

---

# Composition comportementale d'aspects techniques

**Mikaël Beauvois, Thierry Coupaye**

*France Télécom R&D*

*Département Architecture des Systèmes Répartis*

*28 Chemin du Vieux Chêne, BP 98, 38243 Meylan*

*{mikael.beauvois, thierry.coupaye}@rd.francetelecom.com*

---

*RÉSUMÉ. Les infrastructures logicielles devenant de plus en plus complexes, leur conception doit se baser sur de nouveaux types de programmation offrant des sémantiques de composition plus évoluées. Différents paradigmes de programmation sont apparus successivement : la programmation modulaire, la programmation objet et récemment la programmation par composants et la programmation par aspects. La programmation par aspects offre la possibilité d'ajouter à l'application un certain nombre de propriétés non fonctionnelles, implantées sous forme d'aspect. La programmation par composants a pour objectif de généraliser l'application des principes de conception pour construire et faire évoluer les logiciels. Dans cet article, nous allons utiliser la programmation par composants avec une sémantique de composition qui se base sur les comportements des composants, avec comme objectif d'apporter une nouvelle approche pour la composition de services techniques.*

*ABSTRACT. Software systems are more complex, so their design has to use new programming styles that offer more sophisticated composition concepts. There are several programming paradigms such as modular programming, object oriented programming, and recently component oriented programming and aspect oriented programming. Aspect oriented programming allows us to add non functional properties (or aspects) to the application. Component oriented programming generalizes new design concepts to build software. In this article, we apply component oriented programming with component behaviour extension for trying to solve technical services composition.*

*MOTS-CLÉS: composant, composition, comportement, services techniques*

*KEYWORDS: component, composition, behaviour, technical services*

---

## 1. Introduction

La problématique qui nous intéresse dans cet article est la composition de services techniques. Un service technique, tel que la concurrence, la duplication, la persistance, la sécurité, est une unité d'exécution, composée de composants, et potentiellement indépendante d'autres services.

L'orthogonalité (ou la non orthogonalité) des services se définit à partir de leurs séquences d'exécution. Une séquence d'exécution d'un service se traduit par une suite de traitements élémentaires ou opérations (des invocations de méthode dans le monde Java par exemple). La composition des services génère un ensemble d'entrelacements des séquences d'exécution possibles de ces services. Deux opérations appartenant chacune à deux services peuvent être conflictuelles. Deux services sont dits orthogonaux si, quelque soit l'entrelacement des séquences d'exécution de ces services, l'ordre d'exécution des opérations conflictuelles est le même. La non orthogonalité est caractérisée par l'existence d'au moins un entrelacement tel que l'ordre d'exécution des opérations conflictuelles est différent de celui des autres entrelacements. Si l'on veut composer deux services non orthogonaux, il faut déterminer un ensemble d'entrelacements des séquences d'exécution possibles des différents services qui ont le même ordre d'exécution des opérations conflictuelles. Cet ensemble sera défini à partir de propriétés liées à l'ordre d'exécution des différentes séquences. L'approche retenue sera de déterminer dynamiquement un entrelacement correct des séquences d'exécution des services qui respecte l'ordre d'exécution des opérations conflictuelles.

Nous allons voir comment composer des services, et plus particulièrement des services non orthogonaux, en s'appuyant sur une approche par composants, en détaillant le comportement de ces derniers. Dans une approche orientée composants, les services sont modélisés sous forme de composants. Un service peut ainsi être représenté par plusieurs composants (le service implantant la persistance peut, par exemple, être constitué d'un composant gestionnaire de stockage et d'un composant gestionnaire de cache). Les séquences d'exécution possibles de chaque service sont modélisées par le comportement associé à chaque composant. Ce comportement s'appuie sur le modèle présenté dans la partie 2. La composition des services (orthogonaux ou non) correspond à la composition des comportements des composants représentant ces derniers. Cette composition utilisera des propriétés qui déterminent un ordre d'exécution. La partie 3 détaillera ces propriétés et leur insertion dans le modèle de composants réactifs.

## 2. Comportements et composants réactifs

Un comportement est un ensemble d'opérations avec un ensemble de contraintes qui déterminent quand elles peuvent intervenir [ISO 95]. La modélisation des comportements des composants permet d'appliquer des contraintes sur ces derniers afin d'enrichir leur sémantique et ainsi de contrôler plus finement leur composition. L'objectif est de définir un modèle de comportement minimal qui peut être étendu pour

mettre en œuvre d'autres techniques de composition liées au comportement. La modélisation du comportement d'un composant sera utilisée par la suite pour appliquer des contraintes de composition qui sont détaillées dans la partie 3.

Le modèle de comportement choisi se base sur les automates à états finis. Les automates à états finis représentent un modèle minimal pour décrire le comportement de n'importe quel composant. Un automate est défini par des *états* et des *transitions* entre ces états. Une transition est définie par la présence d'un *signal* reçu à partir d'une interface serveur [BRU 02] de type réactif, une *garde* se basant sur une expression booléenne, et des *actions* qui correspondent à l'envoi d'un signal sur une interface cliente [BRU 02] de type réactif ou l'exécution de l'implantation du composant.

La réception d'un signal sur une interface d'un composant peut donc générer une séquence d'actions, qui, à leur tour, peuvent générer en cascade une séquence d'actions sur d'autres composants. L'ensemble des séquences d'exécution possibles d'un composant réactif est modélisé par ces automates.

Le modèle de composants réactifs est un modèle étendu du modèle de base de composants Fractal [BRU 02] (basé sur ODP [ISO 95]), qui décrit la composition structurelle de composants. Un composant réactif est un composant dont le comportement a été modélisé à partir du modèle de comportement détaillé précédemment. Le comportement est défini par le concepteur du composant. Un composant réactif peut interagir avec d'autres composants réactifs par le biais de points d'accès : les interfaces de type réactif. Les composants communiquent entre eux en émettant des signaux sur ces interfaces. Un signal peut par exemple correspondre à une demande d'invocation d'une opération sans retour. Un signal reçu sur une interface fera réagir le composant réactif. Une interface de type réactif définit un ensemble de signaux. Un composant réactif peut interagir avec un composant non réactif par le biais d'interfaces de type non réactif au travers d'opérations (comme défini dans Fractal). Les composants peuvent être connectés entre eux en créant une liaison entre leurs interfaces de type réactif.

### 3. Composition de comportements de composants

Différents types de composition [COU 01] peuvent être définis : la composition structurelle qui permet de composer les composants en termes de liaisons et de relations de contenance, la composition comportementale qui se base sur la composition structurelle et se focalise sur la composition des comportements des composants, la composition contractuelle qui associe des contrats aux composants, contrats exprimés sous forme d'assertions, de clauses logiques, etc.

Les contraintes qui peuvent s'appliquer sur le comportement des composants sont par exemple des contraintes sur la séquentialité, la concurrence, les contraintes de temps réel, etc [ISO 95]. La composition de services non orthogonaux doit s'appuyer sur l'expression des contraintes liées à l'entrelacement des séquences d'exécution, et donc sur la synchronisation des comportements des composants. Ces contraintes sont dites des contraintes d'ordonnement.

La composition de services non orthogonaux s'apparente à la composition des composants implantant ces services, composants qui n'ont aucune liaison entre eux (ils sont structurellement indépendants), mais dont chaque action d'un automate d'un composant doit être ordonnée par rapport aux actions des autres composants car leurs séquences d'exécution sont interdépendantes. Cette composition est une composition parallèle d'automates (représentant le comportement des composants) non communicants à la base (car les aspects ne sont structurellement pas dépendants entre eux), mais qui vont devenir communicants par l'introduction des contraintes d'ordonnement, qui définissent un ensemble d'entrelacements possibles. Cette approche est la définition de l'utilisation de modèles synchrones réactifs. Le modèle synchrone permet de spécifier des processus concurrents qui s'échangent des informations et qui effectuent des traitements en un temps nul au niveau conceptuel, avec des langages tels que Esterel [BER 92] ou Lustre [CAS 87]. Le modèle réactif introduit, en plus de la concurrence, la notion de réaction (un processus réagit à une activation du monde extérieur) (Junior [HAZ 99]). Ces outils seront utilisés dans l'implantation de notre modèle.

L'introduction de contraintes d'ordonnement définit une relation d'ordre temporel entre les séquences d'exécution des différents composants. Le temps considéré est un temps logique. Il existe deux types d'action dans les transitions des descriptions des comportements : des actions séquentielles et des actions parallèles. Lors de la composition des comportements des composants, les comportements induits par les actions séquentielles sont ordonnés par le fait que la séquence est une relation d'ordre temporel ; par contre les comportements induits par les actions parallèles ne sont pas ordonnés entre eux, et les contraintes d'ordonnement peuvent s'y appliquer.

### **3.1. Contraintes d'ordonnement**

Les contraintes d'ordonnement peuvent être appliquées entre les comportements des composants qui s'exécutent en parallèle : la composition parallèle, dans un contexte de temps logique, des comportements des composants signifie que le modèle d'exécution a la possibilité de choisir l'ordre d'exécution des actions des composants dans un contexte de temps physique. Ces contraintes s'expriment à partir des éléments du modèle de comportement des composants [ISO 95]. Le choix d'un autre modèle de comportement implique la définition de nouveaux types de contraintes d'ordonnement. L'application d'une contrainte entre des comportements de composants insère un certain nombre de points de synchronisation dans ces derniers, qui jouent le rôle de "barrières" à l'exécution. La contrainte gère ces points de synchronisation qui autorisent ou non la suite de l'exécution des comportements. Elle joue le rôle d'un ordonnanceur, c'est-à-dire qu'elle détermine quelles actions sont autorisées à s'exécuter à un instant précis.

Une contrainte d'ordonnement symbolise une relation d'ordre temporel (relation de précédence) entre deux séquences d'exécution représentées par les comportements des deux composants considérés. Elle s'exprime à partir des débuts et fins

d'exécution des actions (cf. partie 2) liées à la réception d'un signal sur une interface et les changements d'état des automates. Il existe différents types de contrainte : (1) entre deux actions ( $a1 < a2$ ) où le début de l'exécution de l'action  $a2$  ne pourra commencer qu'à partir de la fin de l'exécution de l'action  $a1$ , (2) entre une action  $a1$  et un état  $st2$  où soit l'action  $a1$  ne pourra s'exécuter qu'à partir du moment où l'état  $st2$  aura été atteint ( $st2 < a1$ ), soit les actions menant directement à cet état ne pourront s'exécuter qu'à partir du moment où l'action  $a1$  a été exécutée ( $a1 < st2$ ), (3) entre deux états.

La validité des éléments de la contrainte d'ordonnement peut être bornée dans le temps, cette limite étant donnée par la fin d'une action associée à un automate ou le changement d'état d'un automate.

Nous envisageons de représenter les contraintes d'ordonnement elles-mêmes sous la forme d'un automate. Selon cette définition, une contrainte d'ordonnement est un composant réactif : il a un comportement propre représenté par un automate qui décrit une relation d'ordre, et des interfaces de type réactif qui reçoivent des signaux provenant des automates des composants à synchroniser et qui émettent des signaux à destination des points de synchronisation introduits dans les automates. Ces interfaces sont générées lors de l'application de la contrainte d'ordonnement. Il est également envisagé d'introduire les opérateurs de conjonction, disjonction et négation entre contraintes.

### 3.2. Composition de composants réactifs

La composition de composants réactifs sans contraintes d'ordonnement correspond à une composition parallèle des automates qui décrivent le comportement des composants. Ces automates communiquent par émission / réception de signaux sur leurs interfaces de type réactif qui ont été liées lors de l'assemblage des composants. Le résultat est comparable à une composition structurelle.

La composition des composants réactifs avec contraintes d'ordonnement est réalisée en composant en parallèle les automates des comportements définis pour chaque composant réactif et les automates des comportements des contraintes d'ordonnement. Les automates des composants communiquent entre eux via leurs interfaces de type réactif liées lors de l'assemblage (composition structurelle), et communiquent avec les automates des contraintes via les interfaces générées lors de l'activation des contraintes.

## 4. Composants et aspects

La programmation par aspects (Aspect Oriented Programming [KIC 97]) permet de spécifier des aspects qui définissent des propriétés non fonctionnelles. Un aspect, tout comme un service technique dans le cadre de cet article, se focalise sur une propriété non fonctionnelle. Ces deux problématiques sont selon nous très proches.

[DOU 02] définit la non orthogonalité entre aspects comme l'activation de ces aspects sur un même point de jonction. Pour résoudre ces conflits, le programmeur d'aspects doit alors définir un ordre d'exécution entre ces aspects pour ce point de jonction lors du tissage par exemple par un aspect de composition. La granularité de la composition est celle de l'aspect. Notre approche, qui se base sur les contraintes d'ordonnement, offre une granularité plus fine, celle des opérations conflictuelles qui constituent les aspects. Ce besoin de granularité plus fine est motivé par l'étude de services techniques fortement interdépendants tels que la persistance, la concurrence et le comportement transactionnel.

## 5. Conclusion

A partir d'un travail sur la notion d'orthogonalité des services techniques, une nouvelle approche basée sur les comportements de composants et des contraintes d'ordonnement a été présentée. Nous avons détaillé un modèle de composant réactif, avec un modèle de comportement associé. Le résultat de la composition de composants réactifs est la composition des comportements de ces composants avec les comportements des contraintes d'ordonnement qui déterminent des relations d'ordre d'exécution entre les composants. Ces modèles sont regroupés dans un framework de composition comportementale qui est un incrément du framework de composition Fractal [BRU 02]. Cet incrément est en cours d'implantation et se base sur la génération de code Esterel [BER 92] et Junior [BOU 00] pour exprimer et composer des composants du point de vue de leur comportement.

## 6. Bibliographie

- [BER 92] BERRY G., GONTHIER G., « The Esterel Synchronous Programming Language : Design, Semantics, Implementation », *Science of Computer Programming*, vol. 19, n° 2, 1992, p. 87-152.
- [BOU 00] BOUSSINOT F., HAZARD L., SUSINI J.-F., « La programmation en Junior », rapport, 2000, INRIA.
- [BRU 02] BRUNETON E., COUPAYE T., STEFANI J.-B., « Recursive and Dynamic Software Composition Sharing », *Seventh International Workshop on Component-oriented Programming at ECOOP 2002, Malaga, Spain*, 2002.
- [CAS 87] CASPI P., PILAUD D., HALBWACHS N., PLAICE J. A., « LUSTRE : A Declarative Language for Programming Synchronous Systems », *Conference Record of the Fourteenth Annual ACM Symposium on Principles of Programming Languages*, Munich, West Germany, janvier 21–23, 1987, ACM SIGACT-SIGPLAN, ACM Press, p. 178–188.
- [COU 01] COUPAYE T., LENGLET R., BEAUVOIS M., BRUNETON E., DÉCHAMBOUX P., « Composants et composition dans l'architecture des systèmes répartis », *Journées Composants 2001 Besançon*, , 2001.
- [DOU 02] DOUENCE R., FRADET P., SÜDHOLT M., « Detection and resolution of aspect interactions », rapport, 2002, INRIA.

- [HAZ 99] HAZARD L., SUSINI J.-F., BOUSSINOT F., « The Junior Reactive Kernel », rapport, 1999, INRIA.
- [ISO 95] ISO, Reference Model of Open Distributed Processing. International Standard ISO/IEC 10746, ITU-T Recommendations X.901-904, 1995, 1995.
- [KIC 97] KICZALES G., LAMPING J., MENHDHEKAR A., MAEDA C., LOPES C., LOINGTIER J.-M., IRWIN J., « Aspect-Oriented Programming », AKŞIT M., MATSUOKA S., Eds., *Proceedings of the 11th European Conference on Object-Oriented Programming (ECOOP'97)*, Jyväskylä, Finland, vol. 1241, Springer-Verlag, 1997, p. 220–242.