Un processus à base de modèles pour les systèmes auto-adaptatifs

Mots clés

Systèmes auto-adaptatifs, Modélisation/UML, Conception, Simulation

Franck CHAUVEL[1], Olivier BARAIS[3], Jean-Marc JEZEQUEL[3], Isabelle BORNE[2]
*Key Laboratory of High Confidence Software Technologies, Ministry of Education, School of Electronics Engineering & Computer Science, Peking university, (*Beijing*, Chine)[1], VALORIA, université de Bretagne-Sud, centre de recherche Yves Coppens[2], IRISA, université de Rennes 1[3]*

Si aujourd'hui les systèmes auto-adaptatifs, capables de s'adapter à leur environnement, font légion, la conception des couches logicielles associées reste pour le moins sommaire. Dans la plupart des cas, les boucles de retro-actions nécessaires pour maintenir le système dans un état cohérent vis-à-vis de son environnement ne font pas partie des problématiques de conception et sont implémentées de manière *ad hoc* lors du développement. Cela génère invariablement de coûteux retours arrière dans le processus de développement qui pourraient être évités en intégrant les problématiques d'adaptation dès la phase de conception.

1. Introduction

Avec l'avènement des systèmes mobiles, la plupart des systèmes logiciels doivent s'adapter à un environnement fluctuant en ressources matérielles et logicielles. Ce besoin d'adaptabilité survient à la fois dans des systèmes à large échelle tels que les systèmes paire-à-paire déployés sur Internet et dans des systèmes de taille plus modeste, voire dans des systèmes embarqués, téléphonie mobile, GPS, etc. Ces systèmes, dits « auto-adaptatifs », doivent donc observer les fluctuations de leur environnement dans le but d'offrir un service de qualité le plus longtemps possible. On utilise généralement des boucles de contrôle (observation/décision/action) inspirées de la théorie du contrôle pour implémenter l'auto-adaptation.

Pour mettre en place ces boucles de contrôle, il faut disposer de sondes et d'actuateurs permettant, d'une part, de mesurer les fluctuations de l'environnement et d'autre part (pour les actuateurs), d'agir sur le système. Ces sondes et ces actuateurs doivent alors être intégrés proprement à l'architecture du système. Cependant, la conception de boucles de contrôle ne fait pas partie du processus de développement logiciel. L'adaptation est un problème qui n'apparaît qu'en phase de développement et dont les solutions *ad hoc* sont généralement implémentées à la main.

La contribution de cet article est un processus de développement dédié aux systèmes adaptatifs. Il s'agit de prendre en compte, dès les premières phases de conception, les problématiques extra-fonctionelles liées à l'adaptation en

L'ESSENTIEL

Cet article propose un processus de développement logiciel dédié aux systèmes auto-adaptatifs. Il combine la conception des sondes et des actuateurs à la conception de politiques d'adaptation nécessaires pour maintenir l'architecture et la configuration du système en adéquation avec son environnement. Basés sur les récentes avancées en matière de métamodélisation exécutable, les modèles générés au cours de ce processus sont simulables et permettent ainsi une première validation des politiques d'adaptation comme nous le montrons sur un système de décodage de flux vidéos.

SYNOPSIS

In this article we describe a model-driven development process devoted to self-adaptive systems. It combines the design of probes and actuators with the design of adaptation policies which can be properly integrated in the software architecture model. The use of new executable meta-modelling techniques enables an early validation of adaptation policies through simulations. This software development process is illustrated on a video-streaming decoder which needs to adjust its architecture and its configuration with respect to the bandwidth evolution.

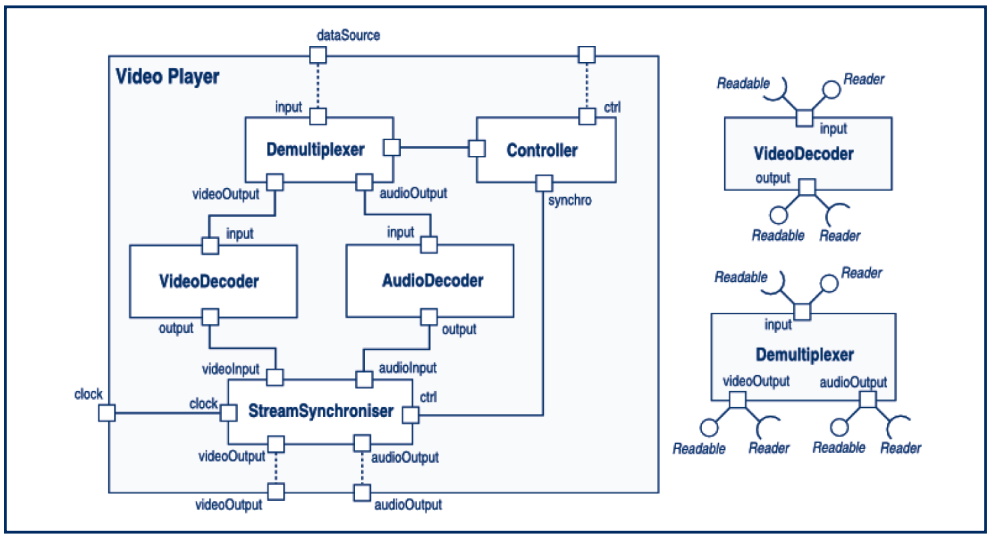

Figure 1. Architecture du lecteur vidéo représentée à l'aide de la notation UML 2.0 [1].

intégrant proprement les sondes et les actuateurs adéquats ; ce qui permet de concevoir et de valider le comportement adaptatif du système par le biais de simulation.

La suite de cet article illustre tout d'abord le problème actuel de conception des systèmes adaptatifs sur le cas d'un lecteur de flux vidéo. Le processus que nous proposons, présenté dans la section 3, repose sur l'utilisation d'un sous-ensemble d'UML (*Unified Modeling Language*) permettant de modéliser la structure, les données, les traitements et l'adaptation au sein d'une architecture à composants. L'ajout d'une sémantique opérationnelle à ces modèles permet de les simuler comme le montre la section 5 qui présente les résultats obtenus dans le cadre du lecteur de flux vidéo. Les sections 6 et 7 terminent cet article en présentant des travaux connexes du domaine et quelques perspectives pour de futures améliorations.

2. Le décodage de flux vidéo[1], un exemple caractéristique

Pour illustrer plus clairement les motivations qui ont guidé notre travail, prenons l'exemple du système de décodage vidéo présenté par la figure 1. L'architecture de ce système, décrite à l'aide de la notation UML, suit le principe général de décodage d'un flux vidéo :

1) le flux vidéo initial est démultiplexé, c'est-à-dire que les informations audio et vidéo sont séparées en deux flux d'information distincts ;
2) les deux flux audio et vidéo sont ensuite décodés séparément par des composants dédiés, appelés "*Decoder*" sur la figure 1 ;
3) les informations audio et vidéo sont ensuite synchronisées pour être jouées sur les périphériques *ad hoc* (haut-parleur et écran).

Dans cet exemple, le concepteur veut maximiser la qualité perçue par l'utilisateur en corrélant l'architecture du système avec la bande passante disponible. En effet, la qualité perçue par l'utilisateur dépend de la vitesse d'acquisition du flux vidéo initial et du temps de traitement induit par les différents décodages. Ainsi, lorsque la vitesse d'acquisition diminue, le système doit être capable d'adapter son architecture pour minimiser le temps de traitement et assurer, dans la mesure du possible, une qualité perçue constante.

Pour mettre en place ce procédé d'adaptation, il est nécessaire de mesurer concrètement les grandeurs mises en jeu. La qualité perçue par l'utilisateur peut être mesurée à l'aide *du frame-rate*[2] et la vitesse d'acquisition du flux vidéo peut se ramener à une mesure de la bande passante disponible. De manière similaire, le système doit disposer de services réduisant directement ou indirectement le temps de traitement du flux vidéo. On peut, entre autres, augmenter le taux de perte lors de l'encodage des images, utiliser des images en niveau de gris, ou bien changer de système d'encodage (passer d'un flux MPEG2 à un flux MPEG4, par exemple).

Cet exemple donne un aperçu des dépendances possibles entre les choix architecturaux et les capacités d'adaptation. Ces dépendances rendent très délicate la tâche du concepteur car il lui est impossible d'anticiper la pertinence de ses choix architecturaux. Cela conduit généralement à de coûteux retours arrière dans le processus de développement pour mettre les choix architecturaux en adéquation avec les choix relatifs aux capacités d'adaptation.

Le processus de développement présenté dans la suite de cet article permet de prendre en compte la problématique de l'adaptation dès la conception du système et contribue à minimiser les retours arrière dans le processus de développement.

[1] Plus généralement désigné par l'anglicisme « video-streaming ».

[2] « frame-rate », soit le nombre d'images traitées par unité de temps.

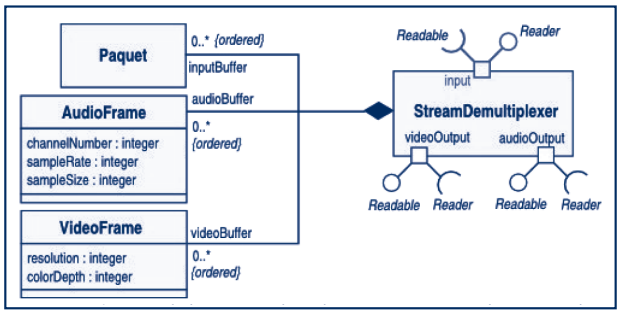

Figure 2. Modélisation des données manipulées par le composant "Synchronizer".

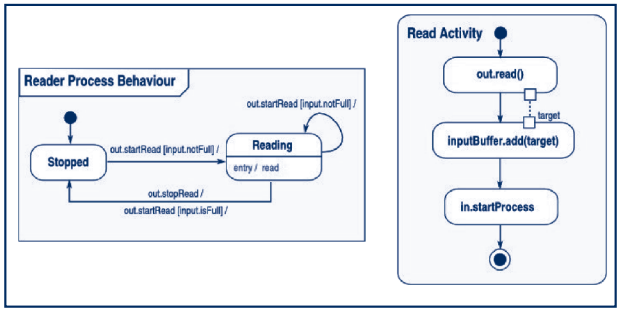

Figure 3. Extrait de la vue comportement du composant "Demultiplexer".

3. Processus de développement

Le processus de développement que nous proposons se déroule en cinq étapes qui aboutissent à une conception complète intégrant les problématiques liées à l'auto-adaptation à l'architecture du système. Ces cinq étapes sont les suivantes :

1) une architecture de base est définie pour répondre aux besoins fonctionnels. Dans l'exemple du lecteur vidéo, il s'agit de concevoir, par exemple, l'architecture présentée par la figure 1 qui est indépendante des besoins en adaptation ;
2) ensuite, l'architecte doit injecter dans sa conception initiale les sondes (resp. les actionneurs) permettant de mesurer (resp. de modifier) les grandeurs mises enjeu par l'adaptation. Dans le cas du système vidéo, il s'agit de la bande passante, du *frame-rate*, du taux de perte lors de l'encodage des images, etc. Dans le monde UML, les sondes et les actionneurs apparaissent généralement sous la forme de services ou de composants dédiés ;
3) une fois que toutes les informations nécessaires sont accessibles sous la forme de sondes et d'actionneurs, le concepteur doit mettre en relation des grandeurs mesurées et les actions à faire sur le système. Pour cela, le concepteur peut utiliser les machines à états d'UML et mettre ainsi en relation les évènements produits par les sondes et les actuateurs ;
4) les politiques d'adaptations complètent et finalisent ainsi la description du système auto-adaptatif. La validation des politiques d'adaptation vis-à-vis de l'architecture du système devient dès lors possible avec l'utilisation de modèles exécutables dotés d'une sémantique opérationnelle. Le concepteur peut alors concevoir les scénarios de test et observer un comportement *a priori* du système par le biais de la simulation ;
5) une fois que l'architecture est en adéquation avec les problématiques d'adaptation, le concepteur peut générer des squelettes de code à la fois pour l'application et les cas de test.

Ce processus de développement est illustré par la suite, à l'aide de la notation UML2 mais en reste indépendant et peut s'appliquer à d'autres outils de modélisation.

4. Modélisation exécutable de systèmes auto-adaptatifs

4.1. Modélisation d'architectures logicielles

Pour mettre en œuvre ce processus de développement, nous avons sélectionné un sous-ensemble minimal du langage UML que nous avons doté d'une sémantique opérationnelle. Ce sous-ensemble tient compte de trois points de vue sur l'architecture logicielle : la structure, les données et les traitements.

- La structure reprend les concepts d'UML utilisés pour décrire un assemblage de composants. Comme l'a montrée la figure 1, elle décrit des composants dont les ports sont reliés par des connecteurs. Chaque port fournit ou requiert un certain nombre d'interfaces définissant un ensemble de services.
- Les données manipulées et stockées par les composants sont représentées à l'aide des concepts du diagramme de classe d'UML2. Ces données sont nécessaires pour typer les services exposés par les composants sur leurs ports. La figure 2 montre les données utilisées par le composant "Demultiplexer" ; elles sont utilisées pour décrire les paramètres des services qu'il expose. Ici, le composant contient trois tampons destinés à contenir les paquets en entrée, les trames audio et les trames vidéo.
- Les traitements métiers internes aux composants sont modélisés à l'aide des concepts du diagramme d'activités d'UML. Ils décrivent le flot de données à l'intérieur des composants. La coordination entre les appels aux services exposés et les activités qu'ils contiennent est modélisée à l'aide de machines à états ("*Statecharts*" dans la terminologie UML). Chaque transition met alors en relation l'appel d'un service exposé sur un port du composant avec le déclenchement d'une de ses activités internes.

4.2. Intégration de sondes/actuateurs extra-fonctionnels

Comme évoqué à propos du processus de développement, le concepteur doit injecter dans l'architecture du système les différentes sondes et actuateurs qui sont

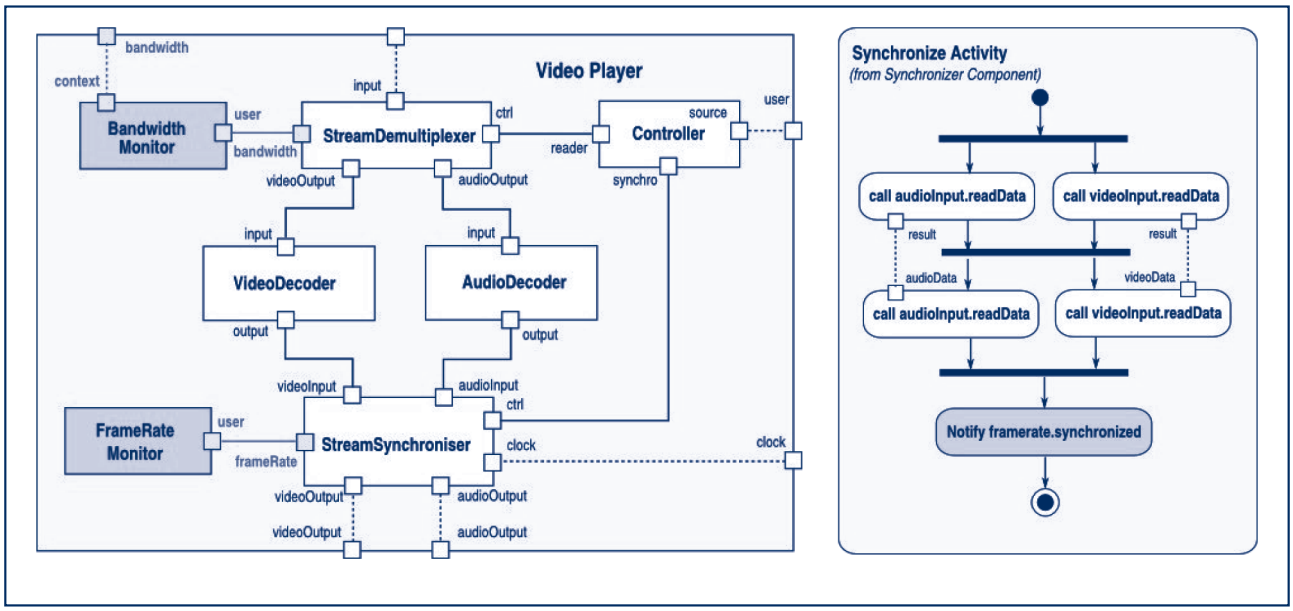

Figure 4. Intégrations structurelle et comportementale de sondes extra-fonctionnelles.

nécessaires pour mettre en place l'adaptation. Au niveau structurel, les sondes et les actuateurs sont généralement réifiés sous la forme de composants connectés dans l'architecture. Ils peuvent également être réifiés sous la forme de services exposés par certains composants. La figure 4 montre dans sa partie gauche les modifications apportées à la structure de l'architecture pour intégrer deux sondes : la première permet de mesurer la bande-passante disponible et est connectée au démultiplexer. La seconde permet de calculer le *frame-rate* et est connectée au composant "Synchronizer".

Cependant, il reste nécessaire d'intégrer également les sondes au niveau comportemental. Il s'agit de refléter les effets de bords du comportement fonctionnel sur le comportement extra-fonctionnel et vice et versa. Par exemple, lorsqu'une trame audio a été correctement synchronisée avec une trame vidéo, il faut notifier la sonde qui mesure le frame-rate[3]. La partie droite de la figure 4 montre les modifications apportées au comportement du composant "Synchronizer".

4.3. Modélisation des politiques d'adaptation

La modélisation des politiques d'adaptation se fait en utilisant des machines à états où les évènements sont émis par les sondes extra-fonctionnelles et les actions, des appels aux services offerts par les actuateurs. Chaque transition met donc en relation les données observées par les sondes et les actuateurs. Dans l'exemple de la figure 5, lorsque la bande passante commence à diminuer, on augmente le taux de perte des images lors de leur encodage. Si la bande passante continue de chuter, on utilise alors des images en niveau de gris et, finalement, lorsque la bande passante devient mauvaise, on utilise un flux MPEG4, moins consommateur en ressources.

C'est le composant composite qui est responsable de l'exécution et de la coordination des politiques d'adaptation. En effet, c'est lui qui connaît les différents éléments qui le constituent et il peut décider de modifier ou de remplacer tel ou tel sous-composant. A ce titre, il définit également la portée des politiques d'adaptation dans la mesure où une politique d'adaptation ne doit pas accéder à des éléments internes aux sous-composants. Les machines à états responsables de l'adaptation sont donc décrites au sein du composant composite (le lecteur vidéo) dans les figures 1 et 4.

4.4. Un outil de modélisation exécutable

Le sous-ensemble d'UML, présenté dans les paragraphes précédents, est suffisamment détaillé pour pouvoir être exécuté. Il faut cependant pouvoir décrire la sémantique opérationnelle associée aux constructions d'UML qui ont été retenues : machines à états, activités, composants, etc. Cette sémantique permettra d'animer les modèles en les confrontant à des scénarios de test.

Sur le plan technique, le sous-ensemble d'UML est capturé par un méta-modèle décrit dans le langage Kermeta [2]. Conçu comme une extension de eMOF [3] à l'aide d'un langage d'action minimal, Kermeta permet de décrire la sémantique opérationnelle d'un méta-modèle de manière impérative et ainsi d'en animer les modèles instances. Kermeta nous a permis de décrire une sémantique possible pour le sous-ensemble d'UML choisi pour les systèmes auto-adaptatifs. Cette sémantique permet de simuler, *a priori*, les modèles de conception et donc de faire une première validation des choix de conception.

[3] On considère ici que le "*frame-rate*" est calculé à partir du temps moyen écoulé entre la synchronisation de deux trames vidéo.

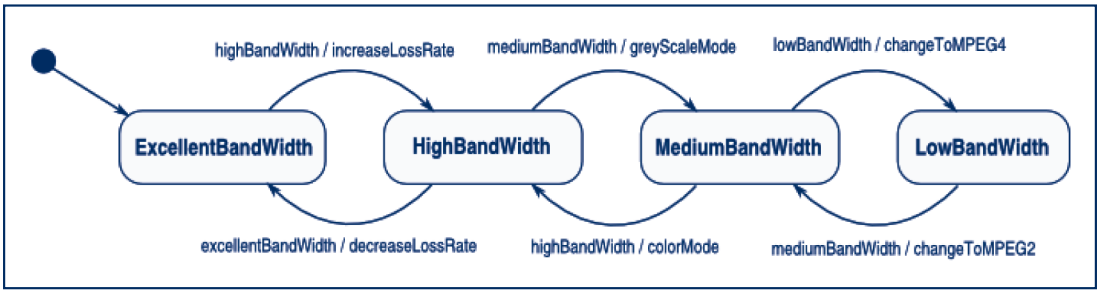

Figure 5. Exemple de politique d'adaptation contrôlant l'évolution de la bande passante disponible.

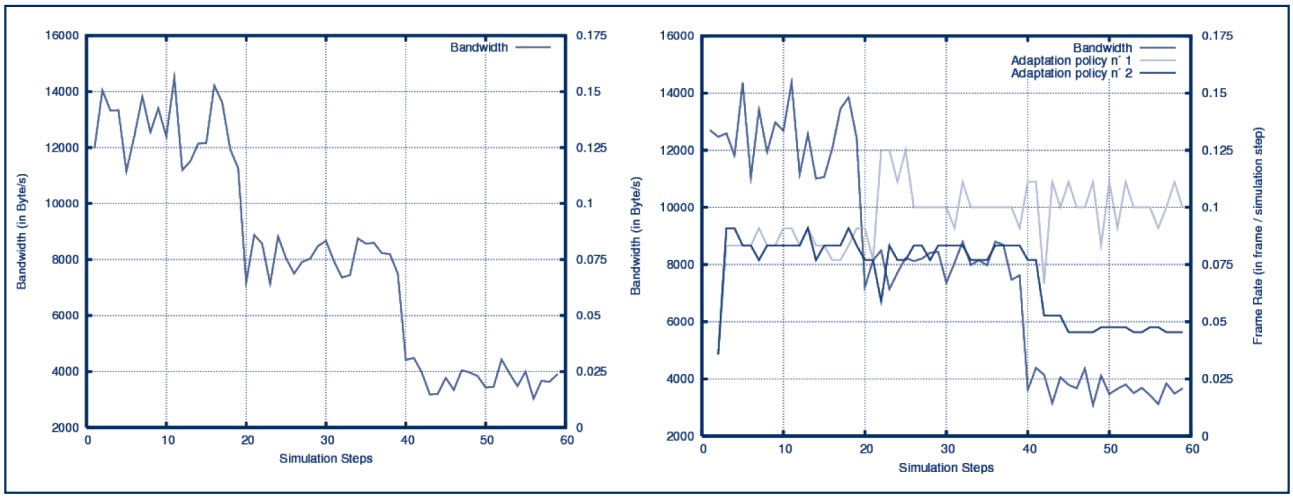

Figure 6. Résultats de simulation pour deux politiques d'adaptation.

5. Simulation *a priori* de systèmes auto-adaptatifs

L'objectif du processus de simulation présenté dans cet article est de permettre au concepteur de valider *a priori* les politiques d'adaptation vis-à-vis des choix architecturaux qui ont été faits en amont. La validation et la vérification *a posteriori* restent naturellement nécessaires.

Considérons, par exemple, une variante de la politique d'adaptation présentée par la figure 5, mais où l'on n'utiliserait pas de décodeur MPEG4. Un tel changement nécessite une modification architecturale « à chaud » du système (déchargement d'un composant logiciel), ce qui peut être coûteux ou impossible selon la plate-forme visée pour le déploiement. Une variante sans modification architecturale peut permettre un déploiement sur une plate-forme ne supportant pas les modifications architecturales à l'exécution. La simulation *a priori* permet de déterminer les performances respectives de ces deux politiques d'adaptation possibles.

On se place dans un scénario où la bande passante passe d'une valeur élevée à une valeur faible par paliers. Ce scénario est présenté par la partie gauche de la figure 6. Dans la partie droite, on peut voir le "*frame-rate*" obtenu lorsque ces deux politiques d'adaptation sont déployées. Il s'avère ainsi que lorsque la bande passante reste moyenne, les deux politiques ont des performances équivalentes. Ainsi, pour un déploiement visant une plate-forme d'exécution très limitée en ressources, la variante sans modification architecturale peut s'avérer tout à fait satisfaisante.

6. Travaux connexes

Balsamo *et al.* [4] présentent les approches possibles pour évaluer les performances non-fonctionnelles de modèles UML. Ces approches reposent sur des outils mathématiques : réseaux de files d'attente, réseaux de Petri stochastiques, simulations numériques. Par mesure de concision, nous ne présentons ici que les plus pertinentes.

Pour décrire les aspects extra-fonctionels, l'OMG fournit deux profils UML : SPT [5] et QoS [6]. Le premier permet d'annoter les modèles UML avec des informations relatives au temps et à la performance. Le second, plus général, permet de définir une grande variété d'annotations extra-fonctionelles. Les annotations peuvent ensuite être analysées à l'aide d'outils dédiés pour obtenir des prédictions de performances comme le font Shen et Petriu [7]. L'idée de base, qui est de dériver des modèles formels à partir des annotations, n'est pas applicable dans le cas des systèmes adaptatifs, puisque les prédictions de performance doivent être accessibles dans le modèle pour mettre en place les boucles de contrôle.

Se basant sur ROOM, V. Cortellessa [8] réifie les ressources disponibles et leur consommation dans l'architec-

ture pour obtenir des prédictions de performance. Notre approche s'inspire de ces travaux et va plus loin en proposant de réifier toutes les propriétés extra-fonctionelles du système, et ainsi de permettre la mise en place des sondes, des actuateurs et des politiques d'adaptations qui s'y rattachent.

Par ailleurs, de nombreux langages de description d'architecture (ADL) sont apparus dans la littérature : ACME [9], -ADL [10], Wright [11], Rapid [12], etc. La plupart permettent de modéliser la structure du système mais ne se focalisent pas sur le comportement. Ceux qui en sont capables ne prennent pas en compte les problématiques extra-fonctionnelles, et ne sont donc pas réellement des outils appropriés pour la conception de systèmes auto-adaptatifs.

D'autres travaux, répertoriés par Bradbury *et al.* dans [13], portent cependant sur les systèmes auto-adaptatifs et le problème de la reconfiguration dynamique. Parmi ces travaux, on peut noter ceux de Allen *et al.* [14] qui proposent une extension de Wright avec des primitives de reconfiguration architecturale, mais elle n'intègre pas de problématiques extra-fonctionnelles. Batista *et al.* [15] proposent une extension d'ACME pour spécifier des reconfigurations dynamiques, qui seront utilisées à l'exécution pour piloter un intergiciel réflexif. Cette approche ne permet pas non plus de faire référence à des propriétés extra-fonctionnelles dans les politiques d'adaptation.

7. Conclusion

Ainsi, l'intégration de capacités d'auto-adaptation dans un système logiciel ajoute de nombreuses exigences architecturales supplémentaires qu'il faut prendre en compte dès le début du processus de développement pour éviter tout conflit avec les exigences fonctionnelles. Actuellement, la conception des boucles de contrôle nécessaires à la mise en place de l'auto-adaptation est, au mieux, repoussée lors de l'implémentation, ce qui amène souvent à de coûteux retours arrière dans le processus de développement.

Pour éviter cela, le processus de développement proposé dans cet article permet de prendre en compte les exigences d'adaptation dès le début du processus de développement. Il s'agit, dans un premier temps, de concevoir les sondes et les actuateurs nécessaires pour détecter les changements de l'environnement et maintenir le système dans un état et/ou une configuration adéquate. Une fois l'architecture dotée de ces mécanismes de base, il est ensuite possible pour le concepteur d'exprimer sous la forme de machines à états le comportement adaptatif voulu. De plus, l'utilisation de modèles exécutables, dotés d'une sémantique opérationnelle, permet de simuler les modèles et d'avoir un aperçu, *a priori,* des performances des politiques d'adaptation mises en place.

Si l'intégration des sondes extra-fonctionnelles dans l'architecture initiale reste manuelle, ce processus est une première étape vers un système de conception et de réalisation dédié aux systèmes auto-adaptatifs. De nombreux points peuvent être améliorés comme la description des scénarios de test qui mêlent des aspects fonctionnels et extra-fonctionnels, ou l'utilisation de machines à états pour décrire les politiques d'adaptation.

Références

[1] OMG, "*OMG Unified Modeling Language (OMG UML), Superstructure*", V2.1.2, 2007.

[2] P.-A. MULLER, F. FLEUREY, J.-M. JEZEQUEL, "*Weaving Executability into Object-Oriented Meta-Languages*", Jamaica, 2005.

[3] OMG, "*Meta Object Facility (MOF) Core Specification*", 2006.

[4] S. BALSAMO, A. DI MARCO, P. INVERARDI, M. SIMEONI, "*Model-Based Performance Prediction in Software Development: a survey*", 2004.

[5] Object Management Group, "*UML Profile for Schedulability, Performance and Time Specification*", 2003.

[6] Object Management Group, "*UML Profile for Quality of Service and Fault Tolerance Characteristics and Mechanisms*", 2004.

[7] D.C. PETRIU, H. SHEN, "*Applying the UML Performance Profile: Graph Grammar Based Derivation of LQN Models from UML Specifications*", 2002.

[8] V. CORTELLESSA, R. MIRANDOLA, "*Deriving a Queueing Network Based Performance Model from UML Diagrams*", New York, NY, USA, 2000.

[9] D. GARLAN, R.T. MONROE, D. WILE, "*Foundations of Component Based Systems*", 2000.

[10] C. CHAUDET, F. OQUENDO, "*piSPACE: a Formal Architecture Description Language Based Onprocess Algebra for Evolving Software Systems*", 2000.

[11] R. ALLEN, D. GARLAN, "*A Formal Basis for Architectural Connection*", 1997.

[12] D.C. LUCKHAM, J.J. KENNEY, L.M. AUGUSTIN, J. VERA, D. BRYAN, W. MANN, "*Specification and Analysis of System Architecture Using Rapide*", 1995.

[13] J.S. BRADBURY, J.R. CORDY, J. DINGEL, M. WERMELINGER, "*A Survey of Self-Management in Dynamic Software Architecture Specifications*", 2004.

[14] R. ALLEN, R. DOUENCE, D. GARLAN, "*Specifying and Analyzing Dynamic Software Architectures*", 1998.

[15] T. BATISTA, A. JOOLIA, G. COULSON, "*Managing Dynamic Reconfiguration in Component Based Systems*", 2005.

Les auteurs

Franck Chauvel a obtenu son doctorat en 2008 à l'université de Bretagne-Sud. Depuis novembre 2008, il effectue un post-doctorat à l'université de Pékin. Ses recherches visent à améliorer les techniques de développement de systèmes logiciels auto-adaptatifs en intégrant des outils issus de l'intelligence artificielle aux procédés du génie logiciel : ingénierie dirigée par les modèles (IDM) et architecture logicielle à base de composants (CBSE) principalement.

Olivier Barais a été diplômé ingénieur de l'école des Mines de Douai en 2002. Il a obtenu un doctorat de l'université de Lille 1 en 2005. Depuis septembre 2006, il est maître de conférences à l'université de Rennes 1 et membre de l'équipe INRIA Triskell. Ses champs de recherche concernent l'ingénierie dirigée par les modèles (IDM ou *model-driven software engineering*), la construction d'architecture logicielle à base de composants et la modélisation par aspects.

Isabelle Borne a obtenu son doctorat en 1984 et son habilitation à diriger les recherches en 1995 à l'université Pierre-et-Marie-Curie (Paris 6). De 1987 à 1994, elle a été maître de conférences à l'université René Descartes (Paris 5), puis professeur à l'école des Mines de Nantes. Depuis septembre 2001, elle est professeur à l'université de Bretagne-Sud et anime l'action « Architecture logicielle et Composants Contractualisés » du laboratoire de recherches VALORIA. Ses domaines d'intérêt couvrent l'ingénierie dirigée par les modèles et les lignes de produits logiciels.

Jean-Marc Jézéquel est professeur à l'université de Rennes I. Au sein de l'IRISA, il dirige l'équipe Triskell qui est un projet de recherche de l'INRIA sur l'ingénierie dirigée par les modèles pour les composants logiciels. Il est l'auteur de plusieurs livres ainsi que de plus de 100 publications dans des revues et conférences internationales. Auparavant, il a été ingénieur dans l'industrie des télécoms, puis chercheur au CNRS.