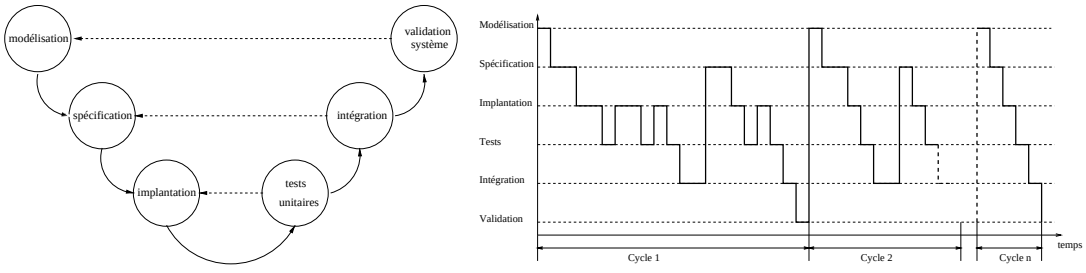
De la modélisation à la réalisation : réduction du cycle de développement des applications temps réel distribuées

Rémy KOCIK, Yves SOREL
INRIA Rocquencourt - Domaine de Voluceau BP105
78153 Le Chesnay CEDEX - France
Tel: 33-(1) 39 63 52 60 - Fax: 33-(1) 39 63 57 86
email: remy.kocik@inria.fr, yves.sorel@inria.fr

1 Cycle de développement d'un système automatisé

La conception d'un système automatisé pose un grand nombre de problèmes matériels et logiciels. Elle nécessite l'implantation d'algorithmes complexes sur des architectures matérielles souvent distribuées et hétérogènes. Afin de gérer au mieux cette complexité les méthodes de développement employées dans les projets sont généralement basées sur un principe de hiérarchisation permettant de décrire le système comme un ensemble de sous-systèmes plus simples et donc plus faciles à concevoir. Ce principe est appliqué dans le cycle de développement le plus utilisé actuellement pour la conception des aspects matériels et logiciels : le cycle en V. Il permet selon une hiérarchisation descendante par étape, d'aboutir à la conception détaillée d'une application à partir d'une description abstraite [11][14].

On peut distinguer trois grandes étapes : la modélisation, la spécification et l'implantation. A ces trois étapes correspondent respectivement trois étapes de validation : les tests unitaires, l'intégration et la validation du système(fig.1(a)). Dans ce document, nous nous intéresserons principalement à la conception de la partie logicielle qui se trouve, comme la partie matérielle, en aval de la modélisation qui leur est commune.



(a) Cycle en V

(b) Itérations du cycle

FIG. 1 – *Cycle de développement d'un système automatisé*

La *modélisation* consiste à déterminer et à décrire sous forme d'équations mathématiques,

le comportement que devra avoir le système. La *spécification* du logiciel est une description "haut niveau" du logiciel. Elle permet de décrire les algorithmes et les contraintes d'implantation qui concrétisent de manière logicielle les équations mathématiques définies par la modélisation. L'*implantation* consiste à écrire les programmes qui doivent exécuter la spécification sur l'architecture matérielle. La première étape de validation rencontrée dans le cycle de développement est l'étape de *tests unitaires*. Elle est souvent longue et fastidieuse. Il s'agit ici de s'assurer que toutes les fonctions qui implantent les algorithmes fonctionnent correctement. Lorsque toutes les fonctions ont ainsi été déboguées et validées séparément, on vérifie lors de l'*intégration* que l'ensemble des fonctions se comporte comme défini par le modèle. Si ce n'est pas le cas, la spécification est remise en cause. La dernière étape, la *validation système*, consiste à vérifier que le comportement de l'application est conforme au cahier des charges. Si cette étape ne valide pas le système, le modèle est remis en cause, il doit être affiné.

Le cycle de développement du logiciel peut être découpé en deux grandes parties en fonction des compétences requises pour mener à bien chacune des étapes qui les compose. On peut ainsi distinguer une partie modélisation système qui regroupe l'étape de modélisation et de validation qui lui est associée. Cette première partie, la plus en amont dans le cycle, nécessite une bonne connaissance des outils mathématiques fournis par la théorie de l'Automatique. La deuxième partie regroupe toutes les autres étapes du cycle, c'est la réalisation proprement dite du logiciel (spécification, implantation). Elle demande de la part des ingénieurs de bonnes connaissances en informatique temps réel.

2 Ruptures dans le cycle

Dans le cycle de développement d'un système automatisé, c'est au moment de la spécification qu'une collaboration s'établie entre les automaticiens qui définissent le modèle du système et les informaticiens qui sont chargés de sa réalisation. C'est l'étape de conception charnière qui consiste à traduire le modèle mathématique en un modèle informatique. La problématique que pose la réalisation de ces deux modèles est différente, ce qui a pour conséquence l'introduction d'erreurs dues à des incompréhensions entre les intervenants de ces deux disciplines. Ces erreurs se répercutent jusqu'à l'implantation et ne sont détectées que lors de la validation, ce qui rallonge considérablement les temps de développement.

C'est pourquoi, dans ce document, nous décrivons dans un premier temps, avec la terminologie employée en Automatique, la problématique rencontrée par l'automaticien lors de la modélisation. Dans un deuxième temps nous exposons, avec la terminologie issue de l'Informatique temps réel, la problématique liée à l'implantation logicielle du modèle défini par l'automaticien en essayant autant que possible de définir les liens qui existent entre les deux terminologies. Nous espérons, par ce biais, contribuer à la réduction du fossé qui existe entre les deux communautés.

Le cycle en V, grâce à la hiérarchisation des problèmes à résoudre, a montré son efficacité dans la réalisation de nombreuses applications. Néanmoins, la complexité croissante de celles-ci induit un coût de développement ayant un impact important sur le coût du produit final. Le temps consacré par les ingénieurs et techniciens à l'écriture des lignes de code et à leur déboguage représente la part la plus importante du budget consacré à la réalisation d'une application. Afin de garantir la compétitivité de telles applications, il est nécessaire de minimiser les temps de développement, c'est-à-dire de réduire la durée de chaque étape et surtout le nombre de réitérations du cycle complet (Fig.1(b)).

Les outils de développement basés sur des méthodes formelles[6] ont beaucoup apporté dans ce sens. En effet, l'intérêt de ces méthodes est de reposer sur des langages de spécification ri-

goureux fondés sur un ensemble de règles mathématiques qui autorisent des vérifications et la génération automatique d'un code sain conforme à la spécification : le nombre d'itérations du cycle en V est réduit car la vérification permet de détecter rapidement des erreurs de spécification ; la génération automatique d'un code conforme à la spécification vérifiée, réduit considérablement les tests unitaires. La spécification ayant été vérifiée, l'intégration n'est plus nécessaire.

Dans ce contexte, la méthodologie AAA [1][15] basée sur un formalisme de graphes a été développée pour réaliser des implantations optimisées d'algorithmes temps réel sur des architectures distribuées. Le logiciel de CAO niveau système SynDEx [2][5] qui implante cette méthodologie a permis, entre autre, la réalisation rapide d'un prototype de véhicule électrique autonome soumis à de fortes contraintes de coût, le Cycab[8]. Cette expérience nous a confirmé les bénéfices que de telles méthodes peuvent apporter au développement d'applications temps réel complexes.

Néanmoins, ces méthodes ne prennent en compte que la partie réalisation du cycle en V, il existe donc encore une rupture entre la modélisation et la spécification d'autant plus gênante qu'elle est susceptible d'introduire des erreurs dès le début du cycle de développement. Dans les projets industriels, ce passage de la modélisation à la spécification repose généralement sur le savoir faire de quelques personnes, voire d'une personne unique. Lors de la modélisation et du passage à la spécification, il est pourtant nécessaire d'être particulièrement vigilant car les erreurs introduites à ce niveau se répercutent tout au long du cycle et ne sont souvent détectées que lors de la phase de validation ce qui nécessitera la réitération complète du cycle pour les corriger. Les principales erreurs qu'on peut recenser à ce niveau sont d'une part les erreurs dans le modèle des lois de commande et d'autre part les erreurs introduites lors de la traduction manuelle du modèle à la spécification.

Pour réduire ces risques d'erreurs, pour améliorer la traçabilité et diminuer le nombre d'itérations du cycle de développement, il est nécessaire d'apporter des solutions homologues à celles apportées par les langages formels pour passer de la spécification à l'implantation : simulation pour vérifier le modèle et passage automatique à la spécification. Dans la dernière partie du document, nous montrons comment l'interfaçage entre Scicos (outil de simulation hybride) et SynDEx œuvre dans ce sens.

3 Modélisation du système automatisé

L'automaticien cherche à maîtriser, en fonction d'un but défini, l'évolution dans le temps d'un système matériel appelé *processus* (ou *procédé*) constitué d'éléments mécaniques, électriques et/ou chimiques liés entre eux. Son travail consiste, dans une première phase de modélisation, à décrire les interactions entre les composants qui constituent le processus sous la forme d'équations mathématiques. Le *modèle de processus* qu'il construit ainsi, lui permet de prédire l'évolution de celui-ci lorsque ses composants sont soumis à différents phénomènes physiques (ou *actions*) externes. Ce modèle est la base mathématique définissant l'ensemble des actions, appelées *lois de commande*, base sur laquelle est conçue un système matériel, le *système de commande*, dont la fonction est de réaliser physiquement les commandes qui permettront de maîtriser l'évolution du processus. Le processus (système dont on cherche à maîtriser l'évolution) et le système de commande (système qui doit agir sur le processus) forment ensemble le *système automatisé* que l'on cherche à concevoir.

Le système de commande peut être conçu comme un ensemble de sous-systèmes, chacun d'eux construit sur la base de lois de commande plus simples [1]. La loi de commande la plus

1. Adéquation Algorithme Architecture
2. Synchronized Distributed Executive

simple décrit un système monovariable. Son entrée, appelée *consigne*, représente la valeur que l'on désire qu'un paramètre physique du processus atteigne. La sortie du système est la *commande*, elle représente l'ampleur d'une action physique appliquée au processus pour que le paramètre physique du processus atteigne la valeur de la consigne.

L'automaticien utilise autant que possible des systèmes bouclés qui autorisent une commande plus fine du processus. Ils permettent, d'une part de corriger, grâce à l'observation de l'évolution du processus (*feedback*) l'erreur entre le processus réel et son modèle (*erreur*), d'autre part, ils permettent de prendre en compte l'influence de perturbations externes sur l'évolution du processus. En contrepartie, ils imposent de réaliser une chaîne de retour d'observation du processus.

4 Réalisation des lois de commande à l'aide d'un calculateur

4.1 Loi de commande numérique

Lorsque le processus que l'on cherche à maîtriser est complexe, il est intéressant, voire même nécessaire, d'utiliser un système à calculateur pour mettre en œuvre les lois de commande. Cette solution numérique apporte des gains en termes de coût, de productivité et de performance[13].

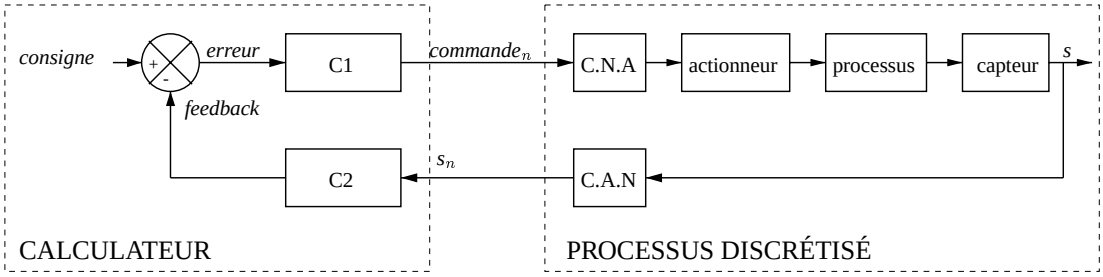


FIG. 2 – *Système de commande bouclé à calculateur*

L'approche généralement utilisée pour réaliser la loi de commande avec un calculateur consiste à discrétiser le processus[9]. Ainsi les interactions calculateur-processus sont concrétisées par des *transducteurs*. Dans les systèmes qui nous intéressent les signaux de *feedback* sont fournis par des transducteurs de type *capteur* qui permettent de mesurer l'ampleur d'un phénomène physique en la convertissant en signal électrique discrétisé par un *convertisseur analogique-numérique (C.A.N)*. Le signal de *commande* calculé est converti en signal analogique par un *convertisseur numérique analogique* (C.N.A) avant d'être appliqué à un transducteur de type *actionneur* dont le rôle est de convertir un signal électrique en un autre phénomène physique capable d'agir sur le processus (Fig.2)[1]. La loi de commande elle même, qui calcule le signal de *commande* à partir de la *consigne* et de la sortie *s* qu'on cherche à maîtriser, est réalisée par un programme informatique qui s'exécute sur un calculateur. Ce système informatique, considéré par l'automaticien comme un simple maillon du système automatisé constitue l'unique centre d'intérêt de l'informaticien.

De son point de vue, l'informaticien doit concevoir une *application* composée d'un *système informatique* et d'un *environnement* physique avec lequel il interagit. Le système est composé d'un *calculateur* et d'un ensemble de programmes qu'il doit exécuter appelé *logiciel*. L'environnement est défini comme l'ensemble de tous les éléments physiques qui sont extérieurs au système informatique et qui sont en interaction avec lui. La frontière entre l'environnement et le système informatique est parfois difficile à établir. C'est pourquoi nous avons choisi ici de considérer que l'ensemble des composants physiques qui peuvent être programmés (microprocesseurs, bus de communication, mémoire, interface E/S, etc.) fait partie du système informatique, alors que tous

les composants physiques non-programmables constituent l'environnement représentant du point de vue de l'automaticien, l'ensemble des éléments qui constituent le processus discrétisé à l'exception des convertisseurs analogique-numérique et numérique-analogique (cf. figure 2).

Le rôle du système informatique est de s'assurer que l'évolution du processus suit toujours les lois définies par l'automaticien lors de la modélisation. Pour cela, il doit constamment être en interaction avec l'environnement de manière à détecter ses changements d'états (*événements d'entrée*), et à réagir en réalisant des opérations de calcul pour produire un changement d'état (*événements de sortie*) de l'environnement afin de contrôler son comportement. C'est pourquoi ce type de système est appelé *système réactif* [7]. L'environnement est parfois aussi appelé *système contrôlé*. La validité d'une action dépend généralement du temps qui s'écoule entre la réalisation de cette action et l'événement d'entrée qui l'a déclenchée. En d'autres termes, sur ce type de systèmes, il est nécessaire que les réactions du système aux événements soient effectuées en un temps inférieur à une borne maximale. Les applications soumises à ce type de contraintes sont qualifiées de *temps réel* et les bornes maximales sur le temps de réalisation des actions sont appelées *contraintes temps réel*.

Suivant la nature de l'application, il est possible de tolérer de temps en temps, et avec une certaine marge, le non respect de certaines contraintes. Ce type de contraintes est appelé *contraintes relatives* en opposition aux contraintes dites *strictes* qui doivent impérativement être respectées[3]. Le non respect de contraintes strictes peut avoir des conséquences catastrophiques sur l'intégrité du système et de l'environnement. Toute la difficulté est de concevoir un système temps réel *prédictible* de manière à pouvoir garantir que les contraintes strictes seront toujours satisfaites et que le dépassement des contraintes relatives restera borné et occasionnel[10].

4.2 Spécification logicielle

La spécification est une description haut niveau des aspects logiciels et matériels du système sous une forme simplifiée sans en donner les détails[14]. Dans un système temps réel, les aspects matériel et logiciel sont fortement liés. Pour réaliser le logiciel il est nécessaire de connaître non seulement les algorithmes applicatifs (par opposition aux algorithmes systèmes par exemple d'ordonnancement) ainsi que les contraintes temporelles sur l'exécution de ces algorithmes, mais il est aussi nécessaire de connaître l'architecture matérielle afin de générer du code exécutable adapté à l'architecture (jeu d'instructions des processeurs, accès capteurs et actionneurs, protocoles des médias de communication, etc.). La spécification sert de point de départ à la conception du logiciel. Elle doit donc décrire à un haut niveau les algorithmes, l'architecture matérielle et les contraintes temporelles d'exécution des algorithmes sur le calculateur.

4.2.1 Contraintes temporelles

La discrétisation du processus et l'implantation numérique des lois de commande n'est pas sans conséquence sur la modélisation de ces dernières. L'automaticien doit choisir des fréquences d'échantillonnage adéquates pour chacune des entrées et sorties du système (*commande, s* et *consigne*). En règle générale pour simplifier les calculs complexes induits par le passage d'un espace mathématique continu à un espace mathématique discret, l'automaticien fait l'hypothèse que les signaux sont échantillonnés périodiquement. Pour la même raison, il suppose que les entrées et les sorties de la loi de commande qu'il considère sont échantillonnées en même temps[13]. En pratique, l'automaticien choisit une fréquence bien supérieure à $2F_c$ (souvent de l'ordre de 8 à 10 fois F_c). La lecture de différents ouvrages sur la théorie de l'automatique montre que ce choix est souvent empirique et repose généralement sur le savoir faire de l'automaticien. Néanmoins, on trouve

quelques règles théoriques ; elles montrent toutes que le choix de la fréquence d'échantillonnage est lié à la constante de temps principale ($\tau = \frac{1}{2\pi F_c}$) du processus à commander et que ce choix n'est pas unique puisque ces règles définissent une plage de valeurs acceptables pour F_e[9][13]. Le choix des fréquences d'échantillonnage se traduit lors de l'implantation par des contraintes sur le rythme d'entrée des événements du système temps réel. Ce type de contrainte est appelé contraintes de *cadence*.

Le système temps réel est aussi soumis à des contraintes de *latence*, qui imposent une borne sur l'intervalle de temps qui s'écoule entre l'arrivée d'un événement d'entrée et la production de l'événement de sortie correspondant. Cette contrainte a pour but de garantir le *temps de réponse* du système qui est un critère important de qualité. Il a été montré que sous certaines hypothèses simplificatrices ce temps de réponse est proportionnel à la latence de calcul, additionné d'un retard $\frac{T_e}{2}$ dû aux opérations d'échantillonnage réalisées à la fréquence $\frac{1}{T_e}$[1].

4.2.2 Contraintes algorithmiques

Pour prendre en compte l'aspect infiniment répétitif des interactions entre le système temps réel et l'environnement (le nombre d'interactions ne peut être borné, il peut donc être considéré comme infini), il est nécessaire d'étendre la notion classique d'*algorithme*, séquence finie d'opérations réalisables en un temps fini sur un support matériel fini. Ainsi, dans le cadre de la conception des applications temps réel un algorithme est une séquence finie d'opérations, réalisables en un temps fini sur un support matériel fini, répétée infiniment, et dont l'exécution est déclenchée par les événements d'entrée. Dans un système temps réel complexe deux types d'algorithmes applicatifs distincts sont à prendre en compte : la *commande* et le *contrôle*. Les algorithmes de commande décrivent les actions qui doivent être appliquées au processus. Ils sont constitués de fonctions de transformations des signaux analogiques échantillonnés définies par les lois de commande (correcteur PID, filtres numériques, etc.). Les algorithmes de contrôle déterminent, en fonction d'événements d'entrées, quelles sont les actions à appliquer au processus à un instant donné.

4.2.3 Contraintes matérielles

Les applications embarquées, et de manière plus générale, les applications industrielles sont soumises à de fortes contraintes de coût (coût financier, encombrement, consommation d'énergie, etc.). Pour satisfaire ces contraintes, de nouveaux types d'architectures matérielles ont vu le jour. C'est dans le domaine de l'automobile, secteur à très forte compétitivité, que les avancées technologiques ont le plus contribué à cette évolution : réduction des câblages en rapprochant les calculateurs des transducteurs et multiplexage des données. Ceci a abouti à des architectures distribuées hétérogènes où la réutilisation de composants courants et peu coûteux est la règle d'or. Les calculateurs sont architecturés autour de microcontrôleurs nécessitant peu de composants externes pour gérer les capteurs et les actionneurs, d'ASIC et de FPGA pour réaliser des fonctions spécifiques de bas niveau évoluant peu dans la vie du produit. Les communications inter-processeurs sont véhiculées par des bus de terrain faible coût (CAN, VAN, K-Bus, FIP, etc.) adaptés aux environnement perturbés.

5 Réduction du cycle de développement

Pour prendre en compte efficacement toutes les contraintes d'implantation, pour éviter les ruptures entre les différentes étapes du cycle en V et ainsi réduire les coûts de développement,

nous proposons une méthodologie basée sur le rapprochement de l'outil de modélisation et de simulation hybride Scicos et l'outil de spécification et d'implantation distribuée SynDEx.

5.1 Simulation hybride

Dans les systèmes numériques de commande à calculateur, les opérations d'échantillonnage et de quantification nécessaires à la discrétisation des signaux analogiques peuvent apporter des erreurs qui tendent à dégrader les performances du système. Ainsi, pour un niveau de performance équivalent, si on veut tenir compte de ces erreurs, il est plus complexe de concevoir des lois de commande pour un système numérique que pour un système analogique. La modélisation de systèmes discrets est beaucoup moins facile à réaliser que la modélisation des systèmes continus pour lesquels la théorie de l'automatique fournit un ensemble de méthodes et d'outils mathématiques beaucoup plus important. C'est pourquoi, afin de simplifier les calculs, l'automaticien cherche à utiliser un modèle *pseudo-continu* de son système de commande plutôt qu'un modèle discret[1]. La simulation hybride qui consiste à simuler l'ensemble du système automatisé (simulation d'un processus continu relié au système temps réel discret) est indispensable pour valider le modèle à partir duquel le système temps réel doit être conçu. Elle permet de paramétrer les lois de commande afin de compenser les approximations réalisées par la modélisation pseudo-continu.

Scicos[12] a été conçu dans ce but, il permet de réaliser une simulation hybride prenant en compte les fréquences d'échantillonnage des signaux continus.

5.2 Spécification du modèle et implantation optimisée des algorithmes

Les lois de commande discrètes et le modèle continu de processus sont décrits dans Scicos à l'aide de graphes flot de données proches des schémas-blocs bien connus des automaticiens. Après avoir extrait la partie concernant les lois de commande discrètes, un traducteur permet de transcrire celle-ci en un graphe SynDEx[2]. La spécification de l'algorithme est ainsi conforme au modèle, il n'est plus nécessaire de passer du temps à spécifier manuellement les lois de commande, seule la spécification de l'architecture matérielle reste nécessaire. SynDEx permet de faire cette spécification du matériel utilisé et propose ensuite une Adéquation (implantation optimisée de l'algorithme sur l'architecture), et peut ensuite générer automatiquement le code correspondant. Ainsi, l'interfaçage entre Scicos et SynDEx permet de couvrir l'ensemble du cycle de développement.

5.3 Simulation hybride avec prise en compte des retards

La simulation hybride, pour être la plus proche de la réalité, doit nécessairement prendre en compte les retards introduits dans les lois de commande par les temps d'exécution des calculs (latence) qui implantent ces lois, retards pouvant provoquer l'instabilité du système. Ceux-ci dépendent de l'implantation réalisée et notamment pour une architecture donnée de l'ordonnancement (allocation temporelle de la ressource CPU) choisi pour l'exécution des calculs associés à chacun des blocs qui composent le graphe décrivant les lois de commande.

Dans les approches classiques d'implantation, le choix de cet ordonnancement est réalisé à l'exécution (ordonnancement dynamique) par un RTOS (système d'exploitation temps réel) en fonction de critères (fréquence d'exécution, priorité, urgence etc.) définis lors de la spécification des blocs (appelés tâches lors de la spécification) par l'informaticien[4]. Dans ce type d'implantation il difficile de connaître a priori les latences de calculs responsables des retards qui ne peuvent donc pas être pris en compte à la simulation.

L'interfaçage entre Scicos et l'outil d'implantation SynDEx permet de résoudre ce problème. En effet, SynDEx réalise une adéquation qui consiste à choisir une distribution (allocation spatiale des ressources CPU) et un ordonnancement optimisé des blocs (appelés dans ce contexte *opération*) qui constituent l'algorithme sur les processeurs de l'architecture matérielle. Grâce à une bonne connaissance de l'architecture (caractéristiques des médias de communications et des processeurs) et à la connaissance des temps d'exécution de chaque opération, SynDEx réalise une prédiction des temps d'exécution de l'ordonnancement choisi. Cette prédiction qui permet de vérifier si les contraintes temps réel sont satisfaites par l'implantation choisie permet aussi de remonter dans la simulation hybride les retards dus à la latence des calculs. Il est ainsi possible de simuler une nouvelle fois le système hybride afin de vérifier que les temps de réponses sont satisfaisants et que le système est stable malgré l'introduction dans les lois de commande de retards dus aux temps de calcul.

5.4 Exemple

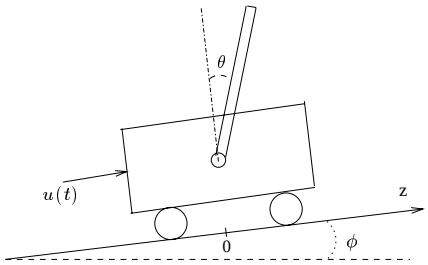
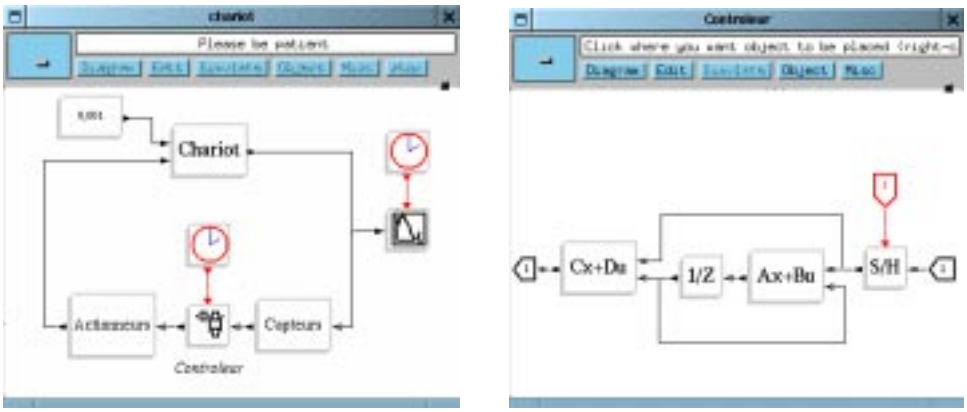


FIG. 3 – *Stabilisation d'un système chariot et pendule inversé sur un plan incliné*

Prenons l'exemple d'un système composé d'un chariot de masse M sur lequel est fixée une barre de masse m et de longueur l pouvant pivoter autour de son extrémité (figure 3). Ce chariot est posé sur un rail situé sur un plan incliné. On cherche à réaliser un système de contrôle-commande qui permet de stabiliser le chariot dans sa position d'origine 0. On suppose que des capteurs nous fournissent la position angulaire θ et la position z du chariot sur le plan incliné. La commande du système est une force $u(t)$ de translation matérialisée par un moteur. Nous montrons sur ce cas d'école bien connu des automaticiens, les bénéfices de la méthodologie proposée.

Modélisation - simulation La première étape de conception consiste à modéliser le système de contrôle-commande et à simuler le comportement du système automatisé afin de s'assurer que le but recherché est bien atteint. La figure 4(a) représente une modélisation réalisée avec l'interface graphique de Scicos. Le bloc `chariot` contient un modèle mathématique continu du chariot (équations de mouvement). Ce bloc est une fonction écrite en C ou en Fortran et à l'aide de blocs de la librairie Scicos. Le modèle de chariot décrit ici estime la position z et l'angle θ à partir de l'angle d'inclinaison ϕ de la pente (ici 0.001 radians) et de la commande $u(t)$ qui lui est appliquée. Le bloc `capteur` représente un modèle des deux capteurs qui réalisent les acquisitions de z et θ. De même, le bloc `actionneur` représente un modèle du moteur. Le bloc `controleur` est un modèle discret du système de contrôle-commande. Celui-ci décrit les algorithmes qui seront exécutés en temps réel. La période d'échantillonnage du `controleur`, ici 10ms, et donc sa cadence d'exécution, est spécifiée par un signal d'horloge relié à l'entrée d'*activation* du bloc. L'automaticien a ici utilisé une description hiérarchique, le bloc `controleur` est un *super-bloc* Scicos regroupant plusieurs blocs décrits en C et à l'aide de blocs de la librairie Scicos. Enfin, un



(a) Système automatisé (b) Extraction du contrôleur

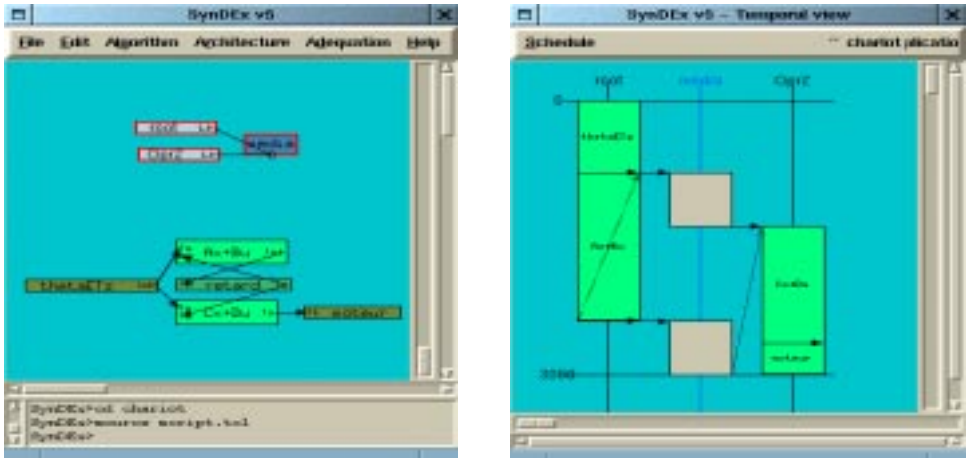
FIG. 4 – *Description Scicos*

bloc (icône représentant un tracé de courbe) permet d'afficher la courbe de simulation d'évolution dans le temps de la position z du chariot et de l'angle θ du pendule.

Extraction du contrôleur Lorsque la simulation hybride du système automatisé montre que le système de contrôle-commande est correctement dimensionné, l'utilisateur peut demander à Scicos de réaliser l'extraction du contrôleur en sélectionnant la région du graphe qui lui correspond (fig 4(b)). On remarque qu'ici l'automaticien a choisi de définir la loi de commande sous la forme d'un calcul matriciel donné par une représentation d'état. Ce graphe Scicos représentant le contrôleur discret peut alors ensuite être transcrit automatiquement en un graphe d'algorithme SynDEx.

Spécification du système temps réel Le graphe d'algorithme généré par Scicos peut être affiché dans l'interface graphique de SynDEx. Il reste à l'utilisateur à spécifier le graphe d'architecture représentant l'architecture matérielle du système temps réel et à fournir à SynDEx les durées maximales d'exécution de chacun des blocs de l'algorithme. Ces durées peuvent être, dans un premier temps, estimées par l'utilisateur ou mesurées par SynDEx après une première exécution du code généré. La figure 5(a) représente l'interface graphique de SynDEx dans laquelle sont affichés le graphe d'algorithme et le graphe d'architecture. Sur le graphe d'algorithme (partie basse) on retrouve l'équivalent des blocs Scicos de calcul matriciel. Le bloc `thetaEtz` est la fonction d'acquisition liée aux capteurs, la fonction `moteur` gère le moteur. Dans un but pédagogique, nous avons supposé ici que l'architecture matérielle est composée de deux microprocesseurs reliés par un média de communication et que les capteurs sont physiquement reliés au processeur nommé `root` et que la commande du moteur est gérée par le processeur `Opr2`.

Adéquation - Simulation temporelle L'utilisateur peut à partir de cette spécification demander à SynDEx de réaliser l'Adéquation et afficher une prédiction temporelle de l'exécution de l'algorithme sur l'architecture (fig. 5(b)). Chaque colonne représente la séquence d'exécution d'une itération sur un processeur ou sur un média de communication, dans notre cas, une séquence de calcul sur `root` et sur `opr2` et une séquence de communication sur `media`. L'axe vertical représente l'écoulement du temps. On peut ici remarquer que l'exécution complète de tous les calculs

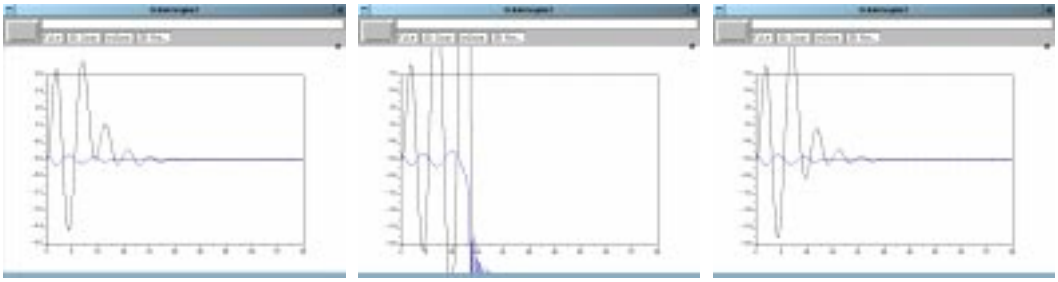


(a) Spécification (b) Prédiction temporelle

FIG. 5 – *Description SynDEx*

(communications inter-processeurs comprises) est estimée à environ 3ms (valeur 3060). On peut ainsi vérifier que l'architecture permet de satisfaire la contrainte de cadence d'exécution de 10ms définie pour la période d'échantillonnage de z et $theta$.

Simulation hybride avec prise en compte des temps de calcul Le temps de calcul de 3ms estimé par SynDEx définit ici le retard entre l'acquisition de z et θ, et la production de la commande $u(t)$. Ce retard peut être introduit dans le graphe Scicos afin de réaliser une simulation hybride plus fine. La figure 6(a) représente la première simulation obtenue après la modélisation. Cette simulation ne prend pas en compte les temps de calcul, elle met en évidence que le système de contrôle-commande dimensionné pour une période d'échantillonnage de 10ms permet de stabiliser le chariot en O. Les figures 6(b) et 6(c) sont les résultats des simulations réalisées avec prise en compte du temps de calcul de 3ms. La figure 6(b) montre que pour une période d'échantillonnage de 10ms, ce retard rend le système instable. La simulation de la figure 6(c) valide le système pour un contrôleur dimensionné avec une période d'échantillonnage de 8ms.



(a) T=10ms, sans retard (b) T=10ms, retard = 3ms (c) T=8ms, retard = 3ms

FIG. 6 – *Simulations*

Génération automatique du code Lorsque les simulations hybrides et temporelles montrent que le comportement du système est satisfaisant, il est possible de demander à SynDEx de générer les programmes à exécuter sur chacun des processeurs. SynDEx produit alors un exécutif qui gère les communications inter-processeurs et qui se charge d'appeler les fonctions relatives aux différents blocs de l'algorithme. C'est à l'utilisateur de fournir les fonctions d'entrées-sorties (ici `thetaEtz` et `moteur`). Il est par contre possible de réutiliser les fonctions C utilisées dans Scicos pour décrire le contrôleur : ainsi, ce sont les mêmes lignes de code qui ont été utilisées et validées lors de la simulation qui sont implantées sur le système temps réel.

5.5 Nouveau cycle de développement

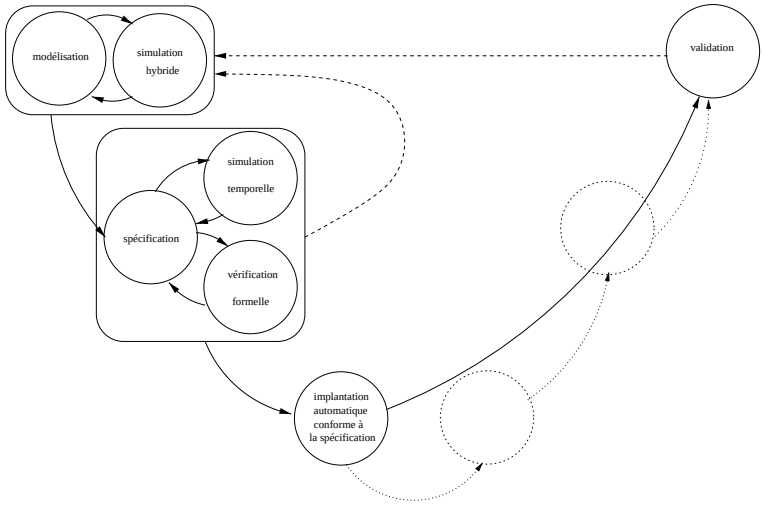


FIG. 7 – *Réduction du cycle de développement*

La figure 7 représente le nouveau cycle de développement réduit obtenu en utilisant ces outils. Chaque étape est ici validée par simulation et/ou vérification avant de passer à l'étape suivante. L'intervention manuelle pour passer de la modélisation à la spécification comme pour passer de la spécification à l'implantation a été réduite au minimum afin de limiter l'introduction d'erreurs. Nous espérons ainsi aboutir à un cycle proche d'un cycle idéal de développement en cascade sans remontée pour lequel chaque étape est validée avant de passer à la suivante.

6 Conclusion

Il existe deux grandes ruptures dans le cycle de développement en V de la partie logicielle d'un système temps réel. La première est le passage manuel du modèle à la spécification, la seconde est le passage manuel de la spécification à l'implantation reposant tous deux sur le savoir-faire des ingénieurs. Ces ruptures sont à l'origine d'erreurs nécessitant de nombreuses réitérations du cycle pour les corriger. Les méthodes formelles ont permis de supprimer la rupture entre la spécification et l'implantation grâce à la vérification et à la génération automatique du code conforme à la spécification. Nous avons cherché ici à éliminer la rupture modélisation-spécification, d'une part en rapprochant les terminologies des domaines concernés par ces étapes, d'autre part en interfaçant un outil de modélisation et de simulation hybride, Scicos, avec l'outil SynDEx de spécification et d'implantation optimisée multiprocesseurs.

Références

[1] BUHLER, H. *Conception de systèmes Automatiques*. Presses Polytechniques Romandes, 1988.

[2] DEJENIDI, R., LAVARENNE, C., NIKOUKHAH, R., SOREL, Y., AND STEER, S. From hybrid system simulation to real-time implementation. In *11th European Simulation Symposium and Exhibition* (Erlangen-Nuremberg, Oct. 1999).

[3] DELACROIX, J. *Un contrôleur d'ordonnancement temps réel pour la stabilité de Earliest Deadline en surcharge : le Régisseur*. PhD thesis, Université Pierre et Marie Curie, 1994.

[4] DORSEUIL, A., AND PILLOT, P. *Le Temps Réel en Milieu Industriel: Concepts, Environnements, Multitâches*. DUNOD, 1991.

[5] GRANDPIERRE, T., LAVARENNE, C., AND SOREL, Y. Optimized rapid prototyping for real time embedded heterogeneous multiprocessors. In *CODES'99 7th International Workshop on Hardware/Software Co-Design* (Rome, May 1999).

[6] HALBWACHS, N. *Synchronous programming of reactive systems*. Kluwer Academic Pub., 1993.

[7] HAREL, D., AND PNUELI, A. On the development of reactive systems. In *Logics and Models of Concurrent Systems*, Springer-Verlag, Ed., k. r. apt ed., vol. 13 of *NATO ASI*. New York, 1985, pp. 477–498.

[8] KOCIK, R., AND SOREL, Y. A methodology to design and prototype optimized embedded robotic systems. In *2nd IMACS International Multiconference CESA'98* (Hammamet, Tunisia, April 1998).

[9] LANDAU, I. D. *Identification et commande des systèmes*. Hermes, 1988.

[10] LE LANN, G. Critical issues for the development of distributed real-time computing systems. Research Report 1274, INRIA, August 1990.

[11] LENOIR, J. Les outils de conception système du logiciel enfoui. *Electronique*, 89 (Février 1999).

[12] NIKOUKHAK, R., AND STEER, S. Scicos : A hybrid system formalism. In *11th European Simulation Symposium* (Erlangen, Germany, october 1999).

[13] OGATA, K. *Discrete-Time Control Systems*. Prentice-Hall International Editions, 1987.

[14] PEREZ, J. *Systèmes Temps Réel: Méthodes de Spécification et de Conception*. DUNOD, 1990.

[15] SOREL, Y. Massively parallel computing systems with real time constraints, the "algorithm architecture adequation". In *Methodology Proc. of Massively Parallel Computing Systems Conference* (Italy, 1994).