SAIA : une utilisation conjointe du génie logiciel et des méthodes formelles

Julien Deantoni, Jean-Philippe Babau

SAIA

Jean-Philippe Babau *(à gauche)*
Professeur à l'Université de Brest
jean-philippe.babau@univ-brest.fr,

Julien Deantoni
INRIA Sophia Antipolis
julien.deantoni@inria.fr,

1 INTRODUCTION

Les ordinateurs personnels sont la manifestation la plus visible de l'arrivée massive de l'informatique dans notre quotidien. Pour autant, la quasi-totalité des processeurs que nous utilisons est intégrée dans des objets de notre quotidien (téléphone, carte à puce, électroménager, voiture, …): ce sont des systèmes embarqués. Ces dernières années, la complexité de ces systèmes s'est largement accrue (intégration de protocoles de communication, adaptation dynamique du comportement). Dans le même temps, il est nécessaire de développer ces systèmes de plus en plus rapidement, afin d'être réactif aux besoins du marché. Enfin, leur aspect enfoui, l'absence d'outils d'administration et parfois d'IHM, leur utilisation dans les systèmes critiques font qu'ils sont soumis à des contraintes fortes de sureté de fonctionnement. Au final, un processus de développement fiable et performant de système embarqué se doit d'allier l'application des principes du génie logiciel et l'utilisation de techniques formelles.

Les systèmes embarqués visés par cette étude sont des systèmes de contrôle de processus. Un tel système est en charge de réguler de manière fiable un processus dans son environnement physique. Pour cela il récupère des informations en provenance du monde extérieur (c'est-à-dire du processus, de son environnement physique et de l'opérateur) à l'aide de capteurs et il agit sur le processus à l'aide d'actionneurs (voire Figure 1). De ce fait, la communication d'un système embarqué avec le monde extérieur est un aspect essentiel de ces systèmes. Par exemple l'ABS (Anti Blocage System) communique de manière permanente avec le monde extérieur ; c'est-à-dire avec la roue et le frein de la voiture.

Le fait qu'un système embarqué soit lié à un processus et donc à la dynamique de ce processus en fait un système temps réel ; c'est-à-dire un système dont la validité dépend non seulement de l'action entreprise mais aussi du temps dans lequel l'action est entreprise. À titre d'illustration, la commande de freinage d'un véhicule est dite correcte si elle ralentit le véhicule, mais aussi, si elle est effectuée dans un temps borné après l'appui sur la pédale de frein.

Les caractéristiques énoncées précédemment impliquent des différences entre le développement des systèmes embarqués de contrôle de processus temps réel et le développement d'un système classique sur deux points principaux. D'une part, la criticité des systèmes et leurs contraintes temporelles imposent une vérification formelle. D'autre part, l'aspect dédié des choix technologiques est spécifique à chaque domaine, voire à chaque système. C'est par exemple le cas du système d'exploitation, du processeur, etc. ; mais plus particulièrement des moyens d'interaction de l'application avec le monde extérieur (les capteurs et les actionneurs). Ces choix technologiques impactent lourdement le comportement temporel des systèmes et doivent donc faire l'objet d'une attention particulière.

Afin de maîtriser finement les paramètres temporels lors du développement et aboutir à un système correct, les mé-

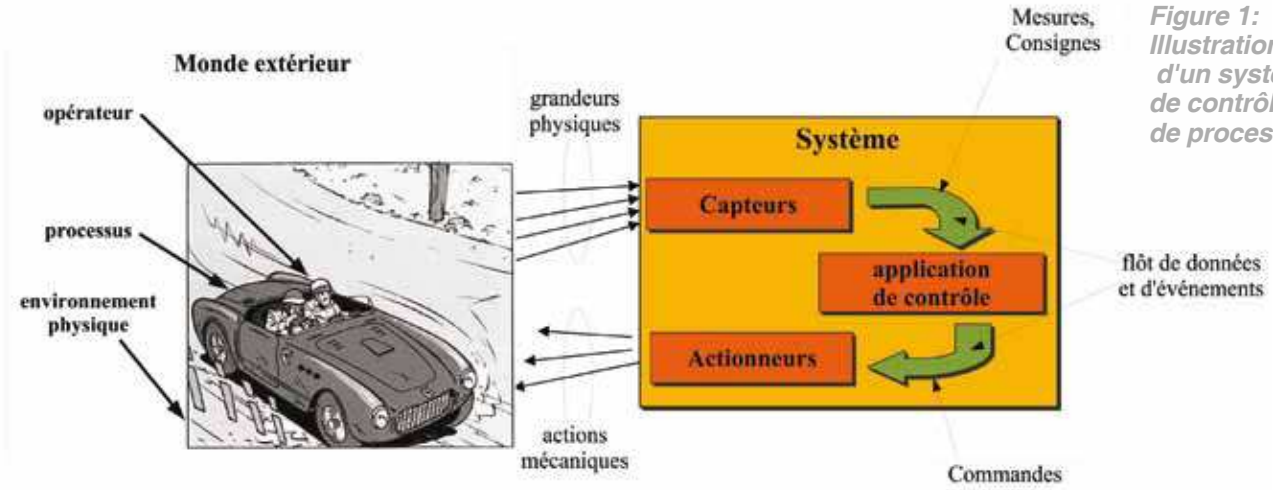


Figure 1: Illustration d'un système de contrôle de processus

Une utilisation conjointe du génie logiciel et des méthodes formelles

thodes utilisées sont souvent centrées sur l'implémentation et prennent en compte la totalité des paramètres du système afin de réaliser des réglages fins [3],[10]. Le système est développé et mis au point pour un contexte matériel et physique donné. Il est alors très difficile de réutiliser la partie logicielle de tels systèmes dans un contexte matériel différent [2].
Pour autant, afin d'être réactif aux besoins du marché, ces systèmes doivent aujourd'hui pouvoir évoluer rapidement, que ce soit au niveau de la partie matérielle ou de la partie logicielle qui y est implantée. En particulier, comme dans les systèmes classiques, un besoin de plus en plus exprimé par les industriels est de pouvoir développer une application logicielle en s'affranchissant de la cible matérielle et en particulier en s'affranchissant des capteurs et des actionneurs. Ceci est d'autant plus vrai que les premières phases du développement des applications logicielles ne sont pas effectuées sur une cible matérielle réelle mais sur une cible matérielle simulée. Permettre de s'affranchir de la cible matérielle lors du développement offre deux avantages certains :
• une application développée et validée sur une cible matérielle simulée peut être déployée sur une cible réelle sans modification ;
• une application peut être déployée sur des cibles matérielles différentes.
Cependant, puisque la qualité de l'interaction entre l'application et le processus impacte fortement la validité du système, afin de construire une application correctement réutilisable, il est nécessaire de pouvoir qualifier la qualité d'interaction nécessaire à une application pour exécuter un contrôle correct du processus. Ces différentes préoccupations nous conduisent à nous poser une série de questions :
• Comment développer une application de contrôle de processus indépendamment d'une technologie spécifique de capteurs et d'actionneurs ?
• Comment spécifier et valider la qualité d'interaction nécessaire à l'application pour assurer un contrôle correct ?
• Comment valider cette qualité d'interaction à l'aide d'une phase de simulation ?
• Et enfin, comment s'assurer qu'une technologie de capteurs et d'actionneurs réels respecte la qualité d'interaction spécifiée par l'application ?
Cet article résume les réponses apportées à ces questions au travers de la thèse effectuée par Julien DeAntoni au sein du laboratoire CITI de l'INSA de Lyon. La thèse s'appuie sur l'utilisation des principes de génie logiciel liés à la structuration de logiciel (software architecture) et aux approches dirigées par les modèles. Plus particulièrement, l'étude s'attache à fournir, d'une part des principes de structuration permettant de développer une application de contrôle de processus indépendamment d'une technologie de capteurs et d'actionneurs ; et d'autre part à assurer, par une analyse formelle, la correction de l'application suite à son déploiement sur une technologie de capteurs et d'actionneurs spécifiques. Ces deux préoccupations ont été traitées en fournissant :
• un style architectural, appelé SAIA (pour SAIA (Sensors / Actuators Independent Architecture) définissant des règles de structuration du logiciel via différents types de composants et des contraintes sur leur association permettant d'obtenir une structure analysable ;
• une formalisation des contraintes temporelles concernant : la communication d'une application de contrôle avec le monde extérieur et la communication de l'application de contrôle avec une plateforme spécifique de communication avec le monde extérieur ;
• une formalisation de l'établissement d'un contrat de qualité de service (QoS : Quality of Service) lors du déploiement de l'application de contrôle sur une plateforme de communication avec le monde extérieur,
• une proposition de méthode et d'outil pour faciliter la mise en œuvre du style architectural et des analyses formelles.
Afin d'atteindre les objectifs précédemment décrits. SAIA est basé sur une ensemble de modèles. En effet, au-delà des langages et des technologies, ce sont bien les concepts, leurs liens et leur enchaînement au cours du développement qui nous intéressent ici. En suivant une approche d'ingénierie Dirigée par les Modèles (IDM), les types de composants, leur QoS ainsi que les contraintes sur leurs associations sont exprimées à l'aide de différents méta-modèles. Une première partie de ce chapitre permet d'introduire les principes généraux de structuration des composants selon le style architectural SAIA. Une deuxième partie permet de décrire plus précisément les types de composants, de connecteurs et de contraintes induits par le style SAIA. Enfin, une troisième partie se focalise sur la QoS.

2 LE STYLE ARCHITECTURAL

2.1 La structuration en couches

La structuration en couche est une façon classique de structurer un système lorsque l'on désire être indépendant d'une plateforme. En se basant sur les idées développées par les modèles de structuration dédiés aux IHMs (notamment PAC [6] et Seeheim [7]), SAIA spécifie le système selon trois couches distinctes. On trouve dans une première couche l'application de contrôle et ses interfaces puis, dans une autre couche, la technologie de communication entre le système et le monde extérieur. Enfin, entre ces deux couches, une troisième couche assure la liaison au travers d'adaptations et de contrôles (voire Figure 2).

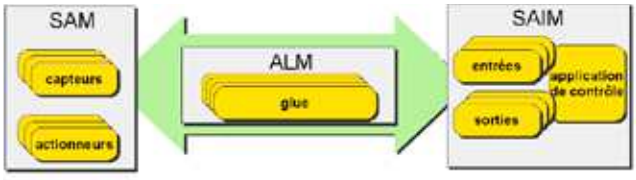


Figure 2: La structuration préconisée dans SAIA

1. La première couche est spécifiée dans le SAIM (Sensors/Actuators Independent Model). Elle définit l'application et ses interfaces en termes d'entrées et de sorties. Les entrées représentent les données et les phénomènes physiques pertinents pour l'application. Les sorties, quant à elles, représentent les actions physiques utiles pour le contrôle du processus. Les entrées ainsi que les sorties sont spécifiées sans fournir de renseignements concernant la technologie grâce à laquelle elles sont acquises ou réalisées. L'ensemble composé des entrées et des sorties est ainsi la spécification d'une plateforme abstraite [1]. Elle fournit un ensemble de services d'écritures et de lectures sur l'environnement extérieur indépendamment de la plate-forme (au sens du MDA [14]) des services fournis par les pilotes des capteurs et des actionneurs.
2. La deuxième couche est spécifiée dans le SAM (Sensors/Actuators Model). Elle définit les services de communication entre le système et le monde extérieur. Cette communication est réalisée par des éléments de la plateforme matérielle : les capteurs et les actionneurs. Les capteurs fournissent une vue du monde extérieur et les actionneurs réalisent des actions sur le processus. Plus précisément, cette couche définit les services des pilotes de capteurs et des pilotes d'actionneurs, accessibles au travers d'interfaces d'entrées et de sorties. Il s'agit donc de la spécification de la plateforme au sens du MDA en termes de capteurs et d'actionneurs.
3. La troisième couche est spécifiée dans l'ALM (Adaptation Layer Model). Elle permet aux deux modèles présentés précédemment de spécifier un système complet. Dans cet objectif, elle réalise une connexion (mapping au sens du MDA) entre la plateforme abstraite et la plateforme concrète. Cette dernière couche, également, appelée connecteur complexe, est au cœur de l'approche et est détaillée dans le chapitre suivant.

2.2 Le connecteur complexe

L'ALM est la spécification d'un connecteur complexe qui réalise la liaison entre les interfaces des composants du SAM et celles des composants du SAIM. Les entrées et les sorties du SAIM sont décrites de manière abstraite alors que les pilotes des capteurs et des actionneurs offrent des services liés aux technologies, les flots d'informations et les interfaces spécifiés par les uns et les autres sont donc, a priori, hétérogènes. Le but du connecteur complexe est de spécifier la « glue » permettant d'homogénéiser les flots d'information entre le SAM et le SAIM.
D'un point de vue structurel, le connecteur complexe est défini par un assemblage de sous-connecteurs. Chaque sous-connecteur est dédié à une et une seule entrée abstraite, ou à une et une seule sortie abstraite. Son rôle est de spécifier comment construire, resp. réaliser, une entrée, resp. une sortie, abstraite vis-à-vis de la plateforme concrète. Un sous connecteur est relié aux éléments de la plateforme concrète utiles à la construction, resp. la réalisation, de l'entité abstraite considérée.
Afin d'éviter une redondance des calculs dans différents sous connecteurs, les sous-connecteur peuvent communiquer entre eux.
Chaque sous-connecteur est lui-même spécifié par un assemblage de composants d'adaptation. L'adaptation consiste à traiter l'hétérogénéité entre les entrées et les sorties de sous-connecteurs. SAIA identifie trois types d'hétérogénéités, et donc de types de composants, entre les flots d'informations spécifiés dans le SAIM et ceux spécifiés dans le SAM.
La première hétérogénéité concerne le type de valeurs spécifiées par les flots de données. Celle-ci peut être exprimée dans une autre unité, un autre repère, etc. Effectuer des conversions pour assurer l'homogénéité des valeurs constitue l'étape de formatage, réalisée par des composants de type Format.
La deuxième hétérogénéité est due à la différence de niveau d'abstraction des composants du SAM et de ceux du SAIM. Par exemple, le SAIM peut spécifier une entrée consigne_Vitesse dans la plateforme abstraite. Afin de réaliser cette entrée sur une plateforme concrète, il est possible d'utiliser un joystick. Les valeurs du flot de données spécifiées par consigne_Vitesse sont donc des vitesses, alors que les valeurs du flot de données en sortie du pilote du joystick sont des coordonnées. Passer de coordonnées de Joystick à une vitesse (et éventuellement une rotation) constitue une étape d'interprétation nécessaire, programmable, mais non automatisable a priori. L'interprétation est le rôle des composants Interpret.
La troisième hétérogénéité concerne la QoS de chacun des flots d'informations. Les tenants et aboutissants de cette source d'hétérogénéité, gérés par les composants QoSAdapt, sont décrits plus en détail dans la suite de ce chapitre.

2.3 QoS et qualité de contrôle

La séparation en trois modèles distincts permet de réaliser une séparation franche du point de vue fonctionnel entre l'application et la plateforme. Cependant, une application de contrôle de processus est aussi sensible à plusieurs paramètres non fonctionnels qui influencent la qualité du contrôle. La qualité de contrôle est un ensemble de contraintes exprimées sur le processus en termes de stabilité ou de marges d'erreurs. Par exemple, pour un robot devant suivre une ligne sur le sol, une contrainte sur la qualité de contrôle est que le robot ne doit pas dévier de la trajectoire désirée de plus de 1cm. La qualité de contrôle d'une application est fortement influencée par les caractéristiques temporelles concernant l'acquisition des informations en provenance du monde extérieur ainsi que sur la réalisation des actions physiques sur le processus [15]. SAIA propose alors de dériver les contraintes de qualité de contrôle en contraintes temporelles sur les flots d'informations en provenance et à destination de l'application. Au niveau du SAIM, cela se traduit par une spécification de la QoS dite abstraite sur la plateforme abstraite qui, lorsqu'elle est satisfaite, assure d'atteindre la qualité de contrôle recherchée par l'application.
Une fois le SAIM et sa QoS spécifiés, il faut être capable de connecter la plateforme abstraite à une plateforme concrète en s'assurant que la QoS est satisfaite. Il est alors d'abord nécessaire de spécifier la QoS de la plateforme concrète, puis il faut évaluer la correction de la connexion SAM/SAIM, via l'établissement d'un contrat de satisfaction de QoS. La QoS du SAM découle d'une évaluation de performances des pilotes existants. La QoS dépend de l'architecture des pilotes et du matériel (capteur, actionneur) associés aux pilotes [4]. Puisque l'étude se focalise sur l'établissement d'un contrat de QoS entre le SAM et le SAIM, les travaux développés dans le cadre de cette thèse présument que la description de la QoS du SAIM et du SAM est disponible. Nous étudions maintenant sur le contrat de QoS entre le SAM et le SAIM.

2.4Le contrat de QoS

Lors de la connexion du SAIM et du SAM, il est nécessaire d'établir un contrat de QoS (contrat de niveau 4 au sens de [12]) pour garantir la correction de l'application vis-à-vis de

Une utilisation conjointe du génie logiciel et des méthodes formelles

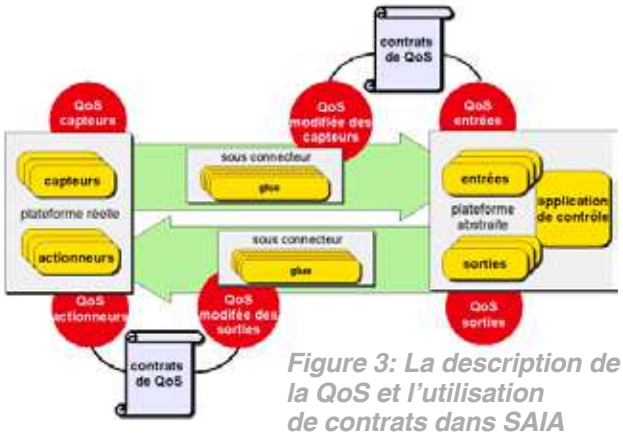


Figure 3: La description de la QoS et l'utilisation de contrats dans SAIA

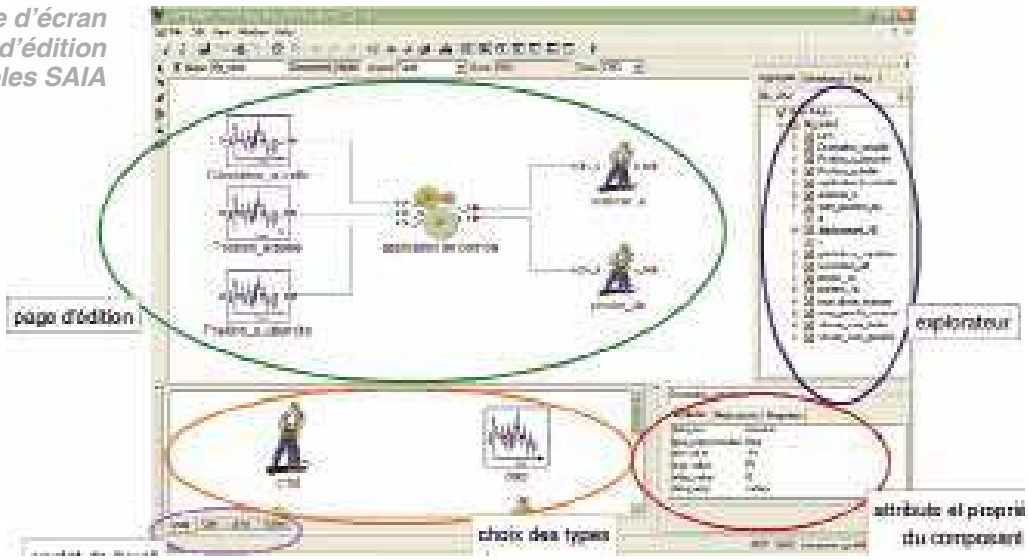
sa qualité de contrôle. Ceci se traduit par l'établissement d'un contrat de QoS entre chacune des entrées et des sorties spécifiées dans le SAIM et chacun des pilotes spécifiés dans le SAM. Ce contrat doit exprimer si la relation de satisfaction entre QoS requise et QoS fournie est vérifiée.
L'établissement d'un contrat de QoS est à la charge du connecteur complexe. Cependant, nous avons vu que le connecteur complexe est composé de sous connecteurs possédant chacun une « glue » afin de rendre homogènes les flots d'informations entre le SAM et le SAIM. En modifiant les flots d'informations, la « glue » modifie également leur QoS, spécifiée dans le SAM et dans le SAIM. Avant de pouvoir établir le contrat de QoS, l'impact de la « glue » sur la QoS est évalué par la réalisation d'analyses temporelles. Ces analyses formelles s'appuient sur des outils du Model-Checking.
Pour assurer la faisabilité de ces analyses, les composants doivent spécifier un comportement temporel pertinent (abstraction correcte du composant à l'exécution) en s'appuyant sur une sémantique opérationnelle temporisée formelle. Les informations temporelles manipulées caractérisent les flots d'information (fréquence et retards), les durées des traitements (temps d'exécution et temps de blocage induits par l'ordonnancement des actions) et le comportement des sous-connecteurs.

L'ensemble des informations temporelles permet d'effectuer les analyses temporelles par observation formelle des flots et ainsi d'évaluer la QoS des flots d'informations modifiés par la « glue » des sous connecteurs.
Il est alors possible de vérifier si la connexion entre le SAM et le SAIM donne lieu à l'établissement d'un contrat ou non (cf. figure 3). La relation de satisfaction s'appuie sur une relation de raffinement ou de simulation entre la QoS fournie et la QoS requise « la QoS fournie simule ou raffine la QoS requise ». L'idée ici est que la QoS requise est vue comme sur-ensemble des QoS acceptables, la QoS fournie est vue comme une solution acceptable parmi les possibles. La notion de simulation est à ensuite préciser selon le type de QoS traitée, soit l'égalité pour des constantes, l'inclusion pour des intervalles et une relation de sous-langage pour des QoS exprimées à l'aide d'expressions régulières.
Une fois les principes posés, nous passons à une phase de mise en œuvre de l'approche.

2.5Mise en œuvre

La mise en œuvre de SAIA comporte deux volets principaux : d'une part la mise en œuvre de la chaîne d'outil permettant d'utiliser SAIA et d'autre part la mise en œuvre de SAIA au travers de différents cas d'étude.
L'expression d'un style architectural nécessite de faire le choix d'un langage de description des composants auxquels s'applique le style. Les avantages liés à l'utilisation de (méta-)modèles afin de manipuler les concepts nous permettent de nous affranchir d'un langage de programmation particulier. Ainsi, SAIA décrit le style architectural au travers de différents méta-modèles. Ces méta-modèles ont été mis en place dans différents environnements de l'IDM (EMF sous Eclipse et l'outil GME [11]). A partie des méta-modèles, il est possible de générer automatiquement divers éditeurs de modèles, permettant de créer des modèles conformes au style architectural décrit précédemment (voire par exemple la capture d'écran Figure 4). Associés à ces éditeurs, différents parsers ont été réalisés pour permettre d'utiliser les outils de validation formelle nécessaire à la réalisation de contrats (en particulier à l'aide du langage formel IF [5]). De

Figure 4: Capture d'écran de l'outil d'édition de modèles SAIA



UNE UTILISATION CONJOINTE DU GÉNIE LOGICIEL ET DES MÉTHODES FORMELLES

plus, au sein de l'environnement Eclipse, un simulateur de modèle a été développé grâce au langage KerMeta [13]. Ce simulateur permet d'animer tous les modèles SAIA afin d'en vérifier la correction.

Afin d'éprouver les concepts mis en œuvre dans SAIA, plusieurs cas d'étude ont été traités dont deux concours internationaux de temps réel [8], [9]. Le premier concours consistait à construire le logiciel de contrôle d'un robot autonome devant aller d'un point A à un point B dans un environnement inconnu, parsemé d'obstacles. Ce concours a permis de montrer la faisabilité de l'approche en réalisant les modèles nécessaires au générateur de code pour une application de contrôle de processus. Nous avons obtenu la deuxième place (mesure du temps d'arrivée avec évitement des obstacles), montrant ainsi que l'utilisation de modèles et l'application de principes de génie logiciel n'étaient pas fatalement contradictoires avec la notion de performance. Lors du deuxième concours, le but du robot était d'aller d'un point A à un point B puis de retourner au point A. La plateforme matérielle était cependant complètement différente de celle du premier concours. Lors de ce concours nous avons entièrement réutilisé le modèle SAIM (application et plateforme abstraite) du premier concours; montrant ainsi que notre approche permettait la réutilisation de l'application, indépendamment de la plateforme matérielle. De plus, en arrivant quatrième parmi un nombre largement supérieur de concurrents, nous soulignons encore une fois le faible surcoût introduit par l'utilisation de modèles. Il faut aussi ajouter, que si le temps de développement lors du premier concours a été de quatre semaines, les développements, lors du deuxième concours, n'ont pris qu'une semaine. Enfin l'utilisation de SAIA nous assure d'une réutilisation correcte des algorithmes de contrôle sur une cible réelle, les concours ayant été réalisés sur simulateurs.

3 CONCLUSION

Les travaux décrits dans ce papier montrent comment, à l'aide de l'utilisation conjointe de principes de génie logiciel et de techniques formelles, intégrés dans une approche dirigée par les modèles, il est possible d'améliorer la réutilisation de logiciel pour les systèmes embarqués de contrôle de processus, tout en contrôlant leur correction. D'une part le style architectural permet d'avoir une structure en couche et d'imposer des contraintes sur les composants impliqués dans cette structuration, d'autre part, les approches formelles permettent, à partir d'informations issus des modèles, de s'assurer de la correction d'un système. La validation formelle de la correction du système permet de vérifier si une application de contrôle est utilisable sur une plateforme matérielle ou non.

Le découplage entre l'application et la plateforme concrète est réalisé via l'introduction d'une plateforme abstraite. Cette plateforme abstraite sert également de support à description de la QoS. Il serait intéressant, dans des travaux futurs, d'étendre cette notion de plateforme abstraite QoS aware à des plateformes de communication et des plateformes d'exécution multitâches et multi-core.

RÉFÉRENCES

[1] Almeida J.P., Dijkman R., van Sinderen M. Pires L.F. – On the notion of abstract platform in MDA development – Proceedings of the eighth IEEE International Enterprise Distributed Object Computing Conference – 2004, pages 253-263

[2] ARTIST 2. Selected Topics in Embedded System Design: Roadmap for research – 2004, pages 22-37

[3] Bate I., Nightingale P., Cervin A. – Establishing Timing Requirements and Control Attributes for Control Loops in Real-Time Systems. Proceedings of the 15th Euromicro Conference on Real-Time Systems – 2003, pages 121-128

[4] Ben-Hedia B., Jumel F., Babau J.P. – Formal evaluation of Quality of Service for data acquisition systems – Forum on specification and Design Language – 2005

[5] Bozga, M. and Fernandez, J.C. and Ghirvu, L. and Graf, S. and Krimm, J.P. and Mounier, L. – IF: A Validation Environment for Timed Asynchronous Systems – Proceedings of the 12th International Conference on Computer Aided Verification (Springer-Verlag London, UK) – 2000, pages 543-547

[6] Coutaz J. – Interfaces hommes-ordinateurs : conception et réalisation – Paru chez Dunod – 1990, pages 161-186

[7] Coutaz J. – Interfaces hommes-ordinateurs : conception et réalisation – Paru chez Dunod – 1990, pages 139-145

[8] DeAntoni J., Babau J.P. – SAIA: Sensors/Actuators Independent Architecture - A showcase through maRTian task specification – Proceedings of the ERTSI 2005 - Embedded Real Time Systems Implementation Workshop, held in conjunction with 26th IEEE International Real-Time Systems Symposium – 2005, pages 43-50

[9] DeAntoni J., Babau J.P. – Cibermouse Design: A case study for SAIA model reuse – Proceedings of the Cibermouse Team reports, a satellite event of the 27th IEEE International Real-Time Systems Symposium – 2006, pages 9-12

[10] Dspace. Produit logiciels intuitifs pour le processus de développement de calculateur (http://www.dspace.fr/ww/fr/fra/home/products/sw.cfm, 2007

[11] ISIS – The Generic Modeling Environment (GME) – Institute for Software Integrated Systems Vanderbilt University – http://www.isis.vanderbilt.edu/Projects/gme/ – 2005

[12] Jézéquel J.M., Defour O., Plouzeau N. – An MDA Approach to Tame Component Based Software Development – Second International LNCS Symposium on Formal Methods for Components and Objects – 2003

[13] Muller P.A., Fleurey F., Jezequel J.M. – Weaving Executability into Object-Oriented Meta-languages – 8th International Conference Model Driven Engineering Languages and Systems – 2005

[14] OMG-MDA – Model Driven Architecture guide V1.0.1 – http://www.omg.org/mda – 2003

[15] Sandfridson M., Törngren M., Wikander J. – The Effect of Randomly Time-Varying Sampling and Computational Delay – Proceedings of the IFAC world congress – 2005