18 Une extension de Fractal pour l'AOP

Nicolas Pessemier[1] — Lionel Seinturier[1,2] — Laurence Duchien[1] — Olivier Barais[1]

[1] *USTL-LIFL, INRIA Futurs, Equipe GOAL/Jacquard*
Bâtiment M3, 59655 Villeneuve d'Ascq Cedex, France

[2] *Université Paris 6, LIP6, Equipe SRC*
4 place Jussieu, 75252 Paris Cedex 05, France

{Nicolas.Pessemier, Lionel.Seinturier, Laurence.Duchien, Olivier.Barais}@lifl.fr

RÉSUMÉ. *Nous présentons une manière originale d'introduire les préoccupations transverses au sein d'une architecture à base de composants Fractal sous forme de liaison que nous nommons liaison transverse. Cette liaison permet de représenter de manière claire les interactions entre composants et composants d'aspect. Ces derniers incarnent les préoccupations transverses. Chaque préoccupation tissée à l'architecture logicielle modifie la structure en introduisant un nouveau composite contenant le composant d'aspect et l'ensemble des composants concerné par la coupe. Ceux-ci sont alors partagés dans ce nouveau composite. Notre représentation par liaison du tissage nous permet d'offrir une API d'introspection de coupe suffisament riche pour donner différentes vues sur l'ensemble des coupes s'appliquant sur une application.*

ABSTRACT. *We present a new way to define crosscutting concerns in a Fractal component-based architecture. We introduce a new kind of binding that we call crosscutting binding. This new binding clearly represents interactions between components and aspect components that embody crosscutting concerns. Each concern weaved to the architecture modifies the structure with introducing a new composite that contains the aspect component and the components that are affected by the pointcut. The components are shared in this new composite. Our representation of the weaving process by bindings offers a pointcut introspection API which is rich enough to give some views on the active pointcuts in a software application.*

MOTS-CLÉS : *séparation des préoccupations, ADL, composant, AOP, Fractal*

KEYWORDS: *separation of concerns, ADL, component, AOP, Fractal*

1. Introduction

19

Depuis que les langages de programmation existent, les concepteurs n'ont eu de cesse de trouver des moyens d'abstraire la représentation d'entités logicielles. Différentes approches s'appuyant sur les composants [SZY 96, SZY 02], la description d'architectures logicielles (ADL [MED 00]), ou encore sur la programmation par aspects [KIC 97] se sont présentées comme de puissants outils pour résoudre leurs problèmes de conception. Les composants permettent une réutilisation optimale, les ADL aident à préciser les interactions entre ces composants, enfin les aspects permettent de séparer les préoccupations transverses d'une application.

Dans ce papier nous présentons Fractal-AOP, une extension du modèle de composants Fractal pour la programmation par aspects. Il est ainsi possible de tisser des aspects à une application programmée à l'aide de composants Fractal. Nous introduisons ainsi un niveau de programmation par aspects complémentaire à celui pris en compte par les langages et les frameworks comme AspectJ [KIC 01], JAC [PAW 01], JBoss AOP [BUR 03] ou AspectWerkz [VAS 04].

Comme nous le verrons à la section 3, nous avons profité des capacités d'extension de l'ADL Fractal v2, pour introduire deux nouvelles balises dans les descripteurs d'architecture de Fractal. Celles-ci permettent de décrire les coupes. Par ailleurs, nous avons également défini un nouveau type de composant appelé composant d'aspect (section 3.1) pour décrire le code de traitement dit *advice* d'un aspect et un contrôleur d'interception Fractal/Julia pour gérer le ou les aspects tissés sur un composant.

Dans un article précédent [PES 04], nous avions présenté les composants d'aspect et le contrôleur d'interception. Cet article rappelle brièvement ces éléments et présente l'extension de l'ADL Fractal (section 3.2), les fonctionnalités d'instrospection de coupes (section 4) qu'offrent notre modèle et une console d'administration (section 4.4) permettant de visualiser les aspects tissés sur une application.

2. Le modèle de composant Fractal et son ADL

Il existe de nombreuses plates-formes à composants comme EJB, .NET ou CCM qui reçoivent de plus en plus d'intérêt aussi bien au niveau académique qu'industriel. Cependant ces modèles sont principalement dédiés à des composants de gros grain pour des applications de type système d'information. Les classes qui implémentent ces composants, suivent des règles de programmation, sont pointées par des descripteurs XML et doivent être exécutées par des serveurs d'applications. Elles ne peuvent pas être manipulées aussi facilement que des objets d'une machine virtuelle. De plus, les services techniques de ces plates-formes sont figés et ne peuvent pas être modifiés, retirés ou ajoutés de façon standard en fonction des besoins.

Ainsi, en dépit de cette large adoption de la communauté, demeure le besoin d'un modèle de composant léger, proche des concepts d'un langage de programmation et ne réclamant pas une mécanique lourde comme les modèles précédemment cités. Le

modèle de composant Fractal [BRU 04] répond à ces besoins. De plus, un ADL à base de descripteurs XML fait partie des sous-projets liés à Fractal.

Le modèle de composant Fractal permet la définition, la configuration, et la reconfiguration dynamique de composants et offre une séparation claire entre les besoins fonctionnels et non fonctionnels d'une application. Construit comme un modèle de haut niveau, son objectif est de fournir une grande modularité et des possibilités d'extension étendues. Le modèle est récursif : un composant est de type primitif ou composite. Dans ce dernier cas, le composant correspond à un assemblage d'autres composants primitifs ou composites. Un composant peut également être partagé entre différents composites.

Les interfaces jouent un rôle central dans Fractal. Il en existe deux catégories : les interfaces fonctionnelles et celles de contrôle. Les interfaces fonctionnelles sont les points d'accès externes d'un composant. Fractal offre des interfaces client et serveur. Une interface serveur reçoit des opérations d'invocation et une interface cliente en émet. Ainsi une liaison Fractal représente une connexion entre deux composants (liaison primitive). Des liaisons de type multiples (liaisons composites) sont autorisées. Le modèle est fortement typé. De ce fait le type d'une interface serveur doit être un sous-type de l'interface client à laquelle elle est reliée.

Comme leur nom l'indique, les interfaces de contrôle permettent un certain niveau de contrôle du composant auquel elles sont associées. Ces interfaces ont à charge les besoins non fonctionnels du composant, par exemple la gestion du cycle de vie ou des liaisons à d'autres composants.

2.1. *Programmer avec Fractal*

Cette section illustre par un exemple les concepts du modèle de composant Fractal. La figure 1 représente le modèle d'une station service. Chaque rectangle correspond à un composant. Les clients utilisent un pistolet pour remplir leur réservoir et payent leur carburant à une caisse connectée à une banque. Chacun de ces composants est un composant primitif Fractal. Nous avons représenté deux composants composites : le premier pour la station, le second pour le système complet. Les composites correspondent à un assemblage de composants primitifs ou composites. Les «T» attachés aux composants représentent les interfaces fonctionnelles (les interfaces de contrôle ne sont pas représentées sur la figure). Les flèches caractérisent les liaisons entre composants et sont orientées : une interface cliente vers une interface serveur.

Fractal met à disposition une API permettant de créer, introspecter et gérer les composants, leurs interfaces et leurs liaisons. Par exemple un composant peut être démarré et arrêté et des liaisons peuvent être créées dynamiquement.

La définition de l'architecture d'une application s'effectue grâce à un descripteur XML de l'ADL Fractal. La figure 2 présente un extrait de cette définition dans notre exemple de la station service.

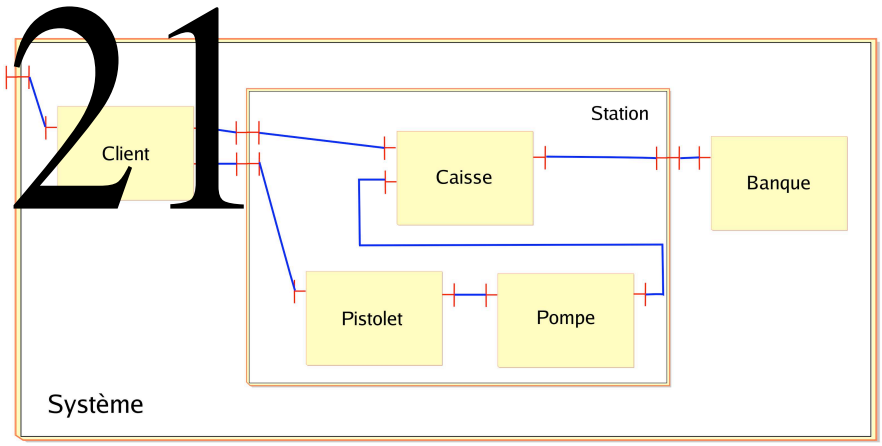

Figure 1. *Une station service avec des composants Fractal.*

```
<!-- Définition d'un composant composite -->
<definition name="station">
   <!-- Le composant station offre ou utilise des interfaces -->
   <interface name="gunProvideGas" role="server"
                        signature="station.GunProvideGas"/>
   <interface name="bankAuth" role="client"
                        signature="station.BankAuth"/>
   <!-- Le composite contient les composants : -->
   <!-- pump, gun et cashRegister -->
   <component name="pump">
      <!-- ... -->
   </component>
   <!-- Liaison des interfaces -->
   <binding client="this.cashRegisterUserInterface"
            server="cashRegister.cashRegisterUserInterface" />
   <binding client="this.gunProvideGas"
                                server="gun.gunProvideGas" />
</definition>
```

Figure 2. *L'architecture logicielle de la station service avec l'ADL Fractal.*

3. Extension du modèle Fractal pour le support de préoccupations transverses

L'extension que nous avons réalisée enrichit un assemblage initial de composants métiers avec un assemblage implémentant des préoccupations transverses. En bref, il est possible de définir des mécanismes de tissage et de coupe au niveau même de l'architecture. Ainsi un nouveau type de liaison a été introduit. Cette nouvelle liaison

peut caractériser par exemple la redirection de tous les appels sortant du composant `caisse` (`CashRegister`) pour permettre à une fonction de cryptage de coder les informations relatives à un client (son numéro de carte banquaire) et symétriquement la redirection des appels entrants du composant `banque` pour les décoder.

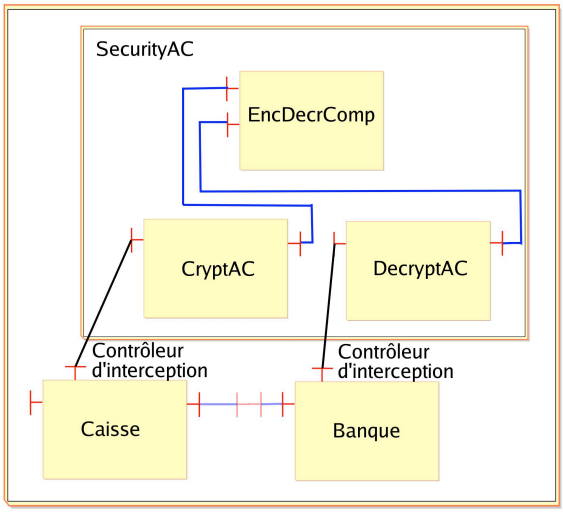

Figure 3. *Un composant d'aspect pour une préoccupation de sécurité.*

A ce stade nous avons besoin de deux éléments pour réaliser notre extension. Il est d'abord nécessaire de caractériser la notion d'aspect qui incarne une préoccupation transverse. Ensuite un mécanisme de tissage de ces préoccupations transverses à l'architecture logicielle doit être introduit. Nous décrivons ces éléments dans les sous-sections suivantes.

3.1. *Composant d'aspect*

Un composant d'aspect (AC pour *Aspect Component*[1]) représente la définition d'une préoccupation transverse. La figure 3 représente notre préoccupation de sécurité : composite `SecurityAC`. Ce composite est composé de trois sous-composants : les composants `CryptAC` et `DeCryptAC` qui sont des composants d'aspect et `EncrDecrComp` qui est un composant Fractal classique offrant les fonctions mathématiques nécessaires pour le codage et le décodage de données.

1. le terme *Aspect Component* provient de nos travaux précédents sur JAC [PAW 01], qui est un framework dynamique d'aspect en Java.

Les composants CryptAC et DeCryptAC sont des composants d'aspect car ils possèdent une interface serveur particulière AspectComponent conforme à l'API AOP Alliance [2]. Cette interface met en œuvre l'interface MethodInterceptor qui définit la méthode suivante.

```
public Object invoke( MethodInvocation mi ) throws Throwable {
  // before method call
  Object ret = proceed(); // proceed the method call
  // after method call
  return ret;
}
```

La méthode invoke exprime le code qui doit être exécuté avant et après le point de jonction. L'objet de type MethodInvocation en paramètre de la méthode invoke contient une méthode proceed pour exécuter le point de jonction.

Un composant d'aspect incarne le comportement à appliquer autour d'un point de jonction, c'est-à-dire une méthode d'aspect (*advice*) dans la terminologie AspectJ. Pour rendre l'interception de méthode effective, les composants CryptAC et DecryptAC doivent être reliés aux contrôleurs d'interception des composants CashRegister et Bank. Ce contrôleur d'interception permet donc d'intercepter les appels entrants et sortants d'un composant pour les rediriger vers des composants d'aspects. Nous détaillons ce contrôleur dans la sous section suivante.

Pour résumer, une préoccupation transverse s'incarne à travers un composant Fractal spécial (un composite dans notre exemple, ou simplement un composant primitif) appelé composant d'aspect qui offre au moins une interface AspectComponent.

3.2. *Liaisons transverses*

Dans notre approche, tisser un composant d'aspect à une architecture est un processus très comparable à une liaison entre deux composants. La différence réside dans le fait que l'interface cliente est ici une interface de contrôle (voir figure 3). Cette caractéristique inhabituelle en Fractal, est néanmoins employée par ailleurs dans d'autres approches mariant Fractal et AOP [FAK 04]. Nous avons mis en place deux façons différentes de réaliser la liaison, que nous appelons liaison transverse, entre un composant Fractal et un composant d'aspect : soit directement, soit de manière déclarative par une expression de coupe.

---

2. AOP Alliance <sourceforge.net/projects/aopalliance> est une initiative open-source pour une définition d'une API commune entre frameworks AOP. Cette API est mise en oeuvre par Spring et JAC entre autres.

3.2.1. *Liaison transverse directe*

Cette liaison démarre du composant aspectisé pour se terminer au composant d'aspect, une liaison transverse est alors directement créée entre les deux composants. Toutes les méthodes du composant seront interceptées par ce composant d'aspect. Cette liaison est semblable à la méta-liaison entre un composant de base et un méta-composant.

Une liaison transverse directe peut être définie de deux manières différentes. La première par API et donc par la méthode suivante, celle-ci prend en paramètre le composant d'aspect s'appliquant sur le composant du contrôleur d'aspect sur lequel la méthode est invoquée :

```
public void bindAC(Component ac);
```

La seconde grâce à l'ADL sous la forme d'une balise comme présenté ci-dessous :

```
<!ELEMENT bindAC (comment*) >
<!ATTLIST bindAC
  client CDATA #REQUIRED
  ac CDATA #REQUIRED
>
```

Le champ `client` correspond au nom du composant aspectisé et le champ `ac` correspond au nom du composant d'aspect à inscrire au niveau du contrôleur d'aspects du composant.

3.2.2. *Liaison transverse par une expression de coupe*

Dans cette situation, une expression de coupe est nécessaire. Elle est composée de trois expressions régulières qui affectent le nom des composants, les noms d'interfaces et les signatures de méthodes. Toutes les méthodes qui vérifient ces trois expressions sont aspectisées par le composant d'aspect. Ce tissage se réalise par l'appel d'une méthode `weave` sur un composite racine de l'application. La méthode permet de traverser récursivement toute la hiérarchie de composants de ce composite racine et de trouver toutes les interfaces et méthodes qui respectent la coupe définie. Cette technique est proche du mécanisme de tissage offert par les frameworks et compilateurs d'aspects.

A la manière de la liaison directe, deux possibilités sont offertes pour l'établissement d'un `weave`. Il est possible de définir le tissage par API ou par l'ADL en XML. Sur l'exemple suivant nous définissons un tissage depuis le composant racine `root` du composant d'aspect `cryptAC` sur les composants nommés `cashRegister` possédant des méthodes dont la signature commence par `askAuth` :

```
<weave root = "root"
       ac = "cryptAC"
```

```
componentExp = "cashRegister"
interfaceExp = ".*"
methodExp = "askAuth.*"
pointcutName = "cryptPointcut" />
```

3.3. *Tissage de plusieurs composants d'aspect*

Lorsque plusieurs composants d'aspects sont tissés, soit directement, soit par une expression de coupe sur le même composant de base, leur ordre d'exécution est celui de leur tissage : i.e. les composants d'aspect tissés en premier sont exécutés en premier. A titre de travail futur, nous prévoyons de compléter notre prototype pour permettre de définir des ordres de précédence globaux ou locaux entre les aspects comme cela existe pour d'autres langages ou frameworks AOP (par exemple AspectJ [KIC 01] ou JAC [PAW 01]).

3.4. *Utilisation des composants partagés*

Lorsqu'une coupe est définie par une balise de type `weave` au niveau de l'ADL, les sous-composants du composant racine (`root`) sont analysés de façon récursivement. Chacun de ces sous-composants vérifie si l'expression régulière correspond à la description de ses interfaces et méthodes. Lors de l'analyse de cette balise `weave`, un pré-traitement est effectué sur l'architecture, le nom de la coupe (le paramètre *pointcutName* dans l'exemple précédent) va se transformer en composite contenant à la fois le composant d'aspect `cryptAC` ainsi que tous les composants répondant à la coupe définie par le `weave`. Ces composants sont alors partagés. Un composant est partagé lorsqu'il appartient à deux composites différents ne possédant aucun composant encapsulant commun.

En bref, par cette modification d'architecture, nous obtenons pour chaque coupe un composite caractérisant la coupe et contenant le composant d'aspect de la coupe ainsi que tous les composants aspectisés par cet aspect. La figure 4 montre l'architecture obtenue après tissage de nos deux composants d'aspects `cryptAC` et `deCryptAC` (les liaisons ne sont pas représentées).

Dans les deux cas énoncés dans cette section, les liaisons établies, qu'elles soient directes ou par expressions de coupe, peuvent être manipulées dynamiquement. Elles peuvent être rompues ou reconnectées pour modifier la stratégie de coupe associée au composant d'aspect. Ainsi ce comportement reste très proche de celui de liaison entre composants métiers.

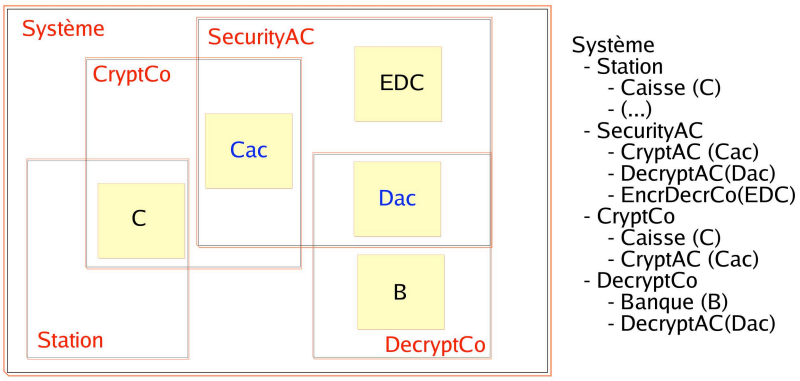

Figure 4. *Architecture de la station après la coupe.*

4. Introspection de coupes

Nous l'avons vu dans la section précédente, les liaisons transverses que nous définissons ont un comportement très souple et similaire à celui des liaisons Fractal entre composants métiers. De ce fait, les possibilités d'introspection offertes par le modèle Fractal peuvent être utilisées pour garder une vue fidèle de notre architecture aussi bien d'un point de vue fonctionnel que non fonctionnel. Nous avons de plus défini une API d'introspection de coupes. Nous sommes ainsi en mesure d'obtenir des informations relatives aux coupes effectuées sur l'architecture, les composants aspectisés par ces coupes ou aspectisable par une coupe donnée.

4.1. *Les composants d'aspect d'un composant*

La première possibilité offerte par notre modèle étendu est de visualiser la liste des composants d'aspects s'appliquant sur un composant donné. En comparant à une approche AOP s'appliquant sur des objets, cela revient à obtenir la liste des aspects tissés sur un objet quelconque. A notre connaissance, aucune autre approche AOP n'offre ce genre de mécanisme. Avec AspectJ par exemple, une fois les aspects tissés à l'application, il n'est plus possible d'avoir une représentation de l'architecture avant tissage, ni les aspects s'appliquant à chaque objet. D'une manière générale, les approches procédant par expression de coupe perdent un certain nombre d'informations une fois le tissage (dynamique ou statique) réalisé.

La présentation exhaustive du code Java de notre API d'introspection de coupe est hors du cadre de cet article. Nous nous contentons d'en donner quelques brefs

aperçus ci-dessous. Ainsi, à titre indicatif, la méthode suivante permet de récupérer les composants d'aspect d'un composant :

```
public Component[] listAC();
```

4.2. *Obtenir les composants aspectisés par un composant d'aspect*

Une autre possibilité de notre API consiste à donner la liste des composants aspectisés par un composant d'aspect particulier. Ainsi profitant des possibilités d'introspection de Fractal, la structure des sous-composants peut être analysée récursivement et nous pouvons construire une liste des composants actuellement affectés par ce composant d'aspect. Notons que grâce à un outil graphique de représentation d'une architecture à base de composants Fractal, il est déjà possible de retrouver ce genre d'information puisque dans notre modèle un aspect correspond à un composant.

Les composants aspectisés par un composant d'aspects sont obtenus à l'aide de la méthode :

```
public String[] listCrosscutComps(Component root, String acName);
```

4.3. *Obtenir les composants potentiellement aspectisés par une coupe*

Un dernier exemple des possibilités offertes par notre API est la possibilité de simuler une coupe pour obtenir une représentation précise de l'ensemble des composants, interfaces et méthodes concernés par l'expression de coupe donnée en paramètre. Ainsi par différence avec la méthode précédente, il est possible de voir si la structure de l'application a été changée et si les composants aspectisés par une coupe donnée à un instant donné le sont toujours à l'instant suivant. On peut alors clairement distinguer les coupes dynamiques des coupes statiques.

Les composants potentiellement aspectisés par une coupe sont obtenus via la méthode :

```
public Pointcut aspectizableComps(Component root, PointcutExp pExp);
```

4.4. *Un outil graphique d'administration et de représentation de coupes*

Browser Framework[3] est une console d'administration générique qui permet à des utilisateurs d'interagir avec un ensemble d'objets quelconques. Le framework est générique et se configure à partir d'une grammaire spécifique (DTD XML) qui peut être étendue. Plusieurs consoles d'administrations ont été réalisées dont notamment la

3. `openccm.objectweb.org/doc/0.8.1/browser_plugin.html`

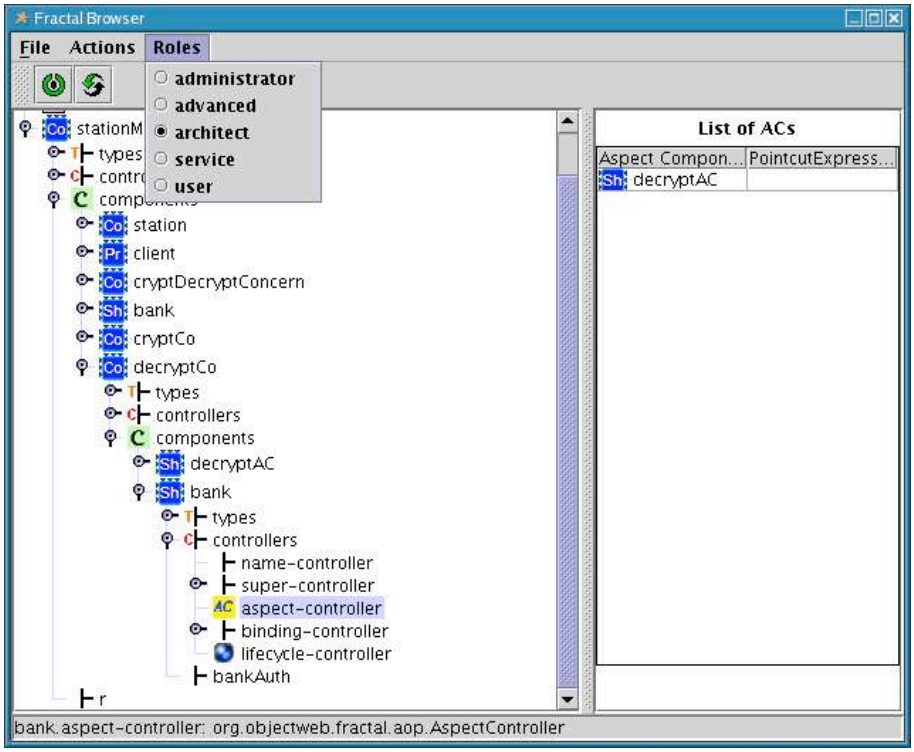

Figure 5. *Le framework Fractal Browser.*

console d'administration d'OpenCCM [MAR 01]. Une implémentation est disponible pour le modèle Fractal. Cette version a été étendue pour qu'elle puisse prendre en compte le contrôleur de composants d'aspect.

L'outil permet de représenter sous forme d'arbre un ensemble de ressources (ici des composants). La figure 5 propose une capture d'écran de l'outil, sur le panneau de droite nous pouvons observer la liste des composants d'aspect du composant `bank`. Sur cette exemple `bank` est aspectisé par le composant d'aspect `cryptAC`.

5. Travaux connexes

Dernièrement des approches ont essayé de combiner composants et aspects. Cette section en présente quelques uns.

Fractalk-AOP [FAK 04] est une autre approche qui étend le modèle de composants Fractal avec des aspects. Une première différence entre ce modèle et le nôtre, Fractal-AOP, réside dans la gestion du tissage. Alors que nous envisageons le tissage comme une liaison primitive entre des composants d'aspect et des composants de base

(cette liaison peut être multiple lorsque le même aspect impacte plusieurs composants de base), Fractalk-AOP introduit une liaison composite qui met en jeu une instance de composant dit de tissage. Le choix d'une liaison primitive limite le nombre d'instances et amène selon nous une architecture plus simple à gérer. Dans [FAK 04], il est mentionné que le composant de tissage gère aussi la coupe. En ce qui concerne Fractal-AOP la coupe est spécifiée dans l'extension de l'ADL. Par ailleurs, les points de jonction envisagés par Fractalk-AOP sont différents des nôtres : nous nous limitons aux éléments architecturaux présents dans Fractal (interfaces, méthodes et donc appels ou réception de méthodes), laissant de côté les points de jonction de type lecture-écriture d'attributs qui appartiennent selon nous au niveau objet. A contrario Fractalk-AOP prend en compte ces points de jonction et d'autres (connexion, deconnexion, ajout, suppression de composants, changement d'état, changement de structure). La granularité des points de jonction n'est donc pas tout à fait la même. Finalement, au niveau de l'implémentation Fractalk-AOP vise Smalltalk et Fractal-AOP Java.

JAsCo [SUV 03] étend le modèle Java bean en introduisant les notions d'*aspect beans* et de *connectors*. Un *Aspect bean* décrit où et quand (les notions d'*advice* et de coupe en AOP) appliquer un comportement indépendant de tout contexte, en utilisant un type de classe interne appelé *hook*. Les *connectors* déploient les *hooks* dans un contexte spécifique. Un des grands apports de JAsCo réside dans son langage de coupe, ses *connectors* permettant de définir un contrôle plus fin sur l'ordre des aspects à exécuter, que le langage de coupe d'AspectJ. Finalement nous pouvons remarquer que la gestion de *hooks* et de *connectors* est complètement centralisée et gérée par un registre de *connectors* qui a été récemment optimisé grâce à HotSwap and Jutta [SUV 04] dans le but de réduire le coût dont souffre toute approche dynamique.

DAOP [PIN 02] est une plate-forme dynamique distribuée où aspects et composants sont des entités de première ordre composées à l'exécution par une couche intergicielle. Comme JAsCo, la gestion des composants et aspects est complètement centralisée et toutes les informations sur l'architecture et ses entités sont enregistrées dans la couche intergicielle. Dans une première phase, aspects et composants sont décrits par un langage spécifique qui permet la définition des interfaces des rôles et des liaisons. Ensuite, à l'exécution les composants et les aspects sont liés en suivant les spécifications de la couche intergicielle. L'originalité de l'approche DAOP réside dans le nom de rôle unique attribué à chaque entité. De cette manière les communications sont effectuées en précisant ces noms et non des références d'objets.

Jiazzy [MCD 03] étend le langage Java pour des composants binaires compilés séparément et liés de manière externe en tant qu'unités. Les unités sont des conteneurs de classes Java pré-compilées et peuvent être de deux types : les *atoms* (construits depuis des classes Java) ou les *compounds* (construits depuis d'autres *atoms* ou *compounds*). De nouveaux comportements peuvent être ajoutés aux méthodes et attributs sans modifier le code source, grâce à des mécanismes de classe ouverte (*open classes*) et de signature ouverte (*open signature*) basées sur l'utilisation des *mixins*. Pour résumer, Jiazzy effectue une séparation des préoccupations au niveau des classes et offre des

mécanismes d'enrichissement grâce aux *mixins* qui s'avèrent moins puissants qu'une approche traditionnelle en AOP comme AspectJ.

JBoss AOP [BUR 03] est un projet qui amène les mécanismes de l'AOP au niveau des applications à base d'EJBs. JBoss AOP permet de modifier une application avec des aspects, d'y introduire de nouvelles fonctionnalités grâce aux *mixins*, ainsi que la gestion de méta données. Les méthodes d'aspect sont programmées grâce à des classes Java qui implémentent une API d'interception. Les coupes sont définies dans des descripteurs XML et tissent les intercepteurs à l'application. Le tissage proposé par ce framework est dynamique.

6. Conclusion

Nous avons présenté une utilisation des composants partagés pour la représentation de préoccupations transverses dans une application à base de composants.

Dans notre modèle un aspect est un composant d'aspect qui coexiste au milieu des composants métiers de l'application. Tisser un aspect à un ensemble de composants revient à définir une liaison que nous appelons liaison transverse. Toute définition de liaison transverse modifie la structure applicative pour partager le composant d'aspect avec les composants qu'il affecte dans un composite représentant la préoccupation transverse.

Notre approche permet donc la définition de liaisons fonctionnelles et non fonctionnelles sur une architecture à base de composants par tissage de composants d'aspect. Ceci peut être effectué par API ou au niveau ADL grâce à un descripteur XML.

Nous offrons des mécanismes d'introspection de coupes comme la possibilité d'avoir une représentation précise de l'ensemble des aspects s'appliquant sur un composant ou encore de connaître l'ensemble des composants affectés par un composant d'aspect.

Nos travaux futurs vont se tourner vers l'élaboration d'un modèle de transformation d'architecture par le tissage de nos composants d'aspects. Notre langage de coupe doit être enrichi car il reste actuellement limité à trois expressions régulières sur les noms de composants, d'interfaces et de signatures de méthodes. Nous nous orienterons par la suite vers une formalisation du modèle de tissage.

7. Bibliographie

[BRU 04] 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, May 2004.

[BUR 03] BURKE B., AL., « JBoss-AOP », 2003, `www.jboss.org/developers/projects/jboss/aop`.

[FAK 04] FAKIH H., BOURAQADI N., « Les aspects et les composants logiciels », *1ère Journée Francophone sur Développement de Logiels Par Aspects (JFDLPA'04)*, septembre

2004.

[KIC 97] KICZALES G., LAMPING J., MENDHEKAR A., MAEDA C., LOPES C., LOINGTIER J., IRWIN J., « Aspect-Oriented Programming », *Proceedings of the 11th European Conference on Object-Oriented Programming (ECOOP'97)*, vol. 1241 de *Lecture Notes in Computer Science*, Springer, juin 1997, p. 220-242.

[KIC 01] KICZALES G., HILSDALE E., HUGUNIN J., KERSTEN M., PALM J., GRISWOLD W., « Getting started with ASPECTJ », *Communications of the ACM*, vol. 44, n° 10, 2001, p. 59-65, ACM Press.

[MAR 01] MARVIE R., MERLE P., GEIB J.-M., VADET M., « OpenCCM : une plate-forme ouverte pour composants CORBA », *Actes de la seconde Conférence Française sur les Systèmes d'Exploitation (CFSE'2)*, Paris, France, Avril 2001.

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

[MED 00] MEDVIDOVIC N., TAYLOR R., « A Classifi cation and Comparison Framework for Software Architecture Description Languages », *IEEE Transactions on Software Engineering*, vol. 26, 2000.

[PAW 01] PAWLAK R., SEINTURIER L., DUCHIEN L., FLORIN G., « JAC : A Flexible Solution for Aspect-Oriented Programming in Java », *Proceedings of Reflection'01*, vol. 2192 de *Lecture Notes in Computer Science*, Springer, septembre 2001, p. 1-24.

[PES 04] PESSEMIER N., SEINTURIER L., DUCHIEN L., « Components, ADL & AOP : Towards a common approach », *Workshop on Reflection, AOP and Meta-Data for Software Evolution at ECOOP'04*, juin 2004.

[PIN 02] PINTO M., FUENTES L., FAYAD M., TROYA J., « *Separation of Coordination in a Dynamic Aspect Oriented Framework*. », *In the 1st International Conference on Aspect-Oriented Software Development*, April 2002.

[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 (AOSD'03)*, ACM Press, 2003, p. 21–29.

[SUV 04] SUVÉE D., VANDERPERREN W., « Optimizing JAsCo dynamic AOP through HotSwap and Jutta », *Proceedings of the Dynamic Aspects Workshop (DAW) at AOSD'04*, March 2004.

[SZY 96] SZYPERSKI C., PFISTER C., « Why Objects are Not Enough », *Proceedings of the International Component Users Conference*, 1996.

[SZY 02] SZYPERSKI C., *Component Software - Beyond Object-Oriented Programming*, Addison-Wesley, 2nd édition, 2002.

[VAS 04] VASSEUR A., « Dynamic AOP and Runtime Weaving for Java - How does AspectWerkz Address it ? », rapport, 2004, AOSD'04.