Les aspects et les composants logiciels

Etude de cas avec le modèle de composant Fractal

Houssam Fakih[*,**] — **Noury Bouraqadi**[*]

[*] *École des Mines de Douai*
941 rue Charles Bourseul, B.P. 838
59508 Douai Cedex, France
{fakih,bouraqadi}@ensm-douai.fr

[**] *Université des Sciences et Technologies de Lille*
Laboratoire LIFL (CNRS UMR 8022), Équipe GOAL
59655 Villeneuve d'Ascq Cedex, France

RÉSUMÉ. *La programmation par composants logiciels promet la réutilisation, mais elle est sujette aux problèmes de dispersion et de mélange de code représentant des propriétés transversales. L'application de la programmation par aspects (AOP) sur les composants logiciels permet de faire face à ces problèmes. Nous présentons dans ce papier* FRACTAL-AOP, *une extension du modèle de composant Fractal qui supporte la programmation par aspects. Cette extension consiste à doter les composants de nouvelles interfaces de contrôle qui donnent accès aux points de jonction. Ces interfaces donnent notamment accès au flot d'exécution des composants. Les aspects sont définis à l'aide de composants génériques. Le tissage se limite alors à des opérations classiques sur les composants constituant l'aspect à savoir : configuration et assemblage. La configuration permet de paramétrer l'aspect avec les points de jonction où il doit intervenir. Quant à l'assemblage, il permet de lier les composants constituant l'aspect avec les autres composants de l'application.*

ABSTRACT. *Component-Based Software Development (CBSD) swears software reuse but it suffers from code scattering and tangling. Aspect Oriented Programming (AOP) deals with these problems. We present in this paper,* Fractal-AOP, *an add-on to the Fractal component model that combines AOP and CBSD into an overall model. Combining is achieved by applying AOP principles on the Fractal component model. Indeed, we define new control interfaces on functional components that expose join points. Aspects are defined using plain Fractal generic components. Weaving relies only on classical operations on components, namely: Configuration and assembly. The configuration allows the definition of pointcuts. The assembly connects the components defining aspects with the rest of application components.*

MOTS-CLÉS : *Programmation par aspects, Composants logiciels, Modèle de composant Fractal.*

KEYWORDS: *Aspect Oriented Programming, Software Components, Fractal Component Model.*

2[e] soumission à *JFDLPA'04*, le 17 septembre 2004.

1. Introduction

Historiquement, la programmation par aspects (AOP, *Aspect-Oriented Programming*) [KIC 97, ELR 01] a été introduite sur la base de la programmation par objets. En effet, le code de base, sur lequel les aspects viennent se tisser, est défini à l'aide de classes et d'objets. L'intégration de l'AOP avec les composants logiciels [SZY 98] revient à revoir les concepts liés au paradigme aspect dans un contexte où les applications sont constitués de composants. Ceci consiste à :

1) définir des points de jonction (points particuliers du flot d'exécution) pour les composants et la façon permettant de les identifier,

2) définir une structure d'aspect comprenant des traitements (*Advices*) et des règles de tissage de l'aspect avec les composants (*before*, *after* et *around*),

3) définir le mécanisme de tissage des aspects avec les composants. Au-delà de la simple intégration de l'application, ce mécanisme doit gérer la multiplicité des aspects liés à un même composant. En effet, plusieurs aspects peuvent être non-orthogonaux et porter sur un même point de jonction. Dans ce cas, il est nécessaire de gérer les interactions entre les aspects (*feature interaction problem*) [PUL 01]. Cette gestion consiste à trouver une éventuelle bonne séquence pour exécuter les traitements des différents aspects.

Un problème connexe est celui de la réutilisation des aspects. Il s'agit de définir des aspects génériques pour qu'ils puissent être utilisés dans des contextes différents [LIE 99]. Cette définition nécessite de découpler les traitements des aspects des points de jonction. Une autre facette de la réutilisation concerne l'étude de la possibilité de composer des aspects existants pour définir de nouveaux aspects.

Les réponses aux questions précédentes dépendent fortement des caractéristiques du modèle de composant adopté. Par exemple, une solution pour un modèle plat doit être revu pourqu'elle soit appliquable à un modèle hiérarchique. En effet, la notion de composite introduit de nouveaux points de jonction comme par exemple les opérations d'ajout et de suppression de sous-composants. Le mécanisme de tissage doit par ailleurs être révisé pour appliquer les aspects aux sous-composants d'un composite.

Nous nous intéressons dans ce papier à l'introduction de la programmation par aspects dans le modèle de composant hiérarchique FRACTAL [BRU 04b]. Notre approche consiste à partir de ce modèle de composant et de l'enrichir avec le concept d'aspect. Nous introduisons brièvement dans la section 2 le modèle de composant FRACTAL. Puis nous décrivons dans la section 3 notre extension FRACTAL-AOP qui étend FRACTAL pour supporter la programmation par aspects. Les travaux apparentés sont présentés dans la section 4. Finalement, le papier se termine avec une conclusion et quelques perspectives de notre travail.

2. Le modèle de composant FRACTAL

FRACTAL [BRU 04a] est un modèle de composant *hiérarchique* fortement *typé*. Les composants composites fournissent une vue uniforme de l'application à plusieurs niveaux d'abstraction. Toutefois, le modèle n'est pas strictement hiérarchique. FRACTAL donne la possibilité à un composant donné de faire partie du contenu de plusieurs composants englobants. On parle alors du *partage* de composants. FRACTAL fournit différentes capacités de réflexion pour permettre de contrôler les composants au déploiement et à l'exécution.

2.1. *Présentation générale*

Un composant FRACTAL est une boîte noire qui peut être manipulée à travers un ensemble d'*interfaces* [1]. La connexion entre une interface cliente et une interface serveur est nommée *liaison* ou *binding*. FRACTAL étant fortement typé, l'interface serveur doit être un *sous-type* de l'interface cliente. La vérification de sous-typage se fait au moment de l'établissement de la liaison entre les deux interfaces.

Les communications entre les composants transitent via leurs interfaces. Ainsi, un composant qui requiert un service envoie un message via une de ses interfaces *clientes* à une interface *serveur* d'un composant fournissant ce service. Une interface serveur donne accès à un sous-ensemble des opérations fournies par le composant. Tandis qu'une interface cliente définit un sous-ensemble des opérations requises par le composant.

La structure d'un composant FRACTAL est divisée en deux parties :

– **Un contenu** : comprend un ensemble fini d'opérations, d'attributs et de sous-composants.

– **Une membrane** : contient toutes les interfaces du composant. Elle contrôle le contenu du composant en interceptant les invocations d'opérations reçues et envoyées via les interfaces. Nous utilisons cette caractéristique pour appliquer l'AOP aux composants (voir section 3).

Une membrane FRACTAL comprend deux sortes d'interfaces : les interfaces *fonctionnelles* et les interfaces de *contrôle*. Les interfaces fonctionnelles sont "reliées" au métier du composant. Tandis que les interfaces de *contrôle* permettent l'introspection et la reconfiguration du composant. La spécification FRACTAL [BRU 04b] définit des interfaces de contrôle pour gérer les attributs, les connexions entre les composants, les sous-composants, les composites, l'état de cycle de vie...

La figure 1 donne une représentation graphique d'un composant Fractal. Par convention, toutes les interfaces de rôle serveur sont représentées sur la partie gauche du composant. Les interfaces de rôle client sont représentées sur la partie droite. Les in-

1. La notion d'interface dans FRACTAL correspond à la notion de *port* qu'on trouve dans d'autres modèles de composant [SZY 98].

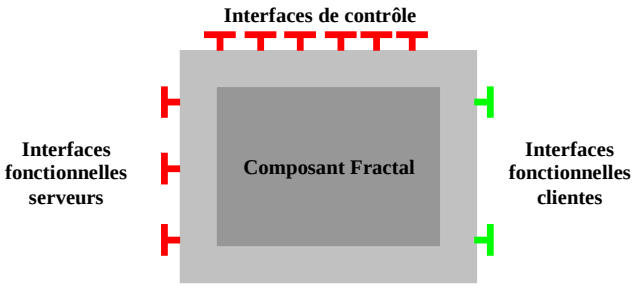

Figure 1. *Représentation graphique d'un composant Fractal et de ses interfaces*

terfaces de contrôle de rôle serveur sont représentées dans la partie supérieure. Notons que la spécification Fractal n'évoque pas des interfaces de contrôle ayant le rôle client.

2.2. *Les liaisons dans* FRACTAL

La spécification Fractal définit deux types de liaisons : les liaisons *primitives* et les liaisons *composites*. Une liaison primitive est une simple référence entre une interface cliente et une interface serveur de deux composants. Ces deux composants doivent être situés dans le même espace d'adressage.

Une liaison composite désigne un ou plusieurs composants dont la seule fonction est la communication. Elle englobe également les éventuelles liaisons primitives qui lient ces composants de communication. Un exemple d'une telle liaison est la communication distante. La liaison regroupe dans ce cas les composants *stub* et *squelette* sur les différentes machines.

2.3. *Un exemple d'application : Gestion des agendas*

Nous illustrons, dans la figure 2, une application de gestion d'agendas construite à partir d'un ensemble de composants Fractal. Chaque composant agenda fournit une interface serveur *Rdv* permettant d'ajouter ou de supprimer un rendez-vous. Chaque composant Organisateur de Rendez-vous fournit une interface serveur *ORdv* permettant d'organiser un rendez-vous entre plusieurs propriétaires d'agendas. Il requiert une interface *Rdv* cliente de *cardinalité collection*, c'est-à-dire pouvant être liée à plusieurs interfaces serveurs. Nous utiliserons cet exemple tout au long de ce papier pour illustrer notre approche.

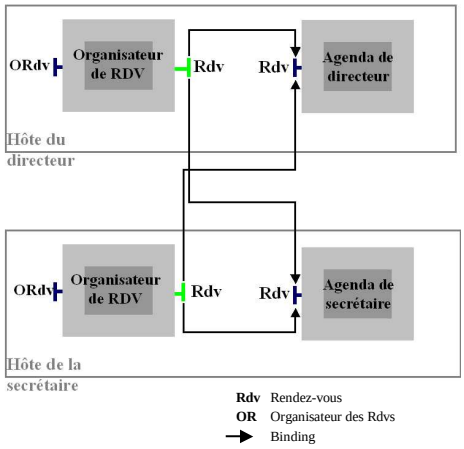

Figure 2. *Un exemple d'application de gestion d'agendas avec* FRACTAL

3. FRACTAL-AOP une extension de Fractal supportant l'AOP

FRACTAL-AOP étend le modèle de composant FRACTAL avec le support de la programmation par aspects. Pour ce faire, les composants sont dotés de deux nouvelles interfaces de contrôle désignées par CEC et SIC dans la figure 3. Par ailleurs, FRACTAL-AOP propose de définir les aspects à l'aide de composants génériques. Ainsi, non seulement il permet d'isoler les définitions des aspects, mais en plus ces aspects sont réutilisables dans différents contextes.

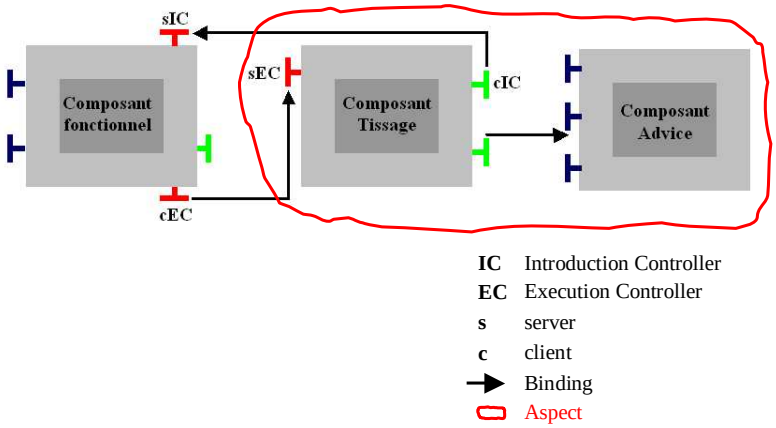

Figure 3. *Un composant tissé à un aspect dans* FRACTAL-AOP

3.1. *Les composants* FRACTAL-AOP

Comme nous l'avons dit plus haut, FRACTAL-AOP enrichit la structure des composants en introduisant deux nouvelles interfaces de contrôle. La première interface permet le contrôle du *flot d'exécution* . Alors que la seconde permet les *introductions*.

Grâce à ces deux interfaces, un composant peut être tissé à n'importe quel aspect sans en "avoir connaissance". C'est la propriété d'*obliviousness* [FIL 00]. En effet, l'aspect supervise et contrôle l'exécution des composants auxquels il est tissé. Ainsi, il a la possibilité d'exécuter ses traitements à l'insu des composants.

3.1.1. *L'interface de contrôle du flot d'exécution*

Cette interface, désignée par CEC dans la figure 3 pour *client Execution Controller*, est une interface de contrôle cliente. Elle permet le contrôle de l'exécution du composant en donnant accès aux points de jonction occurrents dans le flot d'exécution du composant. En effet, à chaque point de jonction, la membrane du composant va émettre une invocation d'opération via cette interface. Ainsi les composants de l'aspect connectés à cette interface peuvent alors intervenir.

Notons que cette notion d'interface de contrôle ayant le rôle client est une extension du Fractal dont la spécification [BRU 04b] n'évoque que des interfaces de contrôle ayant le rôle serveur[2].

La figure 4 illustre l'API EC de contrôle de flot d'exécution.

```
package org.objectweb.fractal.api.aop;
interface ExecutionController {
    receiveMessage(string aMessage, any[] args,
     Interface itfReceiver, Component receiver,
     Interface itfSender, Component sender);
    sendMessage(string aMessage, any[] args,
     Interface itfSender, Component sender,
     Interface itfReceiver, Component receiver);
    getAttribute(string attributeName, any value, Component cmp);
    setAttribute(string attributeName, any newValue, Component cmp);
    connect(Interface itf1, Component cmp1, Interface itf2, Component cmp2);
    disconnect(Interface itf1, Component cmp1, Interface itf2, Component cmp2);
    addSubComponent(Component subComponent, Component composite);
    removeSubComponent(Component subComponent, Component composite);
    changeState(string newState, Component cmp);
}
```

Figure 4. API *d'interface de contrôle de flot d'exécution écrite en* IDL FRACTAL

3.1.2. *L'interface de contrôle d'introduction*

Nous reprenons ici le terme *Introduction* tel qu'il est défini dans ASPECTJ [KIC 01]. Autrement, il désigne une partie des changements transversaux statiques modifiant la

2. Page 11 de [BRU 04b] : "..A control interface is a server interface that corresponds to a "non functional aspect".."

structure du code applicatif. Cette deuxième interface, désignée par sIC dans la figure 3 pour *server Introduction Controller*, est une interface de contrôle serveur. Elle permet d'opérer des *introductions* au niveau de chaque composant. Ainsi, elle donne accès à des opérations permettant de modifier la structure du composant. Autrement dit, l'interface sIC permet de modifier le type d'un composant. Cette extension demeure compatible avec la spécification FRACTAL[3]. Néanmoins, toute modification dans la structure du composant implique de vérifier l'influence de cette restructuration sur les liaisons du composant déjà établies.

La figure 5 illustre l'API IC de contrôle des introductions.

```
package org.objectweb.fractal.api.aop;
interface IntroductionController {
    void setFcItfType(InterfaceType anItfType, ComponentType cmpType);
    void setFcType(ComponentType aCmpType);
    void addMessage(any aMessage, Interface interface, Component component, any impl);
    void removeMessage(any aMessage, Interface interface, Component component);
    void addInterface(Interface interface, Component component, any impl);
    void removeInterface(Interface interface, Component component);
}
```

Figure 5. API *d'interface de contrôle d'introductions écrit en* IDL FRACTAL

3.2. *Les points de jonction*

Grâce aux deux interfaces de contrôle cEC et sIC, les aspects de FRACTAL-AOP peuvent intervenir aux points de jonction suivants :

– écriture/lecture des attributs,

– connexion/déconnexion de deux composants,

– ajout/suppression d'un sous-composant à un composite,

– changement de l'état de cycle de vie d'un composant,

– ajout/suppression d'un attribut/opération/interface au composant,

– réception d'un message sur les interfaces de rôle serveur,

– émission d'un message sur les interfaces de rôle client.

Du point de vue de l'implantation, la structure des composants Fractal rend facile l'interception de ces points de jonctions. En effet, les différentes opérations correspondent à des communications à travers les interfaces fonctionnelles ou de contrôle.

3. Page 25 de [BRU 04b] : "Indeed the *ComponentType* and *InterfaceType* interfaces do not offer any operations to modify an existing type, and the other interfaces specifi ed in this document do not offer a operation to change the type of a component or of an interface. But a Fractal component may perfectly provide a setFcType operation, if needed, since the Fractal model is extensible."

3.3. *L'aspect dans* FRACTAL-AOP

Un aspect est défini dans FRACTAL-AOP à l'aide d'un ensemble de composants. Ceux-ci peuvent être de deux sortes : les composants *Advice* et les composants *Tissage*. Les composants Advice définissent uniquement le métier de l'aspect. Ils sont génériques et donc réutilisables. A l'opposé, les composants Tissage définissent les points de coupe où l'aspect doit intervenir. Ils sont spécifiques à une application particulière puisqu'ils servent à tisser l'aspect dans l'application en question.

3.3.1. *Le composant Advice*

Le composant Advice définit exclusivement et uniquement les traitements relatifs à un et un seul aspect. Ce composant Advice peut être défini à l'aide d'un composant primitif ou un composite. Ceci doit se faire d'une manière générique indépendamment de tout contexte. Par conséquent, les composants Advice sont *réutilisables* puisqu'ils représentent exclusivement le métier de l'aspect sans prendre compte du domaine dans lequel il va être appliqué.

3.3.2. *Le composant Tissage*

Il correspond à une liaison composite entre les composants fonctionnels et les composants Advice. Il gère les définitions de coupes et de règles de tissage. En effet, un composant Tissage est responsable de l'invocation des opérations des composants Advice. Il a également la charge de gérer la multiplicité des aspects (cas de plusieurs aspects portant sur le même point de jonction). Ce composant possède une interface fonctionnelle SEC complémentaire[4] à l'interface CEC. Il possède également une interface cliente pour chaque interface serveur des composants Advice qu'il va utiliser. Le composant Tissage reçoit sur son interface SEC tous les messages donnant accès aux points du flot d'exécution du composant auquel il est lié. Il analyse et filtre ces points de jonction. Si une coupe a lieu, il utilise ses interfaces requises pour exécuter les opérations adéquates des composants Advices auxquels il est connecté.

3.4. *Le cas des composants composites*

La figure 6 montre un exemple d'un composite et des relations entre ses interfaces CEC et SIC et celles de ses sous-composants. La prise en charge des propriétés transversales d'un composite se fait de la même manière qu'avec un composant Fractal primitif, à la seule différence que l'interface cEC définie sur le composite n'expose pas seulement les points de jonction du composite lui même mais aussi les points de jonction de ses sous-composants. Tout message donnant accès à un point de jonction contient un paramètre correspondant au sous-composant auquel se rattache le point de jonction. Nous pouvons alors identifier au niveau du composant Tissage les points

4. Deux interfaces sont dites complémentaires si elles ont le même type mais avec des rôles opposés, *i.e.*, l'une cliente et l'autre serveur.

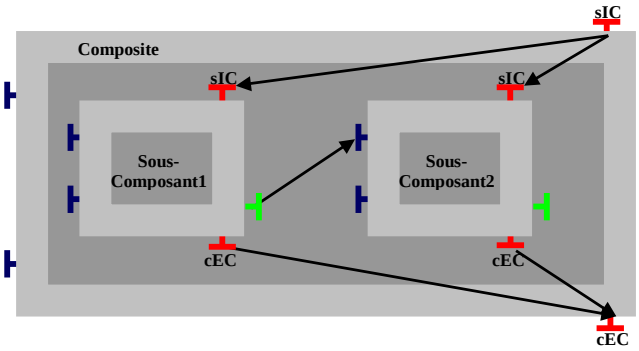

Figure 6. *L'application d'un aspect au composite*

de jonction et les sous-composants auxquels ils appartiennent. Il en va de même pour l'interface SIC qui permet de préciser le composant cible dont nous souhaitons changer la structure. Le modèle Fractal autorise le changement dynamique de la structure (ajout/suppression de sous-composants) et le partage de sous-composants. Pour le moment, nous ne gérons pas l'ajout d'un sous-composant à un composite si ce premier est déjà tissé à un aspect, *i.e.*, un composant Advice ne peut pas être lié à un sous-composant d'une manière directe mais à travers l'interface CEC du composite. Concernant le partage, nous laissons à l'utilisateur le choix de connecter les SIC et CEC du sous-composant au composite de son choix.

3.5. *Exemple d'un aspect dans* FRACTAL-AOP

Dans cette section nous donnons un exemple d'utilisation de Fractal-AOP pour la définition d'un aspect. Nous reprenons ici l'exemple d'agenda introduit dans la section 2. L'aspect développée est celui de l'authentification.

Afin de simplifier le discours nous nous limitons uniquement à vérifier que les requêtes reçues par l'agenda du directeur proviennent bien de l'organisateur de rendez-vous de la secrétaire. Nous utilisons une approche d'authentification dite *upfront login authentication approach* [LAD 03]. Elle consiste à demander, à toute entité essayant d'accéder au composant Agenda, de fournir un login et un mot de passe afin d'obtenir la permission d'utiliser l'interface Rdv.

La conception originale de notre application de gestion agenda ne prévoyait pas le support de l'authentification. Néanmoins, cet aspect peut être ajouté grâce aux interfaces de contrôle du flot d'exécution CEC comme le montre la figure 7. Le développement de notre aspect authentification et son intégration sont réalisés suivant les étapes suivantes :

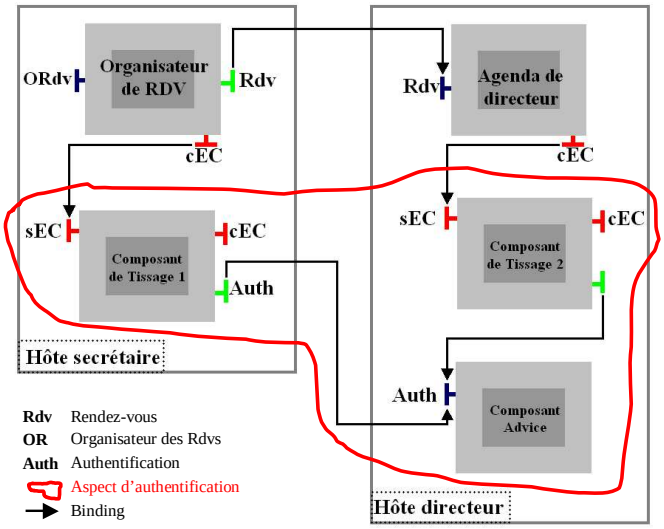

Figure 7. *Exemple d'un aspect d'authentifi cation avec* FRACTAL-AOP

étape 1 : Définition des composants Advice. Dans cet exemple, notre aspect comporte un unique composant Advice. Ce dernier fournit une interface *Auth* définissant deux opérations : *login* et *isAuthenticated*. La première opération permet d'authentifier des entités en se basant sur leur login et leur mot de passe. La deuxième vérifie si une entité a été déjà authentifié ou pas,

étape 2 : Définition des composants Tissage. Dans notre exemple, il existe deux composants de Tissage. Le premier est lié au composant Organisateur de Rendez-vous. Il capture tous les points de jonction correpondant à une émission de messages à partir de son interface client Rdv. Lors de l'émission du premier message par le composant Organisateur Rdv, il suspend ce dernier pour effectuer d'abord l'authentification (utilisation de l'opération *login* du composant Advice). Le second composant de tissage est lié au composant Agenda. Il capture tous les messages reçus par le composant Agenda. Avant d'exécuter un message, il vérifie si l'émetteur a été déjà authentifié (utilisation de l'opération *isAuthenticated* du composant Advice).

étape 3 : Tissage de l'aspect. Le tissage est rendu actif en assemblant les composants de notre aspect avec ceux de l'application. Cet assemblage revient à lier l'interface CEC de chaque composant métier avec l'interface SEC d'un composant Tissage. Ainsi, le composant Agenda est connecté au composant Tissage 1. De manière analogue le composant Organisateur de Rdvs est connecté au composant Tissage 2. Une fois cet assemblage terminé, notre aspect est opérationel. L'application est ainsi enrichie pour assurer l'authentification.

3.6. *La multiplicité des aspects*

La question de la multiplcité se pose quand l'exécution d'un composant doit être altérée par plusieurs aspects. Ces derniers peuvent éventuellement être en conflit s'ils interviennent à un même point de jonction. Dans un tel cas, le développeur doit intervenir afin de préciser la sémantique de l'intégration. Le plus souvent, cela revient à définir l'ordre d'exécution des traitements des différents aspects. FRACTAL-AOP adopte cette même démarche.

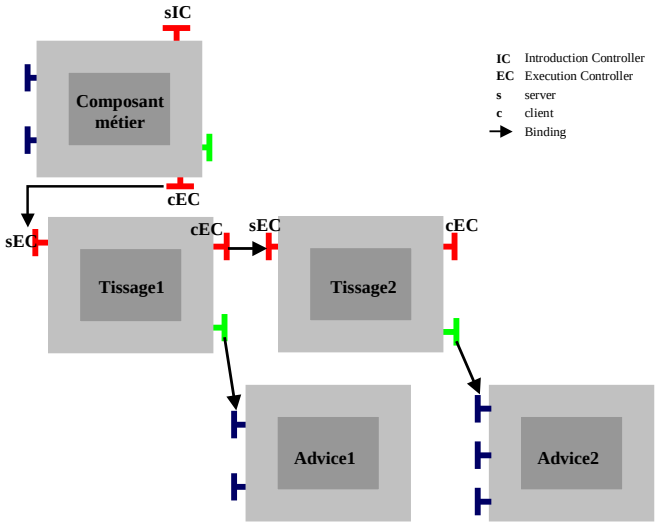

Figure 8. *Une solution pour gérer la multiplicité des aspects*

La figure 8 donne le détail de la solution de FRACTAL-AOP au problème de multiplicité. Les composants de tissage ne sont plus liés directement à l'interface CEC du composant métier mais ils forment une chaîne à la manière du patron de conception "chaîne de responsabilités" [GAM 95]. Afin de constituer cette chaîne, tout composant Tissage possède une interface CEC *optionnelle*[5]. A chaque fois que le composant Tissage reçoit un message sur son interface SEC (i.e. à chaque point de jonction), il déclenche les traitements de l'aspect auquel il appartient. Puis, il retransmet le message sur son interface CEC. Ainsi, le composant Tissage suivant dans la chaîne prend la main et fait ses traitements et ainsi de suite. Dans le cas des aspects non orthogonaux, les traitements des aspects de la figure 8 s'enchaînent de la manière suivante :

1) "before" de Advice 1 : exécution du pré-traitement correspondant à l'aspect 1.

2) "before" de Advice 2 : exécution du pré-traitement correspondant à l'aspect 2.

3) "proceed" : exécution du code correspondant au point de jonction.

4) "after" de Advice 2 : exécution du post-traitement correspondant à l'aspect 2.

5. il n'est pas nécessaire qu'elle soit liée pour que le composant fonctionne.

5) "after" de Advice 1 : exécution du post-traitement correspondant à l'aspect 1.

Il est important de souligner que FRACTAL-AOP permet d'implanter d'autres solutions pour gérer la multiplicité des aspects et résoudre les conflits. En effet, l'intégrateur a la liberté totale de manipuler les composants Tissage et Advice des différents aspects ainsi que leurs liaisons avec les composants applicatifs. A titre d'exemple, il peut insérer un composant de résolution de conflit entre le composant métier et les composants Tissage. Ce nouveau composant serait d'une part lié à l'interface CEC du composant métier et d'autre par lié aux interfaces SEC des composants de tissage. L'intégrateur peut ainsi définir sa propre politique de transmission des points de jonction aux composants Tissage.

4. Travaux Apparentés

4.1. *Travaux sur l'intégration des aspects et des composants*

Dans [FAK 04], nous avons identifié trois catégories suivant lesquelles les travaux concernant l'intégration des aspects et des composants peuvent être classés :

– La représentation des aspects sous forme de composants avec un code de base exprimé dans un autre paradigme (e.g. objet). Nous trouvons dans cette catégorie les travaux de Lieberherr et al. [LIE 99] et le modèle CAESAR [MEZ 03].

– L'application de l'AOP aux composants logiciels. Ici, le code de base est exprimé en termes de composants logiciels, mais ce n'est pas le cas des aspects. JASCO [SUV 03] et JBOSS-AOP [BUR 04] sont représentatifs de cette catégorie.

– l'unification des aspects et des composants logiciels. Il s'agit d'exprimer aussibien le code de base que les aspects à l'aide d'un modèle unique de composants logiciels. Les travaux sur FUSEJ [SUV 04], JIAZZI [MCD 03] et DAOP [PIN 02] font partie de cette catégorie.

Dans ce papier, nous nous sommes intéressés uniquement à l'applicaton de l'AOP aux composants. Les aspects ne sont pas représentés sous forme de composants, bien que ce soit le cas des éléments qui les constituent (composants Tissage et Advice). Nous avons volontairement choisi de ne pas franchir le pas et définir un aspect comme étant un composite. En effet, nous n'avons pas encore étudié toutes les conséquences d'une telle décision, comme par exemple la répartition des aspects entre différentes machines.

JASCO [SUV 03] applique l'AOP au modèle de composant JAVA BEAN de SUN. Il introduit alors deux nouveaux concepts : *aspect bean* et *connector*. Un aspect bean regroupe des coupes abstraites, des traitements d'aspects ainsi que des règles de tissage. Quant au connecteur, il sert à configurer un aspect en définissant des coupes concrètes. Il permet également de gérer la multiplicité des aspects.

JBOSS-AOP [BUR 04] est une autre approche qui introduit l'AOP dans la plateforme JBOSS, qui supporte les composants EJB [DEM 01]. Un aspect dans JBOSS-

AOP correspond à un ensemble d'advices défini dans une seule classe. La définition de tissage et des coupes se fait à l'aide de balises XML d'une manière séparée de la définition des aspects. Cependant, l'approche adoptée par JBOSS-AOP consiste à voir le code de base comme des objets. Le concept de composant n'est pas directement pris en compte.

4.2. *Extensions du modèle* FRACTAL

L'approche Pessemier et al. [PES 04] applique l'AOP aux composants FRACTAL. Les composants FRACTAL sont dotés dans cette approche d'une interface de contrôle d'interception qui permet de capturer seulement les invocations de messages dans les interfaces du composant. Cette interface d'interception est restrictive par rapport à l'interface cEC de FRACTAL-AOP qui donne accès à différents types de points de jonction (voir section 3.2). Les aspects sont définis dans des composants dits *Aspect Component*. Le tissage s'appuie sur un nouveau type de liaisons nommé *direct cross-cut binding*. A l'opposé, FRACTAL-AOP utilise exclusivement les types de liaisons "classiques" définis dans FRACTAL. Par ailleurs, la politique de gestion de multiplicité est implicite dans le modèle de Pessemier et al. car elle est codée dans la membrane des composants métier. Alors que FRACTAL-AOP permet aux développeurs de revoir cette politique en externalisant la gestion des conflits au niveau de composant Tissage.

David et al. [DAV 03] propose un MOP pour Fractal. Tous les messages reçus par un composant Fractal sont réifiés. Un sous-composant, contrôle les messages réifiés et exécute les éventuels pré- et post-traitements méta qu'il implante. Dans FRACTAL-AOP, les composants Tissage sont analogues à ces méta-composants. Mais, ils diffèrent notamment par le fait qu'ils donnent accès à différents points de jonction. Par ailleurs, FRACTAL-AOP sépare explicitement les définitions de points de coupe (composants Tissage) et de traitements des aspects (composants Advice).

Hérault et al. [HéR 03] définissent les services techniques (*i.e.* non-fonctionnels) sous la forme de composants FRACTAL traditionnels. La liaison entre un service technique et un composant fonctionnel se fait en insérant ce dernier dans un composite. La membrane du composite donne la main au service technique pour les pré- et post-traitements. L'interaction entre le composite et le service technique est cependant spécifique à l'implantation JULIA de FRACTAL en JAVA.

5. Conclusions et Perspectives

Ce papier présente FRACTAL-AOP une extension de FRACTAL supportant la programmation par aspects. Dans FRACTAL-AOP, les composants sont dotés de deux interfaces de contrôle supplémentaires : l'une donne accès aux points de jonction dans le flot d'exécution et l'autre permet de modifier la structure du composant. Ces deux interfaces sont génériques et indépendantes du tout contexte applicatif. La définition d'un aspect est séparée en deux parties. L'une générique comprenant des composants

Advice qui définissent la partie métier de l'aspect. L'autre partie est spécifique à une application donnée comprenant des composants Tissage. Ces derniers définissent les points où l'aspect doit intervenir et déclenchent l'exécution des opérations adéquates au niveau des composants Advice. Dans ce contexte, le tissage revient à assembler les composants Tissage aux composants applicatifs. Une implantation existe sur la base de FRACTALK[6], notre implantation du modèle FRACTAL en Smalltalk.

L'objectif final de notre travail est d'unifier les notions d'aspect et de composant. Dans un tel contexte les aspects seraient non-seulement constitués de composants, mais ils seraient eux même représentés sous forme de composants (probablement composites). Cette unification soulève un certain nombre de questions. L'une d'elle est la manière de représenter les aspects des applications réparties. En effet, dans une application répartie, un aspect peut nécessiter d'être tissé à des composants situés sur différentes machines. Comment alors représenter cet aspect ? Sous forme de composant réparti ? A l'aide de liaisons distantes (solution probablement trop coûteuse) ? En répliquant le composant qui représente l'aspect ?

Une autre question consiste aux limites à poser à l'unification. Si on représente les aspects sous forme de composants, ces derniers doivent-ils aussi être dotés d'interfaces de contrôle CEC et SIC ? Ceci revient à offrir la possibilité d'avoir des aspects sur des aspects. Il reste à voir si cette possibilité a un sens et des domaines d'application ?

Remerciements

Les auteurs remercient Laurence DUCHIEN pour ses commentaires et ses remarques autour de FRACTAL-AOP et la première version de ce papier. Ils remercient également Abdelaziz GACEMI d'avoir lu et commenté ce papier.

6. Bibliographie

[BRU 04a] BRUNETON E., COUPAYE T., LECLERCQ M., QUEMA V., STEFANI J.-B., « An Open Component Model and Its Support in Java », *Proceedings of the International Symposium on Component-based Software Engineering*, Edinburgh, Scotland, mai 2004.

[BRU 04b] BRUNETON E., COUPAYE T., STEFANI J.-B., « The Fractal Component model specifi cation », ObjectWeb Consortium, France Telecom and INRIA, février5 2004, http ://fractal.objectweb.org.

[BUR 04] BURKE B., CHAU A., FLEURY M., BROCK A., GODWIN A., GLIEBE H., « JBoss Aspect Oriented Programming », http ://www.jboss.org/, février 2004.

[DAV 03] DAVID P.-C., LEDOUX T., « Towards a Framework for Self-Adaptive Component-Based Applications », *Proceedings of DAIS'03*, Lecture Notes in Computer Science, Paris, novembre 2003, Federated Conferences, Springer-Verlag.

6. `http ://csl.ensm-douai.fr/FracTalk`

[DEM 01] DEMICHIEL L., ÜMIT YALÇINALP, KRISHNAN S., « Enterprise JavaBeans Specification, Version 2.0 », Sun Microsystems Inc., version 2.0 édition, August 2001.

[ELR 01] ELRAD T., FILMAN R. E., BADER A., « Aspect-Oriented Programming », *Comm. ACM*, vol. 44, n° 10, 2001, p. 29–32.

[FAK 04] FAKIH H., BOURAQADI N., DUCHIEN L., « Towards Integrating Aspects and Components », COADY Y., LORENZ D., Eds., *Proceedings of the Third AOSD Workshop on Aspects, Components, and Patterns for Infrastructure Software*, Lancaster University, UK, mars 2004.

[FIL 00] FILMAN R. E., FRIEDMAN D. P., « Aspect-Oriented Programming is Quantification and Obliviousness », *Workshop on Advanced Separation of Concerns (OOPSLA 2000)*, octobre 2000.

[GAM 95] GAMMA E., HELM R., JOHNSON R., VLISSIDES J., *Design patterns : elements of reusable object-oriented software*, Addison-Wesley Longman Publishing Co., Inc., 1995.

[HéR 03] HÉRAULT C., LECOMTE S., « Adaptabilité des Services Techniques dans un Modèle à Composants », *3ème Conférence Française sur les Systèmes d'Exploitation (CFSE)*, La Colle sur Loup, France, octobre 2003.

[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 European Conference on Object-Oriented Programming*, vol. 1241, p. 220–242, Springer-Verlag, Berlin, Heidelberg, and New York, 1997.

[KIC 01] KICZALES G., HILSDALE E., HUGUNIN J., KERSTEN M., PALM J., GRISWOLD W. G., « An Overview of AspectJ », *Lecture Notes in Computer Science*, vol. 2072, 2001, p. 327–355.

[LAD 03] LADDAD R., *AspectJ in Action*, Manning Publications Co., Grennwich, Conn., 2003.

[LIE 99] LIEBERHERR K., LORENZ D. H., MEZINI M., « Programming with Aspectual Components », rapport n° NU-CCS-99-01, mars 1999, College of Computer Science, Northeastern University, Boston, MA 02115.

[MCD 03] MCDIRMID S., HSIEH W. C., « Aspect-oriented programming with Jiazzi », *Proceedings of the 2nd international conference on Aspect-oriented software development*, ACM Press, 2003, p. 70–79.

[MEZ 03] MEZINI M., OSTERMANN K., « Conquering aspects with Caesar », *Proceedings of the 2nd international conference on Aspect-oriented software development*, ACM Press, 2003, p. 90–99.

[PES 04] PESSEMIER N., SEINTURIER L., DUCHIEN L., BARAIS O., « Partage de composants Fractal pour l'AOP », *First French Workshop on Aspect-Oriented Software Development (JFDLPA 2004)*, Paris, France, septembre14 2004, In French.

[PIN 02] PINTO M., FUENTES L., FAYAD M., TROYA J. M., « Separation of Coordination in a Dynamic Aspect Oriented Framework », KICZALES G., Ed., *Proc. 1st Int' Conf. on Aspect-Oriented Software Development (AOSD-2002)*, ACM Press, avril 2002, p. 134-140.

[PUL 01] PULVERMÜLLER E., SPECK A., COPLIEN J., D'HONDT M., DEMEUTER W., Eds., *Proceedings of the Workshop on Feature Interaction in Composed Systems ; In Association with the 15th European Conference on Object-Oriented Programming (ECOOP) 2001*, Universitaet Karlsruhe, Jun 2001.

[SUV 03] SUVÉE D., VANDERPERREN W., JONCKERS V., « JAsCo : an aspect-oriented approach tailored for component based software development », *Proceedings of the 2nd international conference on Aspect-oriented software development*, ACM Press, 2003, p. 21–29.

[SUV 04] SUVÉE D., VANDERPERREN W., WAGELAAR D., JONCKERS V., « There are no aspects », Electronic Notes in Theoretical Computer Science, Elsevier Science, avril 2004.

[SZY 98] SZYPERSKI C., *Component Software : Beyond Object-Oriented Programming*, ACM Press and Addison-Wesley, New York, NY, 1998.