Prédictibilité des assemblages de composants

Des logiciels clairs comme l'eau de roche !

Magnus Larsson, Anders Wall, Kurt Wallnau

Quiconque travaille sur ordinateur sait que les logiciels se comportent parfois de manière inattendue. Sur un PC bureautique, des moyens existent pour s'en prémunir, le « bug » étant plus agaçant que préjudiciable. Dans les applications industrielles, les conséquences sont plus graves. Le dysfonctionnement d'une machine peut non seulement entraîner des pertes de production, mais également poser des problèmes de sécurité et de qualité. La complexité intrinsèque des logiciels fait que souvent des erreurs passent au travers des tests classiques désormais incapables d'analyser toutes les situations possibles. Des approches mathématiques de développement et de vérification réduisent considérablement les risques et contribuent à la maîtrise de la qualité. L'université de Carnegie Mellon et ABB font équipe afin de développer des techniques pour concevoir des logiciels de qualité.

Les logiciels jouent un rôle prépondérant dans l'offre de solutions innovantes d'ABB à ses clients. Or ils doivent souvent fonctionner dans des environnements très contraignants en termes de délai, de sécurité, de fiabilité et de sûreté. Dans ce contexte, les défaillances logicielles coûtent cher et peuvent avoir des conséquences catastrophiques. Qui plus est, l'escalade des coûts de développement de logiciels hautement fiables pose un défi à toute la profession. La taille des systèmes actuels, sans parler de ceux de demain, explique l'incapacité fondamentale de s'appuyer sur des tests pour garantir un haut degré de confiance. ABB et le SEI (*Software Engineering Institute*) de l'université de Carnegie Mellon ont développé une approche qui garantit dès la phase de construction la prédictibilité du comportement critique des systèmes en phase d'exécution. Cette approche réduit le coût des tests et raccourcit les délais de mise sur le marché de nouveaux logiciels dont la qualité est hautement garantie.

On a souvent dit que les trois principes fondamentaux de la programmation étaient la modularité, la modularité et encore la modularité. Bientôt, on dira de même que ceux du génie logiciel sont la contrainte, la contrainte et toujours la contrainte.

Les contraintes se trouvent au cœur de toutes les disciplines techniques. Un problème peut être source de difficultés, mais l'ingénieur chevronné saura le «contraindre» pour le résoudre par des techniques éprouvées et maîtrisées. Ces techniques imposent des contraintes à la fois au problème à résoudre et à la forme que les solutions peuvent prendre. La perte de liberté du fait de ces contraintes est plus que compensée par la possibilité de résoudre de manière prédictible et systématique des classes entières de problèmes.

Le développement logiciel est moins préoccupé par les programmes *per se* que par les grands réseaux de programmes interdépendants. A ce niveau, les défis techniques vont bien au-delà des aspects fonctionnels (domaine de la programmation) pour englober des aspects non fonctionnels tout aussi cruciaux (parfois appelés «attributs de qualité») comme la sécurité, les performances, la disponibilité, la tolérance aux pannes, etc. Une difficulté majeure à laquelle se heurtent les chercheurs en génie logiciel est de mettre au point des techniques pour construire des systèmes prédictibles dans leurs aspects non fonctionnels. Il s'ensuit que ces techniques imposeront des contraintes sur la manière dont les futurs systèmes logiciels seront construits.

Dans cet article, nous montrons comment des «contraintes intelligentes» sont introduites dans le processus de développement logiciel pour prédire à coup sûr la qualité des systèmes logiciels. Ces contraintes peuvent ainsi être intégrées à l'infrastructure logicielle pour élaborer des systèmes prédictibles par construction. Les jours des tests de qualité des logiciels sont peut-être comptés ! De surcroît, la prédictibilité par construction servirait à imposer aux logiciels tiers des standards de qualité objectifs et mesurables dont l'utilité prédictive est établie.

Prédictibilité par construction

L'approche présentée ici part d'un double postulat:

1) Les contraintes intelligentes donnent des systèmes aux qualités d'exécution prédictibles;
2) La technologie des composants encapsule des contraintes qui permettent de développer des logiciels prédictibles par construction.

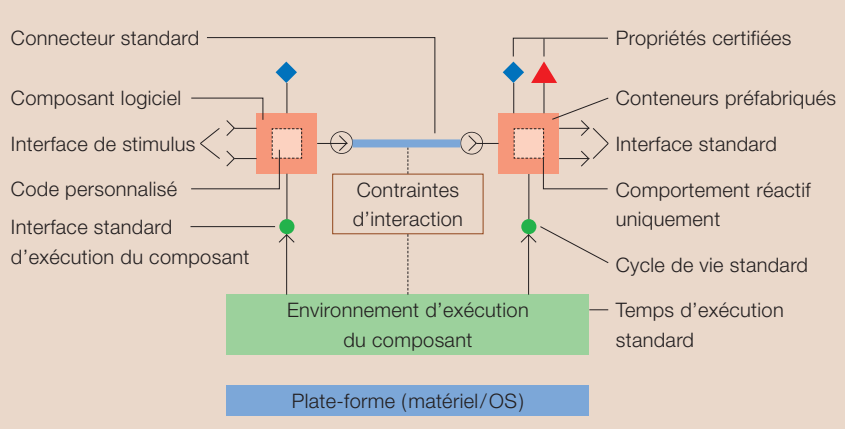
Partant de là, certaines idées sont introduites: une qualité d'exécution doit être définie en termes d'observations pouvant être faites sur les traces d'exécution. Une qualité d'exécution est *prédictible* si et seulement s'il existe une théorie (règle) de prédiction des observations futures. Ici, l'essentiel est que la qualité est définie par rapport à une théorie prédictive et que cette théorie doit donner confiance dans ses prédictions.

Cette idée n'est pas nouvelle dans le domaine des sciences ou du logiciel. Le comportement temporel d'un système logiciel doit être prédictible en utilisant la théorie d'ordonnancement à priorité statique RM (*Rate Monotonic*) généralisée ou la théorie des files d'attente temps réel. Ces deux théories (toutes les théories, de manière générale!) émettent des hypothèses sur les systèmes étudiés et tout système qui confirme ces hypothèses est prédictible au titre de ces théories. Les contraintes intelligentes garantissent la satisfaction de ces hypothèses; elles sont intelligentes *précisément* parce qu'elles sont formalisées par des théories prédictives.

Définir une contrainte intelligente est une chose; garantir sa satisfaction en est une autre. Un idiome récurrent de la technologie des composants particulièrement efficace pour l'encapsulation de contraintes intelligentes est illustré en 1.

Dans cet idiome, le logiciel personnalisé est déployé dans des conteneurs préfabriqués [1]. Un conteneur restreint à la fois la visibilité du code personnalisé à son environnement externe et celle de l'environnement au code personnalisé. Différents types de conteneur peuvent jouer des rôles différents dans un schéma de coordination global (défini par l'architecture). Un composant logiciel dans cet idiome est un conteneur combiné au code personnalisé. Les composants sont strictement réactifs: ils ne réagissent qu'aux stimuli reçus à travers l'interface du conteneur et y répondent uniquement par le biais de celle-ci. L'environnement d'exécu-

1 Idiome du conteneur



tion d'un composant fournit des mécanismes de coordination (ou «connecteurs») et implémente d'autres politiques de gestion des ressources partagées par les composants.

A ce stade, il importe que l'utilisateur ne s'arrête pas à une technologie de composants particulière: de nombreuses implémentations de l'idiome sont possibles, y compris souvent des versions simples. L'important c'est que les types de conteneur et de connecteur, l'environnement d'exécution et la possibilité d'imposer des contraintes à des modes autorisés d'interaction entre composants puissent tous servir à encoder, ou encapsuler, des contraintes intelligentes. Par ailleurs, le nombre réduit et l'uniformité des abstractions de cet idiome simplifient considérablement le travail d'automatisation de pans importants du processus de construction et de prédiction, pour obtenir une prédictibilité par construction.

L'idée est simple et mieux comprise par analogie. Un compilateur Java ou C# vérifie la bonne constitution des programmes, notamment le respect de la théorie des types du langage de programmation. Si cette contrainte est satisfaite, le compilateur garantit alors certaines propriétés d'exécution du programme (techniquement des propriétés de sécurité). Dans ce cas-ci, la même idée est appliquée mais au niveau des assemblages de composants et non plus du programme. La théorie des types est remplacée par la théorie des comportements pour les aspects d'exécution non fonctionnels.

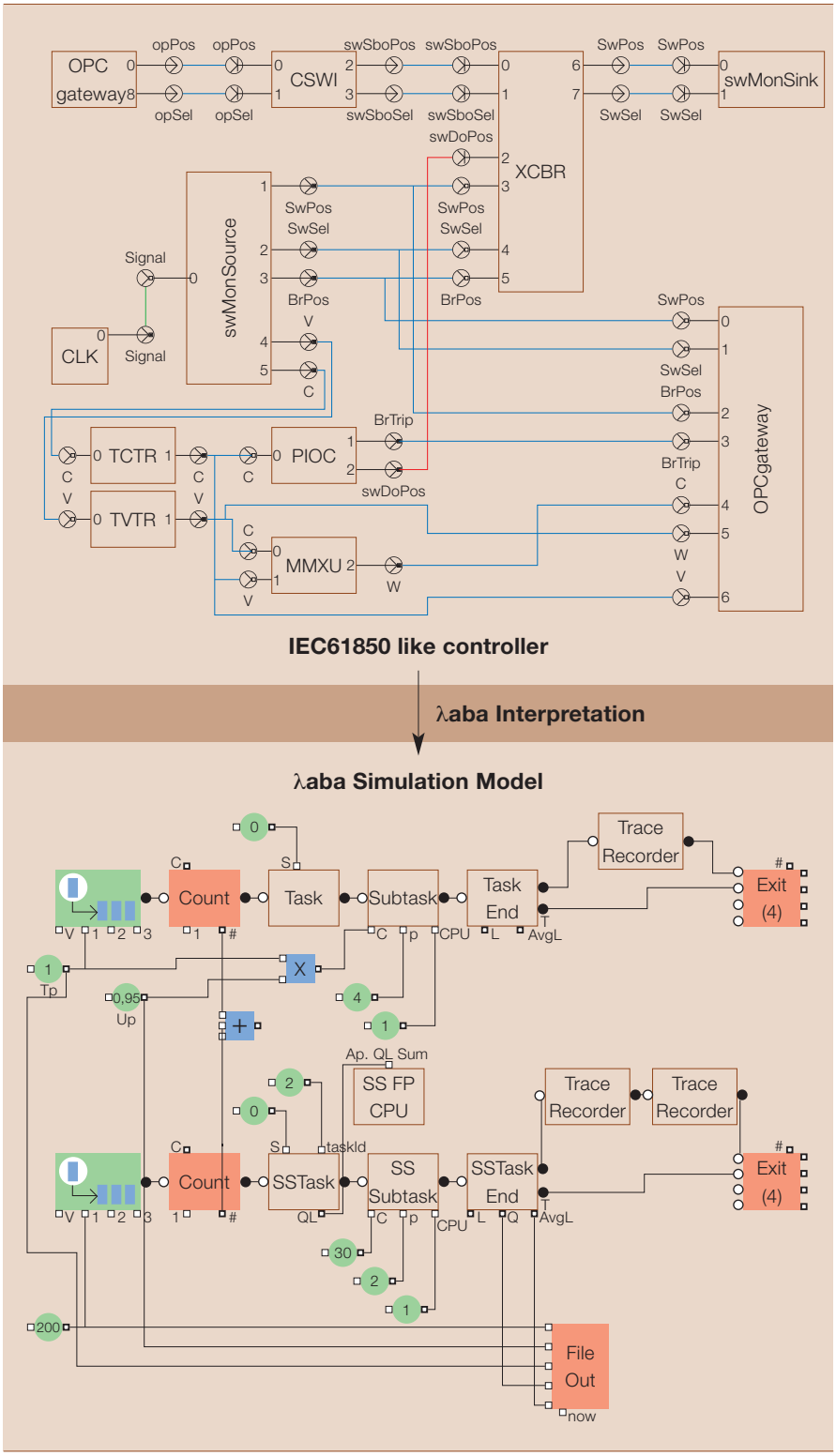
Au lieu de spécifications dans un langage de programmation, on utilise des spécifications dans un langage de description d'architecture (CCL [4]). En fait, CCL formalise l'idiome du conteneur et permet d'automatiser la prédiction et la génération du code, pour réaliser la prédictibilité par construction. Si les spécifications en CCL sont conformes à l'idiome du conteneur et satisfont d'autres contraintes spécifiques du cadre de raisonnement (*reasoning framework*), les systèmes spécifiés seront prédictibles par construction. L'objectif ultime de la prédictibilité par construction est de ne bâtir que des systèmes au comportement prédictible.

D'autres projets ABB ont déjà combiné technologie des composants et comportement non fonctionnel prédictible (ex., PECOS [13]). Cependant, aucun travail antérieur n'avait généralisé ces idées à plusieurs aspects non fonctionnels ou insisté sur le rôle de la validation et de la certification tel que présenté ici.

Qualité certifiable

Les théories analytiques identifient les propriétés du logiciel à connaître si son comportement doit être prédictible. Une technologie des composants impose une encapsulation standard des logiciels, notamment la manière dont ses composants sont spécifiés et les détails de leur implémentation qui doivent être donnés par les fournisseurs. Tous ces éléments constituent une base concrète de définition de standards de qualité objectifs pour les logiciels tiers.

2 Interprétation



A titre d'exemple, supposons la prédiction du comportement temporel des assemblages de composants avec la théorie des files d'attente temps réel de Lehoczky [2]. Cette théorie possède, entre autres, trois propriétés:

- un ordre d'ordonnancement dynamique de la plus petite échéance d'abord EDF (*Earliest Deadline First*),
- l'identification des entités ordonnançables (ex., unités d'exécution),
- le deuxième moment de temps de fonctionnement prévu pour chaque entité ordonnançable.

La première propriété est satisfaite par l'environnement d'exécution des composants et la deuxième par les conteneurs. La troisième, toutefois, doit être satisfaite par le fournisseur de composants. A ce stade, il est important de clarifier certains points. Primo, ce qui doit être satisfait est une mesure précise du deuxième moment de temps de fonctionnement, ce qui correspond aux «propriétés certifiées» en 1. Savoir s'il est souhaitable de restreindre la plage de valeurs que ces mesures peuvent prendre est une autre question. Secundo, la théorie des performances doit donner une définition précise de la mesure et peut éventuellement servir de guide au processus de mesure lui-même.

Bien évidemment, la quête du logiciel «de confiance» ne s'arrête pas à la qualité certifiable. Une approche plus globale s'impose, à savoir améliorer la technologie du logiciel et ses processus de développement, y compris les processus sociaux [10] [12]. Quand bien même, l'approche présentée ici est un premier pas vers la définition de standards de qualité à la fois objectifs et prédictibles. En particulier, il est intéressant d'observer l'approche décrite directement sur les produits logiciels plutôt qu'indirectement, par exemple en mesurant le niveau de maturité des processus des organisations de développement.

Application ABB

ABB et l'Institut SEI ont d'abord testé la faisabilité de la prédictibilité par construction dans le domaine de l'automatisation des postes électriques [3]. Ce travail a donné trois résultats principaux.

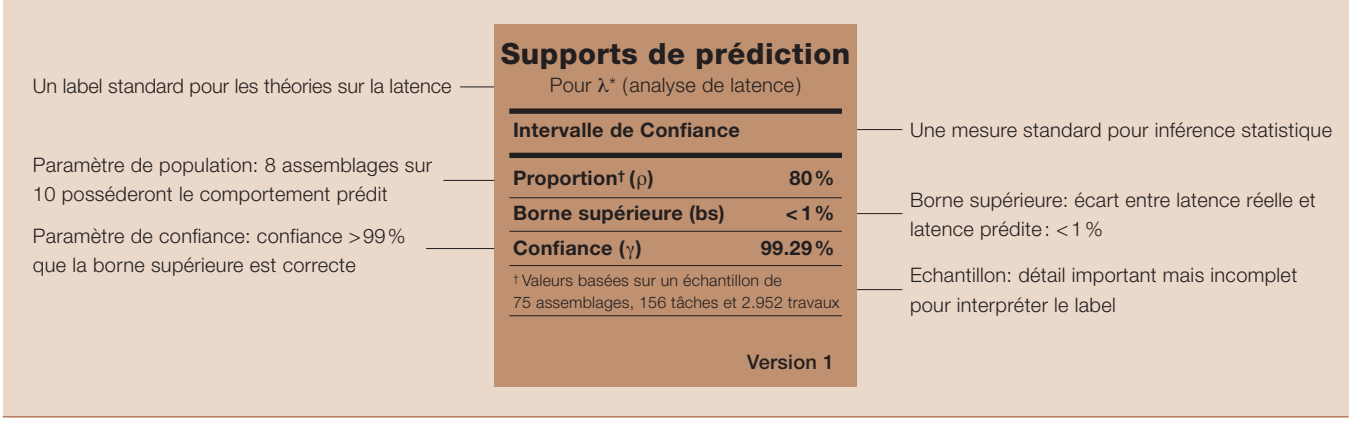
1) On a démontré comment la CEI 61850, norme du domaine de l'énergie électrique, pouvait s'adapter à la technologie des composants logiciels. Des composants et assemblages conformes CEI 61850 ont été spécifiés en langage CCL. En soi, il s'agit d'un résultat mineur mais qui constitue un point de départ pour rapprocher la norme des systèmes logiciels qui la mettent en œuvre.
2) Le cadre de raisonnement λaba («λ» pour latence, «aba» pour *average-case, with blocking and asynchronous interactions*) a été développé pour prédire le temps de latence moyen des assemblages de contrôle-commande et de protection normalisés CEI 61850. Le λaba utilise des contraintes intelligentes pour garantir la satisfaction d'un certain nombre d'hypothèses clés de l'analyse RM (*Rate Monotonic*) généralisée [5].

   Le mécanisme de la prédictibilité par construction est illustré en 2. Un assemblage CEI 61850 est représenté en notation graphique CCL (moitié supérieure de 2. Si l'assemblage est bien formé selon le λaba, une «vue des performances» analysable est construite (moitié inférieure de 2). A noter que CCL est basé sur des abstractions, bien connues des ingénieurs, qui utilisent les blocs de fonctions CEI 61131 ou UML. En effet, CCL peut être remplacé par un nombre quelconque de notations de conception.
3) Une technique a été présentée pour la validation empirique rigoureuse de la puissance prédictive du λaba ou de tout autre cadre de raisonnement de prédiction du comportement temporel [6]. Elle utilise la génération aléatoire d'assemblages sous contraintes pour construire des échantillons représentatifs dans un domaine d'application et, partant de là, bâtir des intervalles de confiance (ou de tolérance) statistiques utiles pour déduire la précision des prédictions futures. En réalité, ce label statistique sert à certifier le cadre de raisonnement lui-même. 3 et [7] décrivent le mode d'interprétation des labels statistiques.

ABB et le SEI ont appliqué les enseignements tirés de cette première expérience au domaine de la commande des robots industriels. Deux résultats importants découlent de ce travail.

Le premier est lié à la question suivante: un système matériel de commande périodique temps réel peut-il être complété, en toute sécurité, par un logiciel tiers à comportement d'exécution stochastique tout en garantissant à cette extension le meilleur fonctionnement? En résumé, un système matériel de commande temps réel peut-il être «ouvert» tout en continuant d'offrir des garanties solides sur le temps? C'est pour répondre à cette question que le cadre de raisonnement λss («λ» pour latence, «ss» pour serveur sporadique) fut développé. Le λss utilise un conteneur de serveur sporadique comme

3 Cadre de raisonnement certifié



contrainte intelligente centrale [8]. Ce conteneur protège les tâches périodiques des rafales stochastiques. Le cadre de raisonnement permet à tous les comportements périodiques d'être effectivement « réduits » à un seul effet périodique net qu'il utilise alors dans un ensemble d'équations de files d'attente pour borner le temps de fonctionnement moyen des modules d'extension logiciels (*plug-ins*) 4. Au final, on garantit que ces *plug-ins* ont une capacité d'intrusion bornée et prédictible sur le comportement temporel périodique, un accès garanti au processeur ainsi qu'une latence prédictible.

Le deuxième résultat montre à quel point les techniques de vérification de modèles logiciels (cf. glossaire page 54) ont évolué ces dernières années et que la marge potentielle de progrès est importante si ces techniques sont combinées efficacement au développement à base de composants. La vérification de modèles est particulièrement performante pour démontrer le comportement correct des logiciels concurrents et réactifs, type de logiciel qui prévaut en automatisation industrielle. Le code IPC (*Inter-Process Communication*) de l'armoire de commande d'un robot a été soumis à un vérificateur de modèles du commerce. Les propriétés vérifiées sont typiques du code IPC :

- Chaque fois qu'un message est envoyé à X, X reçoit ce message (sans *timeouts*),
- Chaque fois qu'un message est envoyé à X, Y ne reçoit jamais ce message,
- Chaque fois qu'un émetteur reçoit une réponse, il s'agit de la réponse au dernier message envoyé,
- Un émetteur n'est jamais bloqué lorsqu'il veut écrire un message dans une file d'attente non pleine,
- Les messages (ou les réponses) ne sont jamais écrits après déconnexion.

La vérification a mis en évidence une violation (ou « contre-exemple » dans le jargon de la vérification de modèles) de la troisième propriété. Les ingénieurs produit ont bel et bien confirmé l'existence d'un problème, diagnostiqué et résolu dans une version ultérieure du code. Or ce qui est frappant c'est que le problème, bien que suspecté, soit resté caché des années durant ! Rétrospectivement, cela n'est pas surprenant car les erreurs temps réel des logiciels sont notoirement difficiles à reproduire, ce que dénoncent certaines statistiques sur ce cas particulier de vérification de modèles.

- L'espace des états des seules parties pertinentes du code comportait $\cong 10^{1932}$ états, après abstraction !
- L'état erroné est apparu à la 179ème interaction entre les unités d'exécution communiquantes.

Ces chiffres montrent qu'une méthode classique de test aurait eu très peu de chance de détecter l'erreur.

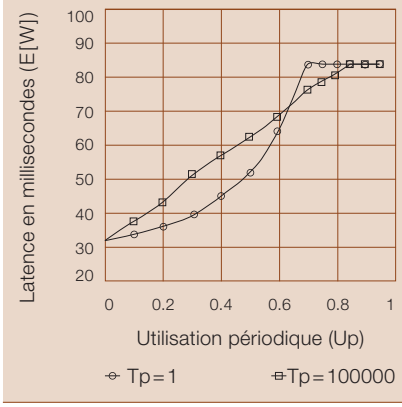
Depuis que ce travail a été réalisé, le cadre de raisonnement de vérification de modèles ComFoRT a connu des développements supplémentaires. ComFoRT exploite la technologie des composants logiciels et met en œuvre plusieurs techniques complémentaires les plus modernes de réduction des complexités [9] [11].

Le défi majeur lié à la tâche de vérification IPC – et auquel doit faire face toute personne utilisant la dernière génération de vérificateurs de modèles – est de produire des modèles qui sont des abstractions valides du système en question. Un atout clé de ComFoRT est la génération totalement automatique des modèles abstraits pertinents à partir des spécifications de conception CCL 5. C'est l'une des fonctionnalités de ComFoRT visant à simplifier notablement la vérification de modèles par les programmeurs et ingénieurs logiciel.

Résultats

L'assemblage prédictible ne prétend pas être la panacée des problèmes de qualité du logiciel. Il doit s'intégrer dans une stratégie globale de qualité basée sur des processus de développement logiciel à haut degré de maturité, des développeurs bien formés et motivés, des standards d'architecture correctement documentés et une culture de l'excellence. Partant de là, certaines conclusions peuvent néanmoins être formulées en ce qui concerne la viabilité de cette approche pour les domaines d'activité d'ABB.

4 Latence prédictible des *plug-ins*

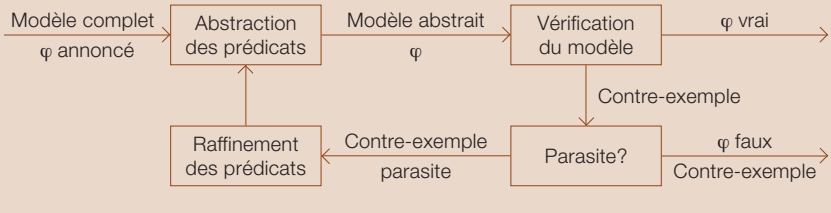


Conclusion principale : le bien-fondé et l'utilité du postulat de départ de ce travail – à savoir utiliser des contraintes intelligentes pour la prédictibilité et les encapsuler dans la technologie des composants. Ce postulat ne dépend pas d'un choix particulier de composant, ni de la spécification ou de la méthode d'analyse. Si toutes les méthodes n'ont pas la même efficacité, ce travail met en lumière ce qui est le plus important dans chacune d'elles si elles doivent servir le but de la prédictibilité par construction.

Cependant, avant de pouvoir adopter la technologie de prédictibilité par construction, deux conditions doivent être remplies :

1) L'équipe de développeurs doit avoir les coudées franches en ce qui concerne la définition de l'architecture du logiciel. En effet, des compromis sont nécessaires entre prédictibilité et liberté de conception ; une équipe qui n'est pas en mesure d'imposer ou d'assouplir des contraintes de conception ne pourra pas

5 Construction automatique de modèles ComFORT par raffinement des abstractions guidé par contre-exemples



prendre les décisions qui s'imposent. Les systèmes hérités du passé posent également des défis qui bridents la liberté de choix.

2) La qualité prédictible doit avoir une réelle valeur. Si cela peut paraître à première vue évident, c'est loin d'être le cas. Par exemple, certains systèmes intégrés de gestion sont caractérisés par une tolérance de conception «large» pour un aspect comportemental comme la latence. Des prédictions temporelles hautement précises peuvent ici ne pas avoir de grande valeur (même si la sécurité en a). Un contrôleur embarqué, toutefois, peut avoir une tolérance de conception «étroite» en ce qui concerne le comportement temporel.

L'élément le plus coûteux de l'infrastructure technique pour une prédictibilité par construction est le cadre de raisonnement validé. Les éléments comme les technologies des composants et les spécifications sont certes cruciaux, mais ne posent pas de réelles difficultés techniques. La conclusion est que les cadres de raisonnement deviendront probablement pour ABB des éléments d'actifs plus importants que les autres éléments et pourront jouer un rôle majeur dans la définition de standards de conception «intelligents» pour le Groupe.

Travaux en cours

Les travaux en cours se focalisent sur le développement d'un système logiciel prédictible de contrôle-commande et de protection appelé *Soft P&C* pour le domaine de l'automatisation des postes électriques. Un système *Soft P&C* est essentiellement un système complet d'automatisation de poste installé sur un ordinateur central, plus ou moins standard, sans matériel propriétaire. Un tel système est techniquement réalisable, mais convaincre les clients que la solution satisfait les exigences critiques de performance et de fiabilité ne suffit pas; encore faut-il en fournir la preuve, ce que font précisément les concepts présentés dans cet article.

Collaboration ABB-CMU

Les chercheurs d'ABB et du SEI travaillent ensemble sur la prédictibilité des assemblages de composants certifiables depuis 2001. Cette collaboration profite aux deux parties. D'une part, le CMU/SEI développe ses technologies en prise directe avec l'industrie et peut ultérieurement les adapter à une classe plus large de problèmes analogues. D'autre part, ABB peut exploiter très en amont des technologies émergentes pour prédire les attributs de qualité des systèmes logiciels complexes.

Dr Magnus Larsson
Dr Anders Wall
ABB AB, Corporate Research
Västerås (Suède)
magnus.larsson@se.abb.com

Kurt Wallnau
Software Engineering Institute
Carnegie Mellon University (USA)
Kcw@sei.cmu.edu

Glossaire

Trace d'exécution: séquence de modifications survenant dans un système ou composant observé dans le temps.

Vérificateur de modèles: outil informatique servant à vérifier si les modèles possèdent les propriétés désirées.

Vérification de modèles: analyse de toutes les traces d'exécution d'un système (partie matérielle ou logicielle) de manière exhaustive pour vérifier qu'il possède des propriétés données. Si l'analyse n'est pas exhaustive, c'est parce que l'abstraction analytique assure qu'il n'y a aucune autre information nouvelle (par exemple, du fait de symétries, de pertinences ou de répétitions).

Espace des états: un état est une valeur qu'un jeu de variables d'un système peut prendre en cours d'exécution. Par exemple, un interrupteur simple peut prendre deux états: ON ou OFF. On appelle espace des états l'ensemble des états qu'un système peut prendre. La taille de cet espace croît avec la complexité du système et, plus spécifiquement, exponentiellement avec le nombre de variables du système. Lorsqu'un système comporte un grand nombre de variables, le nombre d'états explose littéralement au point que le vérificateur de modèles n'est plus capable de les analyser dans un temps donné ou de les stocker. On parle alors d'explosion des états. Les vérificateurs de modèles utilisent de puissants mécanismes d'abstraction pour limiter l'*explosion des états*. Les plus performants peuvent vérifier dans un délai très court de très gros systèmes.

UML: (*Unified modeling language*) langage de modélisation objet unifié largement utilisé pour la description et la spécification des logiciels.

Bibliographie

[1] **Ward-Dutton, N**., "Containers: A sign components are growing up." Application Development Trends, pp 41–46, Jan 2000.
[2] **Lehoczky, J. P.,** "Real-time queuing theory," in Proceedings of the IEEE Real-Time Systems Symposium, 186–195, IEEE, New York, 1996.
[3] **Hissam et. al.,** *Predictable Assembly of Substation Automation Systems: An Experiment Report, Second Edition,* Technical Report CMU/SEI-2002-TR-031, www.sei.cmu.edu/publications/documents/02.reports/02tr031.html
[4] **Wallnau, K., Ivers, J.,** *Snapshot of CCL: A Language for Predictable Assembly*, Technical Note CMU/SEI-2003-TN-025, www.sei.cmu.edu/publications/documents/03.reports/03tn025.html
[5] **Klein et. al.,** *A Practitioner's Handbook for Real-Time Analysis: Guide to Rate Monotonic Analysis for Real-Time Systems,* Kluwer Academic Publishers, 1993.
[6] **Larsson M.,** Predicting Quality Attributes in Component-based Software Systems, Ph. D thesis Mälardalen University Press, ISBN 91-88834-33-6
[7] **Moreno, G., Hissam, S. Wallnau, K.** "Statistical Models for Empirical Component Properties and Assembly-Level Property Predictions: Toward Standard Labeling," in the 5th ICSE Workshop on Component-Based Software Engineering, May 2002, www.preview.sei.cmu.edu/pacc/CBSE5/Moreno-cbse5-final.pdf
[8] **Hissam et. al.,** *Performance Property Theories for Predictable Assembly from Certifiable Components,* Technical Report CMU/SEI-2004-TR-017, www.sei.cmu.edu/publications/documents/04.reports/04tr017.html
[9] **Ivers, J., Sharygina, N.,** *Overview of ComFoRT: A Model Checking Reasoning Framework,* Technical Note CMU/SEI-2004-TN-018, www.sei.cmu.edu/publications/documents/04.reports/04tn018.html
[10] **Meyer, B.** "The Grand Challenge of Trusted Components," 660–667. Proceedings of the 25th International Conference on Software Engineering (ICSE). Portland, Oregon, May 3-10, 2003. Los Alamitos, CA: IEEE Computer Press, 2003.
[11] **Clarke, E., Kroening, D., Sharygina, N., Yorav, K.,** "Predicate Abstraction of ANSI-C Programs Using SAT," in Formal Methods in System Design, 25, 105–127, 2004, Kluwer Academic Publishers.
[12] **Wallnau, K.,** *Software Component Certification: 10 Useful Distinctions,* Technical Note CMU/SEI-2004-TN-031, http://www.sei.cmu.edu/publications/documents/04.reports/04tn031.html
[13] **Nierstrasz, O., et al,** "A Component Model for Field Devices" Proceedings First International IFIP/ACM Working Conference on Component Deployment, ACM, Berlin, Germany, June 2002. Cf. également http://www.pecos-project.org/