# Reconfiguration d'architecture logicielle : la nature des communications entre composants

Ngoc Tho Huynh, Maria-Teresa Segarra, Antoine Beugnard

# Reconfiguration d'architecture logicielle : la nature des communications entre composants

Ngoc-Tho Huynh, Maria-Teresa Segarra et Antoine Beugnard

IRISA / Université Européenne de Bretagne / TELECOM Bretagne, Brest, France
{tho.huynh, mt.segarra, antoine.beugnard}@telecom-bretagne.eu

**Résumé**

Un problème important de la reconfiguration dynamique des systèmes adaptatifs est d'assurer l'intégrité des données, et en particulier, le processus de reconfiguration doit prendre en compte les requêtes en attente de traitement. Ce problème dépend (entre autres) de la capacité du serveur à stocker des requêtes en attente de traitement. Dans le cas où le serveur ne peut pas stocker ces requêtes et en fonction de si le client peut changer son implantation ou pas (conscient ou non de la reconfiguration), l'architecture du logiciel après reconfiguration n'est pas la même. Dans cet article, nous présentons un cas d'étude pour mettre en évidence l'importance de la prise en compte de ces deux aspects dans la reconfiguration.

**Mots-clés:** modèle, variabilité, composant, communication, reconfiguration.

## 1 Introduction

L'architecture logicielle d'une application est le résultat d'un processus de développement. Dans l'ingénierie dirigée par les modèles, le processus manipule et transforme des modèles de conception d'application. Dans une ligne de produits, un des modèles de conception est le modèle de variabilité. Ce modèle est utilisé pour spécifier la partie commune et la variabilité des applications dérivées du modèle. Des éléments communs sont présents dans tous les produits et des éléments variables représentent les différences entre les produits [2]. La différence est spécifiée par des variantes dans le modèle de variabilité. La variabilité à la conception nous permet de configurer plusieurs produits différents. Plusieurs travaux utilisent actuellement des techniques issues des lignes de produits pour développer des logiciels adaptatifs comme dans [7], [3] et [1] c'est-à-dire des logiciels dont le comportement peut changer à l'exécution. Ces travaux utilisent le modèle de variabilité des lignes de produits pour spécifier la variabilité du logiciel. Le mapping entre l'architecture de la ligne de produits et le modèle de variabilité est spécifié. À partir des choix des éléments dans le modèle de variabilité (configuration de la ligne), une configuration de l'architecture est déduite. Cette configuration peut être réalisée avant l'exécution du produit ou pendant son exécution. Dans ce second cas, la nouvelle configuration peut être calculée à partir du modèle de variabilité et de l'architecture et des différences avec la configuration actuelle.

Un problème important de la reconfiguration dynamique des logiciels adaptatifs est d'assurer l'intégrité des données, et en particulier, le processus de reconfiguration doit prendre en compte les requêtes en attente de traitement. Ce problème dépend (entre autres) de la capacité du serveur à stocker des requêtes en attente de traitement. Dans le cas où le serveur ne peut pas stocker ces requêtes et en fonction de si le client peut changer son implantation ou pas (conscient ou non de la reconfiguration), l'architecture du logiciel après reconfiguration n'est pas la même. L'objectif de cet article est de mettre cela en évidence et de proposer un début de solution. Le papier est organisé comme suit. Le paragraphe 2 introduit des connaissances de base nécessaires à la compréhension de notre étude de cas. Le paragraphe 3 présente le cas de reconfiguration que nous considérons ainsi que trois de ses déclinaisons et une comparaison critique de celles-ci. Enfin, le paragraphe 4 conclut l'article.

## 2 Connaissances de base et hypothèses

Comme abordé dans la section précédente, afin de développer un logiciel adaptatif, un modèle de variabilité est utilisé pour spécifier la partie commune et la variabilité. Dans cet article, nous utilisons CVL [4] et les concepts associés pour illustrer notre propos. Dans CVL, le modèle de variabilité est représenté par un modèle de fonctionnalité [5]. L'architecture de la ligne de produits est appelée modèle de base et peut être spécifiée en utilisant un langage conforme à MOF[1], comme UML. Le mapping entre le modèle de base et de variabilité se fait par des points de variation qui indiquent le changement dans l'architecture lors de la modification de la configuration du modèle de variabilité. C'est à partir de ces points de variation que les actions de reconfiguration sur le modèle de base peuvent être générées.

Le modèle de base de notre approche est spécifié comme une architecture de composants. Les composants communiquent ensemble grâce à des connexions. Dans une connexion entre deux composants, un composant joue le rôle de fournisseur de services (un serveur) via une ou des interfaces et l'autre joue le rôle de demandeur de services (un client) via des dépendances. Nous considérons un exemple simple avec un unique client et un unique serveur. Le serveur peut être de deux types : avec tampon (SaT) ou sans tampon (SsT). Le client peut être synchronisé, et ne faire qu'une requête à la fois (CS) ou asynchrone (CA) et faire potentiellement plusieurs requêtes sans attendre de réponse.

Le cas d'utilisation de la section suivante met en évidence les conséquences sur le client d'une reconfiguration d'un serveur avec tampon vers un serveur sans tampon. L'autre cas ne présente pas de difficulté architecturale.

## 3 Etude de cas : d'un SaT à un SsT

Dans ce paragraphe, nous présentons un cas d'étude de reconfiguration simple qui nous permet d'illustrer notre propos. L'application démarre avec SaT comme composant serveur. Pendant son exécution, le SaT doit être remplacé par un SsT. Selon le type du client (CS ou CA), nous allons présenter les contraintes de reconfiguration qui pèsent sur le client ou la connexion entre le client et le serveur. Nous verrons qu'il y a trois cas pertinents.

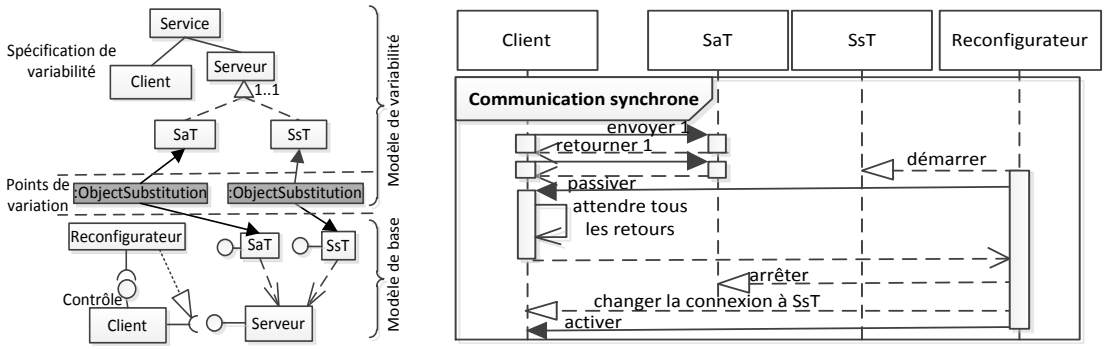
### 3.1 Le client est synchrone et reste synchrone

Dans ce cas, la communication entre le client est toujours synchrone. Ce cas est représenté par le modèle de variabilité et le modèle de base de la figure 1a. Le modèle de variabilité indique que l'application est constituée de deux composants : un client et un serveur (le haut de la figure 1a). Deux implémentations sont possibles pour le serveur. SaT correspond à un composant de type serveur qui maintient un tampon pour sauvegarder les demandes. SsT correspond à celui de type serveur qui ne maintient pas de tampon. Les points de variation *ObjectSubstituation* (les parties grises) permettent de spécifier les variantes différentes du même composant - *Serveur*. Le modèle de base (en bas de la figure 1a) représente le modèle d'architecture de composants en UML. Une interface de contrôle est définie dans le composant client pour contrôler la reconfiguration. Celle-ci est gérée par le composant *Reconfigurateur*.

La figure 1b représente les actions de reconfiguration réalisées par le *Reconfigurateur* pour modifier l'architecture de l'application. Les flèches (- - - -▷) correspondent à des actions de base fournies par la plate-forme d'exécution des composants. Pour ces actions, aucune interface de contrôle explicite n'est nécessaire dans le modèle de base.

Pour que la reconfiguration préserve l'intégrité des données, SaT doit être en état *quiescent* [6]. Pour ce faire, le *Reconfigurateur* demande au client de se mettre en état *passif*. Lorsqu'il

[1] http://www.omg.org/mof/




(a) Modèles (b) Actions de reconfiguration

Figure 1: Type de communication synchrone

atteint cet état, le *Reconfigurateur* est certain que le SaT est en état *quiescent*. Ensuite, le *Reconfigurateur* arrête le SaT et change la connexion du client. Enfin, il active le client.

On constate dans ce cas simple que le client doit offrir un service spécifique de passivation pour la reconfiguration.

## 3.2 Le client est asynchrone et est conscient de la reconfiguration

Dans ce cas, la communication entre le SaT et le client est de type asynchrone. Après la reconfiguration, le client sera connecté au SsT qui ne peut pas mémoriser les requêtes en attente de traitement. Le choix ici pour éviter la perte de données est de modifier le type de communication d'asynchrone à synchrone. Cette exigence est spécifiée par une contrainte sur le type de communication entre le client et le SsT. En CVL, elle est spécifiée en OCL[2] dans le modèle de variabilité (le parallélogramme dans la figure 2a). Dans ce modèle, le type de communication du client est représenté par une variable *synchrone* dans l'implémentation du client. De plus, un autre type de point de variation, *SlotValueAssignment*, est utilisé pour le paramètre de composant. Il est lié à la spécification *synchrone* dans le modèle de variabilité et met à jour la valeur du composant correspondant dans le modèle de base. Par rapport au cas précédent, le client peut offrir deux interfaces de communication, une pour l'envoi et une autre pour la réception (en bas de la figure 2a).

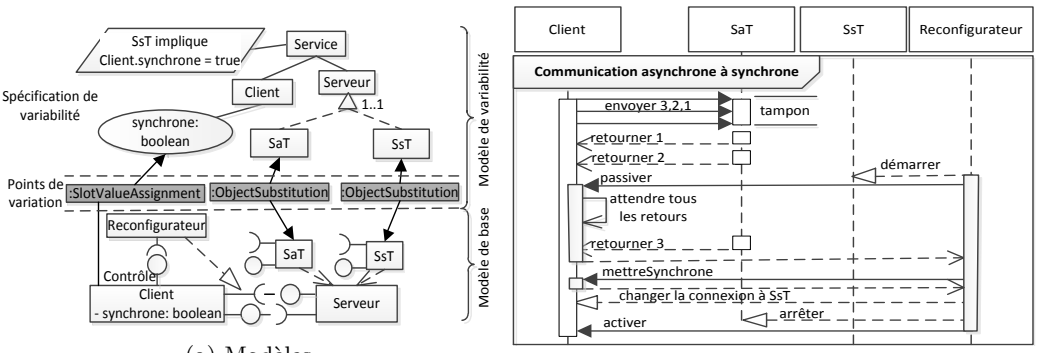
Les actions de reconfiguration pour ce cas sont représentées par un diagramme de séquence (figure 2b). Par rapport au premier cas, le *Reconfigurateur* doit contrôler le type de la communication du client, c'est-à-dire que le client change son comportement à l'exécution. Cette action de contrôle est générée grâce à la représentation de la variable *synchrone* et la contrainte dans le modèle de variabilité.

On constate dans ce cas que, en plus de savoir passer en mode passif, le client doit être capable de changer la nature de sa communication avec le serveur en passant d'asynchrone à synchrone. S'il ne sait pas le faire, on se trouve dans le cas suivant.

## 3.3 Le client est asynchrone et n'a pas été prévu pour évoluer

Dans ce cas, le type de communication entre le client et le SaT est asynchrone comme dans le cas précédent, mais ici il n'a pas été prévu de changer son type de communication. Dans ce

[2]http://www.omg.org/spec/OCL/

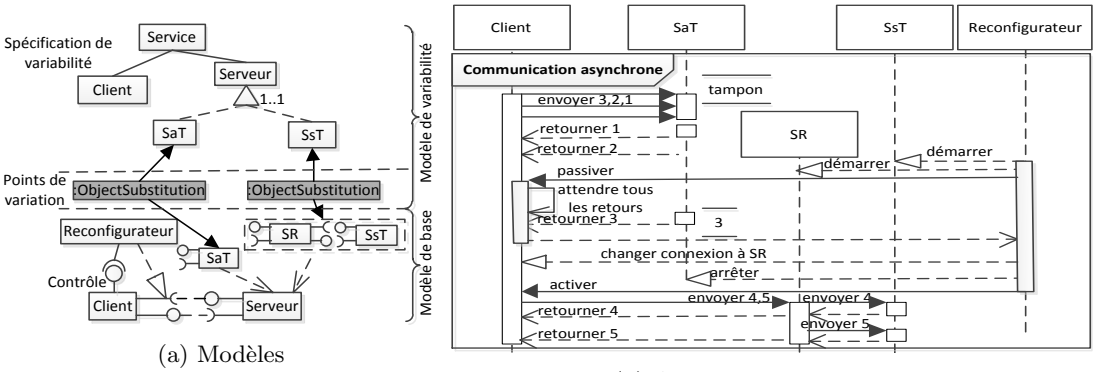



(a) Modèles

(b) Actions de reconfiguration

Figure 2: Type de communication asynchrone à synchrone

cas, comme le SsT n'a pas de tampon, l'application aura besoin d'un composant intermédiaire (serveur relais - SR) qui jouera le rôle de tampon pour les requêtes du client. Dans ce cas, au lieu de communiquer avec le serveur, le *Reconfigurateur* contrôle la connexion du client avec le SR (figure 3b).




(a) Modèles

(b) Actions de reconfiguration

Figure 3: Type de communication asynchrone

On constate, que par rapport au deuxième cas, il est nécessaire d'ajouter un composant.

## 3.4 Discussion

Notre étude de cas met en évidence la nécessité de gérer la multiplicité potentielle des requêtes et leur stockage pour éviter des pertes. Lorsque le serveur n'assure pas cette fonction, il faut qu'elle soit assurée ailleurs. Un client asynchrone ne pouvant pas assurer la gestion du flux, les solutions proposées consistent donc à modifier le client (cas 3.2) ou externaliser le stockage (cas 3.3). La première solution demande une anticipation peu naturelle lors de la conception. La seconde pourrait être automatisée et mise en œuvre par des mécanismes de reconfiguration qui seront le sujet de futurs travaux.

L'étude de cas présentée fait une hypothèse simplificatrice forte avec un unique client. Elle permet d'illustrer les besoins minimaux de reconfiguration : le client doit savoir se mettre dans un état passif. Si plusieurs clients étaient autorisés, le cas du client synchrone deviendrait aussi

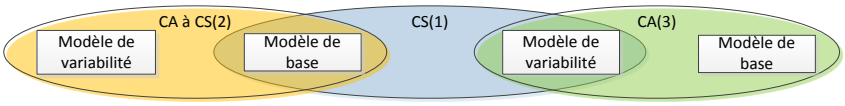



Figure 4: Une comparaison entre les trois cas

complexe que le cas du client asynchrone et nécessiterait des solutions similaires au cas 3.2 - adaptation interne du client - ou du cas 3.3 - introduction d'un tampon externe.

L'étude de cas montre également la nécessité d'utiliser à la fois un modèle de variabilité et de base (Figure 4). Un modèle de variabilité qui ne permet pas d'expliciter et relier ces deux dimensions semble, mais des études plus poussés sont nécessaires, ne pas pouvoir répondre au problème que nous posons. Le modèle CVL répond à ce besoin.

# 4 Conclusion

Cet article présente le problème de la reconfiguration en fonction de la nature des communications entre composants. Ce problème est mis en évidence par un cas de reconfiguration avec deux types de client. Nous montrons que dans certains cas, la reconfiguration n'est possible qu'avec des informations sur la nature de la communication entre le client et le serveur. En particulier, une information sur la dynamique des requêtes permettrait d'automatiser l'insertion de composants d'adaptation spécifiques (des tampons dans notre exemple).

Une reconfiguration ne peut être simplement un ensemble de connexion/déconnexion de composants sans hypothèse forte (souvent implicites) sur la nature des communications voire du nombre d'instances des composants. En relâchant les hypothèses, comme dans cet article, les actions de reconfiguration ne sont pas les mêmes si le composant est ou non conscient de la reconfiguration. Nous avons examiné des clients qui sont toujours conscients de la reconfiguration puis qu'ils doivent savoir être mis à l'état passif. Si les clients ne sont pas conscients, il pourrait être nécessaire de changer l'architecture. Cet aspect sera étudié dans des travaux futurs.

# Références

[1] Nelly Bencomo, Gordon S. Blair, Carlos A. Flores-Cortés, and Peter Sawyer. Reflective component-based technologies to support dynamic variability. In *Second International Workshop on Variability Modelling of Software-Intensive Systems 2008*, pages 141–150, 2008.

[2] Rafael Capilla, Jan Bosch, and Kyo Chul Kang, editors. *Systems and Software Variability Management, Concepts, Tools and Experiences*. Springer, 2013.

[3] Carlos Cetina, Pau Giner, Joan Fons, and Vicente Pelechano. Autonomic computing through reuse of variability models at run-time: The case of smart homes. In *XV Jornadas de Ingeniería del Software y Bases de Datos (JISBD 2010), Spain*, pages 99–99, 2010.

[4] CVL Submission Team. Common variability language (CVL), OMG revised submission, 2012.

[5] Kyo Kang, Sholom Cohen, James Hess, William Novak, and A. Peterson. Feature-oriented domain analysis (FODA) feasibility study. Technical Report CMU/SEI-90-TR-021, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 1990.

[6] Jeff Kramer and Jeff Magee. The evolving philosophers problem: Dynamic change management. *IEEE Trans. Software Eng.*, 16(11):1293–1306, 1990.

[7] Gustavo G. Pascual, Mónica Pinto, and Lidia Fuentes. Self-adaptation of mobile systems driven by the common variability language. *Future Generation Computer Systems*, 2014.