Software Co-Design*, Rome, May 1999.

[5] André Charles. Computing synccharts reactions. In *Synchronous Languages Applications and Programming*, Porto, Portugal, July 2003.

[6] D. Harel. Statecharts: a visual formalism for complex systems. In *Science of Computer Programming*, volume 8, pages 231–274, 1987.

[7] Ramine Nikoukhah and Serge Steer. Scicos - a dynamic system builder and simulator user's guide.