Rapport de Stage - DEA RACOR

Une mise en œuvre du modèle de composants Fractal en Smalltalk

Réalisé par :

Ali HAMADI
hamadi@ensm-douai.fr

Encadré par :

Dr. Abdelhak-Djamel SERIAI
GIP, École des Mines de Douai
seriai@ensm-douai.fr

Dr. Noury BOURAQADI
GIP, École des Mines de Douai
bouraqadi@ensm-douai.fr

École Nationale Supérieure des Mines de Douai
Septembre 2004

Résumé

Le paradigme "*composant*" est apparu en réponse aux limites de l'approche de conception par objets. Il a introduit une nouvelle méthode pour la conception des applications logicielles. Cette méthode dénommée CBSE (*Component Based Software Engineering*) est basée sur l'assemblage d'entités logicielles appelées *composants*.

Pour mettre en œuvre ces composants logiciels, il est nécessaire de disposer, d'un côté, d'un modèle abstrait de composants qui définit les entités et leur mode d'interaction et de composition, et d'un autre côté, d'une infrastructure à composants, qui met en œuvre le modèle abstrait et permet de construire, déployer, administrer et exécuter des applications conformes à ce modèle.

Ainsi, à travers ce rapport nous étudions le modèle Fractal comme modèle de composants logiciels, et nous proposons notre plate forme en Smalltalk, que nous appellerons : FracTalk, permettant de valider le modèle abstrait de Fractal, et de créer et d'exécuter des applications à base de composants Fractal.

Mots-Clés : Composant logiciel, Fractal, réflexivité, récursivité, Smalltalk, UML.

Remerciemment

Je tiens à remercier très sincèrement :

Monsieur Abdelhak-Djamel Seriai, enseignant-chercheur à l'ENSM-Douai, qui m'a accueilli dans le laboratoire CSL et dirigé mon travail de recherche. Je tiens ici à lui témoigner toutes mes reconnaissances pour son aide, sa disponibilité, et son encouragement tout au long de ce stage. J'apprécie la confiance qu'il m'a témoigné et les conseils avisés qu'il m'a prodigué.

Monsieur Noury Bouraqadi, enseignant-chercheur à l'ENSM-Douai, pour ses critiques et ses remarques apportées, et pour les discussions qu'on a eu ensemble.

Mes amis : G.Abdelaziz et F.Houssam, doctorants à l'ENSM-Douai, S.Abdallah et L.Gabriel, stagiaires à l'ENSM-Douai, pour leur soutien, leur aide constante, leurs encouragements, et l'ambiance remarquable qu'ils ont créée.

Je tiens en outre à remercier toutes les personnes qui ont contribué de loin au bon déroulement de mon travail. Je les remercie chaleureusement.

Table des matières

Chapitre 1

Introduction Générale

Le développement de systèmes complexes à base de composants (CBSE :*Component Based software Engineering*) suscite aujourd'hui beaucoup d'intérêts aussi bien dans le monde de la recherche que dans celui de l'industrie. Plusieurs modèles industriels ou académiques pour les composants logiciels ont été proposés (EJB, CCM, Fractal,...etc).

Le paradigme composant vient en réponse au problème de complexité de gestion des logiciels. Il considère toutes les étapes du cycle de vie des applications et s'intéresse non seulement à leur facilité de programmation, mais également à leur facilité d'administration.

Ces dernières années ont vu émerger différentes plates-formes à composants telle que les EJBs ou CCM. Mais il s'est avéré que leurs spécifications sont complexes et leurs mises en œuvre sont plus complexes. De ce fait, la communauté académique a proposé un autre modèle : le modèle de composants Fractal, dont l'objectif principal est de faciliter la spécification, la construction, le déploiement, et la reconfiguration d'applications à base de composants.

Le modèle Fractal est un système général, minimal, et extensible. Il n'est pas dédié à un langage ou un environnement d'exécution particulier. Le modèle est principalement défini à travers cinq concepts : *composant*, *interface*, *liaison*, *contrôleur*, *contenu*. Et il est caractérisé par la *récursion*, la *réflexion*, et le *partage*.

La problématique à étudier, à travers ce travail de stage, est la mise en oeuvre du modèle abstrait de composants Fractal en utilisant le langage Smalltalk. L'objectif étant triple :

1. Étudier de manière rigoureuse une projection du modèle abstrait de Fractal vers les éléments d'un langage de programmation objet (en l'occurrence smalltalk). Ce modèle d'implantation doit être clair, modulaire et extensible.
2. Exploiter toute la richesse du langage de Smalltalk (réflexivité, méta-programmation,...) pour la réalisation des propriétés du modèle Fractal.
3. Réaliser la séparation du modèle abstrait Fractal des spécificités d'une implémentation en Java.

Notre travail a consisté à étudier le modèle de composant Fractal et le langage de programmation Smalltalk pour proposer par la suite une implantation du modèle abstrait Fractal en Smalltalk qu'on a nommé : FracTalk.

Le reste du rapport est organisé de la façon suivante : Le chapitre 2 présente un aperçu général sur les composants logiciels et leurs modèles suivi d'une description du modèle Fractal. Nous dressons à la fin une synthèse détaillée de cet état de l'art.

Dans le chapitre 3, nous décrivons la manière dont on a mis en oeuvre le modèle Fractal en Smalltalk : de la conception à la réalisation.

Nous concluons enfin en récapitulant les principaux résultats obtenus dans cette étude, tout en évoquant certains axes de recherche, dans le prolongement de notre travail, et qui nous paraissent souhaitable d'explorer.

Chapitre 2

Etat de l'Art

2.1 Introduction

L'objectif de ce chapitre est de montrer les notions de base de la programmation orientée-composants, à savoir les composants logiciels et leurs architectures (les modèles), tout en mettant l'accent sur le modèle Fractal : principe, spécification et implantations existantes.

2.2 Composants Logiciels

2.2.1 Pourquoi les composants logiciels ?

L'avènement du modèle objet a bel et bien permis de combler les insuffisances des modèles qui existaient dans plusieurs côtés [MP02]. Il a permis de renforcer les aspects concernant la sécurité et l'interopérabilité entre entités hétérogènes à travers le concept de "l'encapsulation". Avec un tel modèle, de nouveaux outils ont vu le jour. Par conséquent, les applications réparties ont connu une grande évolution, et avec l'émergence des systèmes dit ouverts (distribué et constamment en évolution), le modèle objet, bien qu'il a abordé l'évolution des objets, ne propose pas une véritable solution pour prendre en charge ce type de système, et a connu quelques défauts tels que [GS03b, MP02] :

- La structure de l'application est généralement peu visible : l'ensemble de fichiers de codes est nécessaire ;
- La difficulté de la modification (ajout/suppression) de fonctionnalités si cela s'avère nécessaire ;
- La construction de l'application est prise totalement en charge par le programmeur : construction des différents modules, définition des instances et interconnexion des modules, etc.

De ces faits, le modèle basé sur le concept de composants logiciels est apparu pour répondre au mieux aux limites de l'approche de conception par objets.

2.2.2 Qu'est ce qu'un composant logiciel ?

"*Un composant logiciel est une unité de composition spécifiant, par contrats, ses interfaces (fournies et requises) et ses dépendances explicites aux contextes. Un composant logiciel peut être déployé indépendamment et peut être sujet de composition par un tiers pour la conception d'applications logicielles*". Cette définition[1]est donnée par les participants à la première édition de *Workshope on Component Oriented Programing* [SP96].

Une autre définition, relativement consensuelle, à la notion de composant logiciel est la suivante : " *les composants logiciels sont des unités de composition de logiciels* "[PMB00]. Certes, cette proposition est peu satisfaisante mais elle est le point de convergence de nombreux travaux autour du concept de composant. Ainsi, issue de la recherche, la structure d'un logiciel est vue comme étant un assemblage, ou plutôt une interconnexion, de composants. Du point de vue industriel, les besoins en terme de composants logiciels s'expriment plutôt par la nécessité de disposer de technologie permettant de composer des applications à partir d'éléments logiciels réutilisables. Cette notion de réutilisation [GS03a, Chi98] est une propriété importante de l'approche orientée composant. Pour comprendre cette propriété, il peut être utile et intéressant de faire l'analogie avec les composants électroniques -qui servent à mettre en place des circuits électroniques- dont chacun d'eux est considéré comme une boîte noire pour laquelle il suffit de connaître les services rendus, les services utilisés et les règles d'interconnexion.

La conception d'une application selon une approche orientée composant consiste donc à voir une application comme une collection de composants logiciels indépendants, interconnectés à l'aide d'une plate-forme de communication. Chaque composant possède un certain nombre de ports qui sont ses points d'entrée et de sortie, c'est à dire les services qu'il est capable de rendre et les services dont il a besoin pour fonctionner.Et donc l'interconnexion consiste à relier ces différents points entre eux.

En conséquence, pour mieux clarifier la définition de base d'un composant logiciel, il suffit de retenir les trois caractéristiques suivantes [Szy98] :

- Un composant logiciel est une unité de composition spécifiant, par contrat, ses interfaces (fournies et requises) et ses dépendances explicites aux contextes ;
- Un composant logiciel peut être déployé indépendamment (l'installation sur différentes plates-formes) ;
- Un composant est capable de s'auto décrire, ce qui permet aux constructeurs d'applications de l'utiliser facilement sans qu'ils aient besoin d'en connaître son fonctionnement.

2.2.3 Développement d'applications à base de composants

L'ingénierie de conception des applications logicielles à base de composants CBSE[2] est née du besoin de faciliter, le plus possible, la réutilisation du code, ce qui permet de concevoir des applications robustes et réduites très rapidement. En fait, pour créer une application, les développeurs interconnectent, d'une manière homogène et à l'aide d'une plate-forme, des entités

[1]La version originale de la définition du composant :"*A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties*"

[2]Component Based Software Engineering

logicielles (les composants) dont les sources (origines) sont éventuellement hétérogènes [MP02]. De cette façon, au lieu de réécrire systématiquement le même type de code en commençant de zéro, ils peuvent utiliser des composants logiciels standards, et ajuster simplement les éléments variant d'une application à l'autre. En résumé, la méthode CBSE vise principalement à :

- **Réduire le temps** de développement des applications logicielles ;
- **Diminuer le coût** de production et de maintenance des logiciels ;
- **Simplifier le développement** des applications par l'utilisation des parties de code préexistantes ;
- **Améliorer la qualité et la fiabilité des logiciels** qui résulte de l'utilisation des composants programmés par des spécialistes et testés par des experts ;
- **Fournir des possibilités de décomposition** qui permettent de modifier plus facilement les applications au fur et à mesure de l'évolution des besoins des utilisateurs.

De nombreuses solutions à base de composants sont proposées aujourd'hui et qui forment l'ensemble des modèles de composants décrits dans la section suivante.

2.2.4 Modèles de composants

Principalement il existe trois catégories de modèles de composants logiciels [MP02] :

- **Modèles académiques** : dont l'objectif est l'étude spécifique sur des aspects particuliers des modèles de composant. Exemples : Darwin, Unicon, Olan, Javapod et Fractal.
- **Modèles industriels** : ces modèles répondent aux besoins des producteurs d'applications. Exemples : COM, DCOM, COM+, JavaBeans,et Entreprise JavaBeans(EJB).
- **Modèles de références** : l'objectif principal est de définir des normes tout en tenant compte des résultats académiques, tels que : ODP et Corba Component Model (CCM).

A l'heure actuelle, les modèles les plus utilisés sont en nombre de trois [Don01], encore appelés composants métiers : EJB de Sun MicroSystems, COM+ de Microsoft et CCM de l'OMG [Rei01]. Pour plus de détail, une étude est proposée dans [MP02]. Pour notre étude, nous travaillons sur le modèle de composants Fractal. Ce choix est motivé d'une part, par l'engouement et le succès de ce modèle dans la communauté "*composant*" et d'autre part par la qualité de ce modèle de composants. La section suivante aura pour objectif de décrire le modèle Fractal et les implémentations existantes.

2.3 Modèle Fractal

2.3.1 Principe

Le modèle de composant Fractal est un modèle général de composants logiciels, "modulaire" et "extensible" qui permet d'implémenter, de déployer, et de gérer des systèmes et applications complexes sur différentes plates formes avec divers langages de programmation [BCS04]. Ce modèle est spécifié conjointement par France Telecom R&D et l'INRIA Rhône-Alpes dans le cadre du consortium Objectweb [3]. Ce projet a vu le jour à partir de janvier

[3] http ://www.objectweb.org

2002. Fractal est un modèle développé pour répondre aux limitations des modèles actuels de composants logiciels. Fractal se caractérise par [BCL+04, Kra03] :

- Un fondement sur une base mathématique rigoureuse (réflexion et récursivité d'où la nomination du modèle : Fractal).
- Une composition (assemblage) hiérarchique de composants
- Un partage de composants.
- Une possibilité d'adaptation et de reconfiguration.
- Une présence d'outils de description globale.

2.3.2 Spécification

Un composant Fractal est une entité logicielle exécutable qui est encapsulée et a une identité distincte. Un composant Fractal est composé d'un ***contrôleur***, appelé également ***membrane*** et d'un ***contenu***. Le contrôleur ; à travers des points d'accès nommés **Interfaces** ; exerce (par réflexion) un contrôle arbitraire sur le contenu qui est lui-même défini récursivement comme un ensemble de sous-composants. Ainsi, une interface est le point d'accès vers le composant (similaire à un " port " dans les autres modèles de composants) [BCL+04] et qui supporte un nombre fini d'opérations. Il existe deux sortes d'interface :

- **Les interfaces serveurs** : correspond aux points d'accès acceptant des invocations d'opérations entrantes qui indiquent les services offerts par le composant.
- **Les interfaces clients** : correspond aux points d'accès supportant des invocations d'opérations sortantes qui indiquent les services qu'utilise le composant pour réaliser ses propres services.

Chaque interface d'un composant Fractal a un nom unique le distinguant des autres interfaces de ce composant.

- Le contenu

Le contenu d'un composant Fractal est constitué d'un ensemble d'autres composants, nommés **sous-composants** qui peuvent être soient :

- **Primitifs** :Une sorte de composant Fractal qui encapsule du code exécutable.
- **Composites** :Une sorte de composant Fractal qui encapsule un ensemble de composants avec un niveau d'emboîtement quelconque (ainsi une autre propriété qui explique l'appellation Fractal) tel que le partage [Szy98].

Le contenu gère tout ce qui est fonctionnel dans le composant, c'est la partie " métier ".

- Le contrôleur

La partie contrôleur du composant Fractal contrôle un ou plusieurs aspects non fonctionnels (techniques) tel que : l'introspection, la configuration, la sécurité, les transactions...c'est la partie "non-métier". Le contrôleur peut avoir un ensemble d'interfaces internes et d'interfaces externes. Les interfaces externes sont accessibles de l'extérieur du composant, tandis que les interfaces internes sont accessibles uniquement des composants sous-composants. Le contrôleur d'un composant Fractal est essentiellement composé de divers objets *contrôleurs* et *intercepteurs* [BCL+04]. Un objet contrôleur implémente les interfaces de contrôles (plus

de détails dans la section suivante). Tandis qu'un objet intercepteur est utilisé pour exporter l'interface externe d'un sous composant comme une interface externe du composant père (le composite).

Le schéma 2.1 décrit quelques définitions de base de la spécification Fracatl.

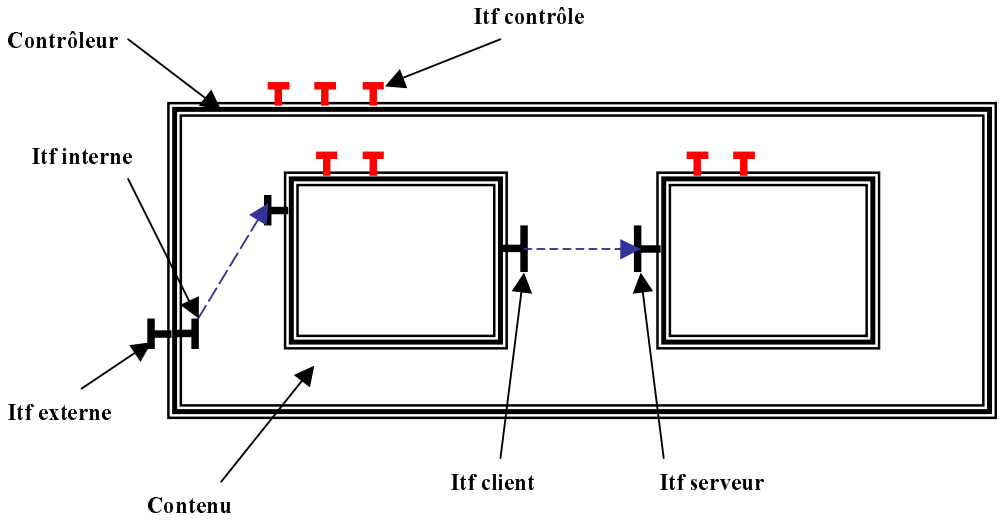

FIG. 2.1 – Schéma descriptif des interfaces

- Les interfaces de contrôle

Parmi les interfaces élémentaires de contrôle qui permettent la configuration d'un composant Fractal, on peut citer :

- **Name Controller** : permet d'associer un nom à chaque composant.
- **Attribute Controller** : un attribut est une propriété configurable du composant. Il est utilisé pour configurer l'état du composant. "Attribute Controller" contient les opérations de lecture et d'écriture pour chaque attribut.
- **Binding Controller** : c'est le gestionnaire de liaison entre composants. Il Permet de connecter ou déconnecter une interface cliente d'un composant avec une autre interface serveur d'un autre composant. C'est à travers le binding que la communication entre les différentes entités du modèle Fractal (les composants) est assurée (à voir dans la prochaine section).
- **Content Controller** : s'occupe de la gestion du contenu du composite. Et donc, liste, rajoute et supprime les sous composants d'un contenu du composite.
- **Life Cycle Controller** : cette interface de contrôle est très importante pour la gestion de vie du composant. Elle contient les méthodes permettant de démarrer ou arrêter l'exécution d'un composant en vu de le supprimer ou simplement le remplacer sans influencer l'exécution de l'application globale.
- **Super Controller** : permet de s'informer sur le composant père (composite) qui contient les différents sous composants.

- La communication et l'assemblage

Une application conçue à base de composants Fractal supporte l'interaction et la composition entre les différents entités, c'est à dire, communication et assemblage des composants. La communication entre composants Fractal n'est possible que si leurs interfaces soient liées. Le rôle principal du "binding" dans le modèle Fractal correspond au concept du "connecteur" défini dans les autres modèles de composants. Dans le modèle Fractal on parle de deux sortes de liaison (binding) :***primitive*** et ***composite***.

- Le **binding primitif**(simple) est une liaison entre une interface cliente et une interface serveur à la même adresse mémoire signifiant que l'opération émise par l'interface cliente doit être acceptée par une interface serveur spécifique (et non pas par n'importe quelle interface serveur).
- Le **binding composite**(composé) représente le chemin de communication entre un nombre arbitraire d'interfaces composant.

On parle également d'un binding normal, import, ou export. Le schéma de la figure 2.2 illustre les sortes de binding.

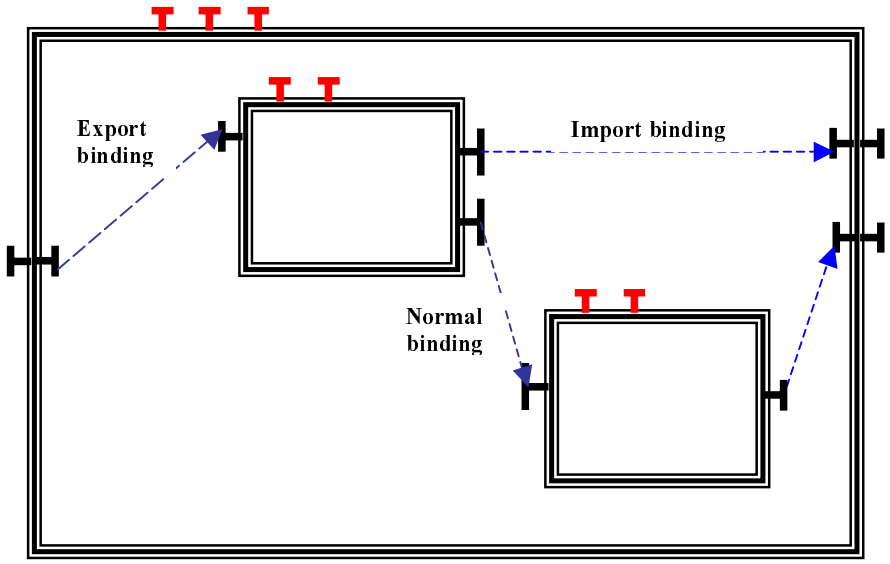

FIG. 2.2 – Composant Fractal avec les différents binding

Une des caractéristiques importantes du modèle Fractal est le fait de pouvoir inclure dans un composant un ensemble d'autres composants : l'assemblage. Ces composants peuvent être partagés ou composés (Figure 2.3).

Les composants partagés sont utiles pour préserver l'encapsulation du composant et permettre l'accès aux ressources du système de bas niveau.

- Le typage

Le modèle Fractal est doté d'un système de type pour les composants Fractal et ses interfaces. Le type de composant Fractal (non modifiable à l'exécution) est défini par l'ensemble

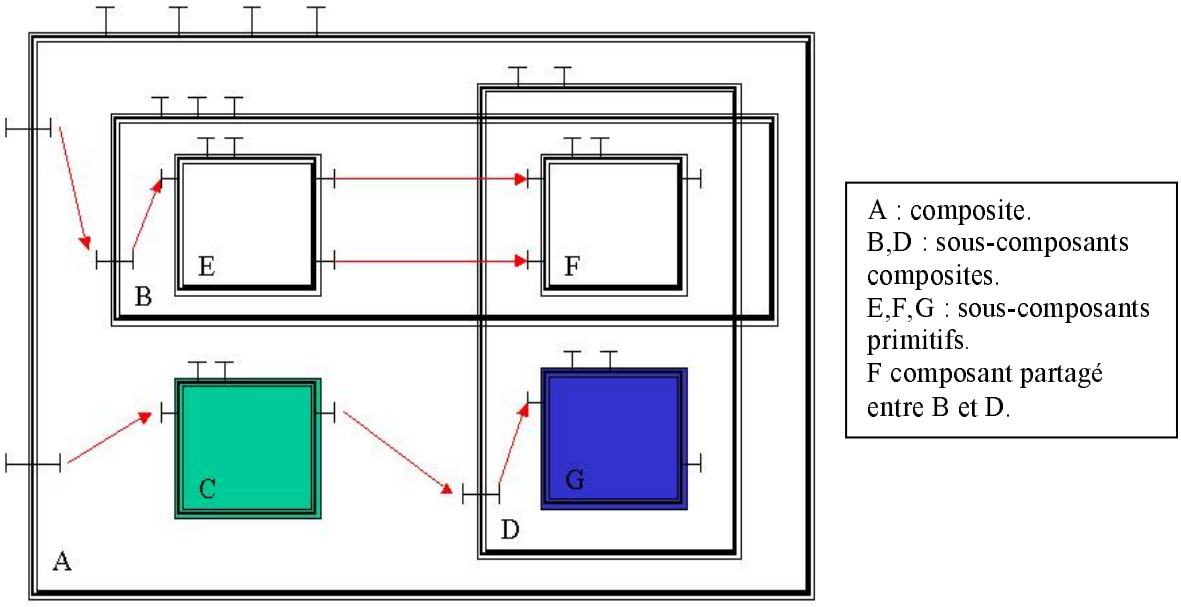

FIG. 2.3 – Composant Fractal avec les différents emboîtements

de types de ses interfaces qui sont :

- Le ***nom*** de l'interface.
- La ***signature*** qui représente le type du langage.
- Le ***rôle*** de l'interface qui est soit client ou serveur.
- La ***contingence*** qui spécifie une garantie sur la disponibilité des fonctionnalités de l'interface : obligatoire " Si interface serveur, il doit être disponible. Si interface client doit être liée à une interface serveur " ou optionnel " pas de garantie, ni d'obligation ".
- La ***cardinalité*** de l'interface selon sa liaison : singleton ou collection.

Tout au long de cette partie du rapport nous avons présenté le modèle Fractal et sa spécification. Dans ce qui suit nous allons présenter quelques expériences d'implantations du modèle Fractal et nous décrirons une des implantations du modèle Fractal considéré comme une implantation de référence, en l'occurrence Julia.

2.3.3 Implantations

Il existe principalement trois implantations du modèle de composants Fractal [BBC+04], développées dans des contextes variés :

1. En centralisé : la plate-forme ***Julia***, première implantation de référence du modèle de composant Fractal en Java développée conjointement par France Telecom R&D et l'INRIA.
2. En réparti pour le calcul à grande échelle : l'mplantation ***ProActive***, qui est une bibliothèque spécifiquement dédiée au développement et à l'exécution d'applications à objets en Java, réparties, parallèles, et mobiles, a été étendu vers un modèle à base de composants Fractal.

3. Pour les noyaux de systèmes d'exploitation : le noyau ***Think*** développé en C, qui est un environnement de systèmes d'exploitation à base de composants, a été enrichi des concepts et principes issus du canevas à composant Fractal.

On constate que *Think* et *proActive* ont été améliorés par l'insertion du modèle de composants Fractal afin d'exploiter ses avantages. Cependant, *Julia* est la seule implantation en Java dédiée complètement à Fractal. Dans les sections suivantes on décrit les objectifs de *Julia* et les principaux structures de données utilisées dans cette implantation.

- Objectifs

Le but principal de Julia est de fournir un canevas pour programmer des contrôleurs de composants, c'est à dire d'implémenter un ensemble d'objets de contrôle extensible, à partir duquel l'utilisateur peut choisir et assembler librement les objets de contrôle qu'il souhaite, afin d'obtenir un contrôleur de composant Fractal [BBC+04].

Un autre but important est de fournir un continuum allant de la configuration statique à la reconfiguration dynamique en fournissant un ensemble d'objets de contrôle suffisamment riche et flexible pour que l'utilisateur puisse faire les compromis mémoire / vitesse qu'il souhaite.

Le troisième but de Julia est d'implanter les objets de contrôle précédents de façon à ce qu'ils aient le moins d'impact possible sur les applications, que ce soit en mémoire ou en temps d'exécution.

Le dernier but de Julia est de fournir un canevas qui soit utilisable sur n'importe quel JDK, y compris ceux qui sont très limités, comme la KVM et le profile J2ME.

- Structures de données

Un composant Fractal est généralement représenté dans Julia par plusieurs objets Java, qui peuvent être séparés en trois groupes :

- Les objets qui référencent les interfaces du composant (il y en a un par interface serveur),
- Les objets de contrôle qui implémentent le contrôleur du composant(un objet de contrôle peut implanter un nombre quelconque, même nul, d'interfaces de contrôle),
- Les objets qui implémentent le contenu du composant.

Le fait que chaque interface Fractal soit représenté par un objet Java séparé vient du fait que les références d'interfaces sont typées.

2.4 Conclusion

Les modèles de composants logiciels constituent une évolution logique, un progrès des modèles à objets dans l'industrie des logiciels. Cette évolution prend en compte la structure d'une application. Ainsi, pour mieux spécifier les aspects qui doivent être satisfait par un modèle, le découpage de l'application en étapes selon les besoins aide au mieux pour choisir le modèle de composant adéquat, et facilite par la suite l'évolution de ce modèle. De plus, la

visibilité des interconnexions entre composants permet de changer la structure de l'application (l'adaptation).

La technologie orientée composant est meilleure par rapport à celle de l'orientée objet pour l'industrie des logiciels, car en plus des propriétés d'encapsulation, de modularité et de spécification d'interfaces fournis, le modèle des composants permet le déploiement, la réutilisation, la reconfiguration dynamique et la spécification des interfaces requises. Les besoins des usagers ne cessent de se compliquer, mais en contre partie, les fournisseurs ont des contraintes à respecter.

A l'heure actuelle, il n'existe pas un modèle universel de composants logiciels. C'est plutôt les besoins qui imposent le modèle. Néanmoins, il faut tenir compte de l'évolution de ces besoins afin d'éviter d'être trop lié au modèle de composants choisi. On parle ainsi de modèle dynamique. Mais la majorité des modèles sont statiques où il y a un manque d'adaptation aux changements brusques. De ce fait, tout changement dans le modèle est coûteux et fait perdre le modèle en lui-même. L'idéal est d'avoir un modèle dynamique (auto-configuration, auto-description...), pas au sens propre du terme mais qui répond de mieux en mieux aux exigences du marché.

Malgré l'engouement autour des modèles à composants industriels comme les EJBs ou CCM, la communauté académique a proposé un nouveau modèle pour faciliter la spécification, la construction, le déploiement ou la reconfiguration d'applications à base de composants logiciels. C'est le modèle de composants Fractal qui est particulièrement apprécié pour sa légèreté, son extensibilité et sa proximité des concepts des langages de programmation.

Julia, l'implantation de Fractal en Java, est plus au moins conforme au modèle Fractal. Ceci est dû au fait qu'elle n'implémente pas toute la richesse du modèle (par exemple, pas de notion explicite de partage de composants). En plus, Julia nous oblige d'être restreint par la machine virtuelle Java.

Après cette étude de l'existant, notre travail se situe dans le même contexte que l'implantation Julia, c'est à dire réaliser une projection du modèle abstrait du modèle Fractal. Cette projection est faite à travers le langage Smalltalk (le premier véritable langage de programmation orienté objets). Le chapitre suivant sera consacré à notre travail de mise en œuvre : de la conception à la réalisation, de l'implantation du modèle Fractal en smalltalk qu'on appellera : FracTalk.

Chapitre 3

Mise en œuvre de Fractal en Smalltalk

3.1 Introduction

Le modèle Fractal tel que décrit dans la spécification du consortium Objectweb n'est qu'un modèle abstrait. De plus, à travers l'étude de l'existant des travaux concernant la mise en œuvre du modèle Fractal, nous avons constaté l'absence d'une stratégie qui décrit la manière dont les différents éléments de la spécification Fractal sont liées entre eux.
La projection du modèle Fractal vers les éléments du langage de programmation smalltalk nous a permis d'avoir un modèle concret du Fractal, qu'on appellera : FracTalk.
L'objectif de cette partie du rapport est la description de la mise en œuvre de FracTalk : de la conception à la réalisation.

Ce chapitre est divisé en deux parties, la première est consacrée à la présentation du modèle conceptuel de Fractal, tandis que la seconde partie fera l'objet de la description de l'implantation Fractalk.

3.2 Etude du modèle conceptuel de Fractal

La phase de conception constitue une étape importante dans la réalisation de l'implantation FracTalk. Afin de bien distinguer les différentes parties clés du modèle Fractal et mettre en valeur ses fonctionnalités pour en final avoir une projection claire, modulaire et extensible, nous avons traduit la spécification Fractal en UML (*Unified Modeling Language*). Cette traduction, qui est la représentation "UML" du modèle Fractal, constitue une première dans le domaine de composants Fractal, et représente ainsi une contribution de notre part à la communauté de recherche en composant Fractal.

3.2.1 Pourquoi UML ?

UML est un langage semi-formel qui permet d'exprimer et d'élaborer des modèles objet, indépendamment de tout langage de programmation [Pen02]. En d'autres termes, UML est un outil qui donne une dimension méthodologique à l'approche objet et qui permette de mieux maîtriser sa richesse.
Le choix d'UML comme support de conception est motivé par le fait qu'il offre une plate forme

de communication performante qui facilite la représentation et la compréhension de solutions objet grâce à :

- Sa notation graphique qui permet d'exprimer visuellement une solution objet, ce qui facilite la comparaison et l'évaluation de solutions ;
- L'aspect formel de sa notation qui limite les ambiguïtés et les incompréhensions ;
- Son indépendance par rapport aux langages de programmation, aux domaines d'application et aux processus qui en font un langage universel.

3.2.2 Modèle UML de Fractal

Pour réaliser le modèle UML de Fractal par construction du diagramme de classes avec les classes et leurs associations (agrégation, composition et héritage), nous avons procéder comme suit :

1. Représentation de la brique de base de la spécification Fractal qui est le composant Fractal en définissant : son type, ses sortes (composite ou primitif), ses attributs, ainsi que ses interfaces (voir Figure 3.1).

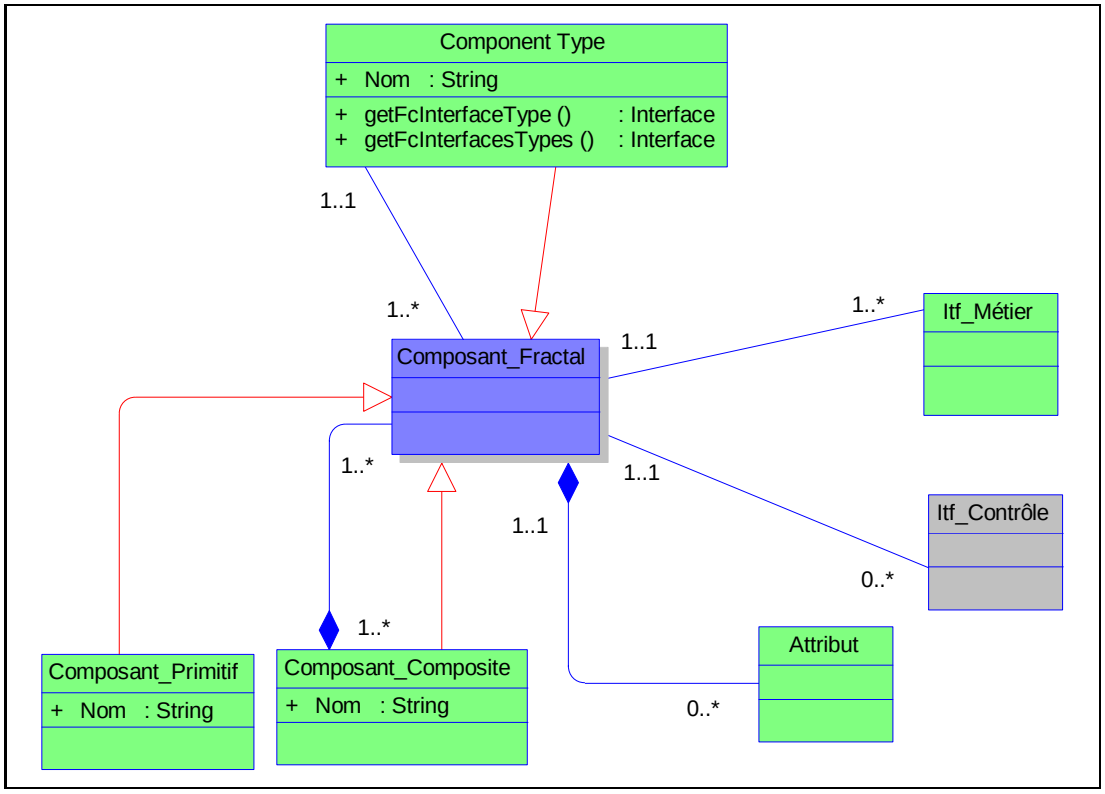

FIG. 3.1 – Représentation UML du composant Fractal

Un composant Fractal qui peut être soit primitif ou composite (composé d'au moins un composant primitif) a un et un seul type défini par le type de chacun de ses interfaces métier ou contrôle. Un composant Fractal peut avoir un ou plusieurs interfaces métier de rôle serveur ou client, et plusieurs interfaces de contrôle ou ne pas avoir. Dans ce

dernier cas, le composant est appelé un *composant de base*. Chaque composant Fractal peut avoir des attributs. Un attribut est une propriété configurable du composant. Il sert à configurer l'état du composant sans faire appel au binding. Ces attributs sont gérés par la classe Attribute Controller (cf.2.3.2). Le diagramme de classes UML illustre parfaitement les définitions qu'on vient de mentionner.

2. A partir de chaque élément de la définition du composant Fractal, on représente ses caractéristiques. Cette représentation concerne principalement : les interfaces (voir Figure 3.2), le type (voir Figure 3.3) et le binding (voir Figure 3.4).

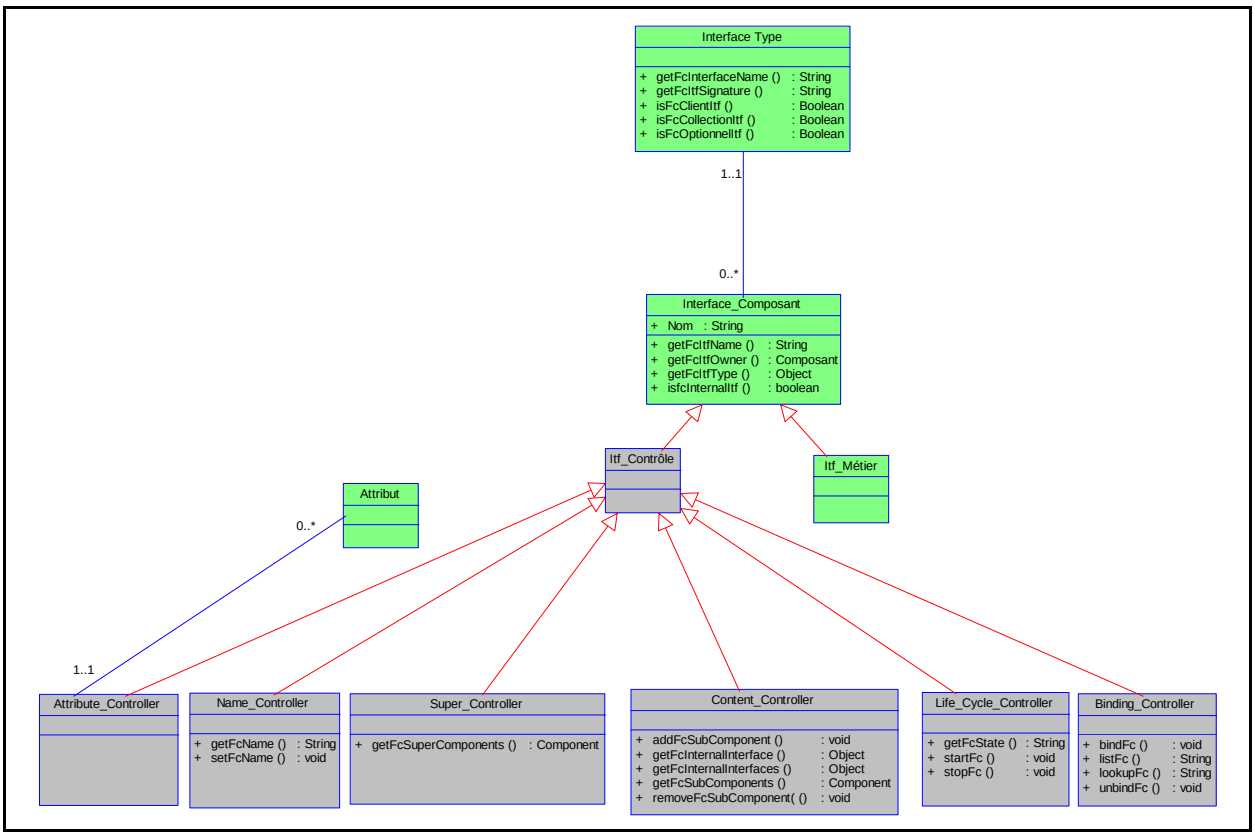

FIG. 3.2 – Représentation UML des interfaces du composant Fractal

Fractal définit un ensemble d'interfaces de contrôle offrant des primitives de base (voir méthodes de classes sur la figure 3.2) pour configurer et reconfigurer dynamiquement les composants. Ceci concerne essentiellement le contrôle des attributs, des lisons, et des sous composants d'un composant, ainsi que le contrôle de son cycle de vie. Une reconfiguration d'un composant ayant lieu en trois étapes : suspension de l'activité du composant (en utilisant son contrôleur de cycle de vie), reconfiguration proprement dite (c'est à dire ajout, retrait ou modification de liaisons, de sous composants, ou d'attributs), et enfin reprise de l'activité du composant.

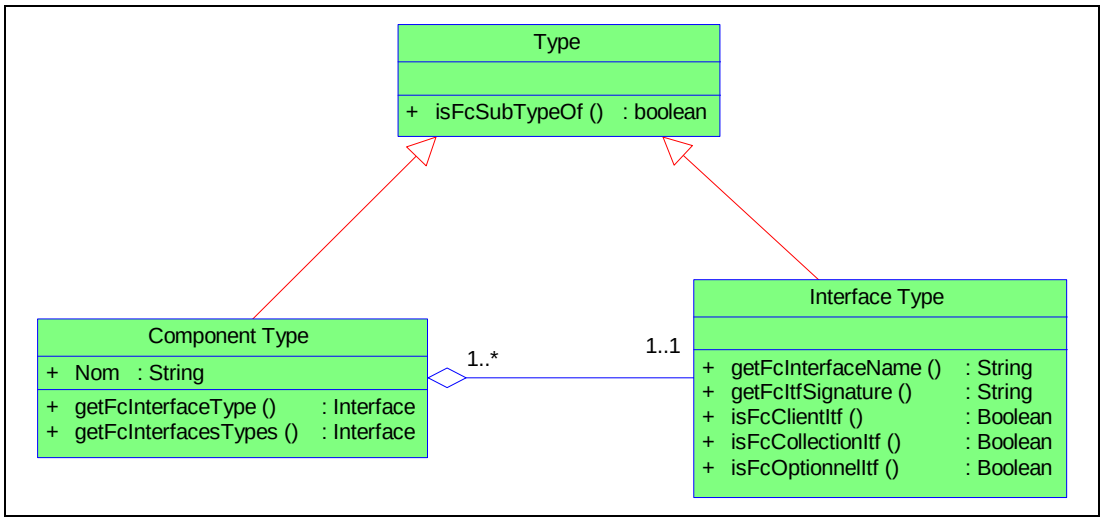

FIG. 3.3 – Représentation UML du système de typage

La figure 3.3 définie le système de typage pour les composants et leurs interfaces. Ce système reflète les caractéristiques principales de l'interface d'un composant (nom, type du langage, rôle, contingence et cardinalité) qu'on a introduit au chapitre 2 (cf.2.3.2). Le typage nous permet de garder une cohérence et une compatibilité entre composants. Par exemple lors du binding, les types des interfaces à liées sont vérifiés pour assurer la compatibilité.

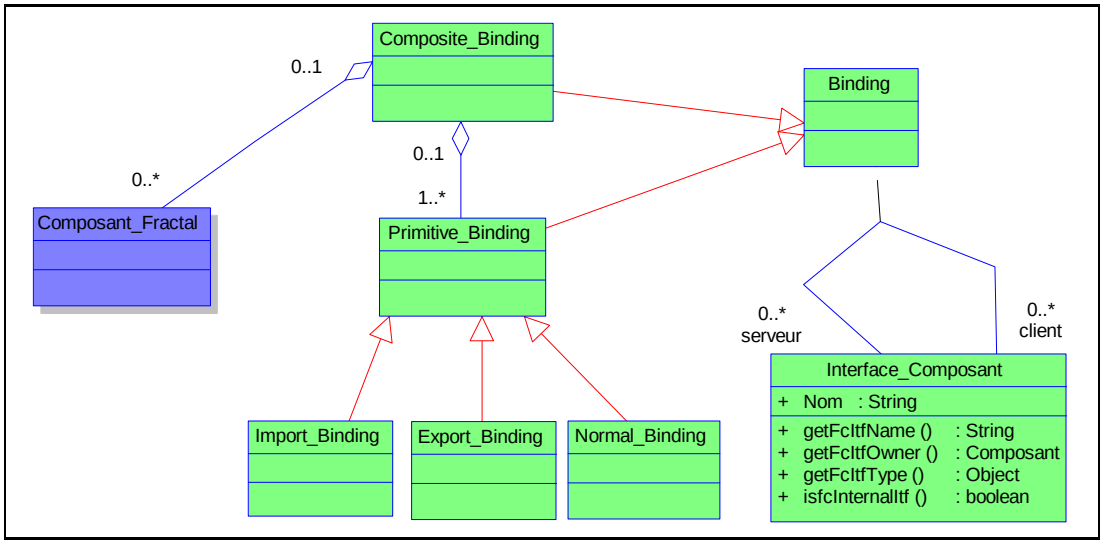

FIG. 3.4 – Représentation UML du Binding

La notion du binding est une notion plus qu'importante dans la réalisation d'application (qui peut être vu comme étant un seul composant fractal) à base de composants. Un composant Fractal est constitué d'au plus un binding composite composé de un ou plusieurs binding primitives (de sorte import, export, ou normal). Un Binding est une connexion, une liaison entre une interface de rôle client et une interface de rôle serveur. Notons que

la cardinalité 0..* figurant sur le schéma UML (Figure 3.4) représentant l'association du binding entre interfaces clients et serveurs reprend parfaitement la sémantique de *cardinalité* (singleton ou collection) et de *contingence* qui sont des caractéristiques de type d'interface.

3. Représenter la partie implantation (voir Figure 3.5).

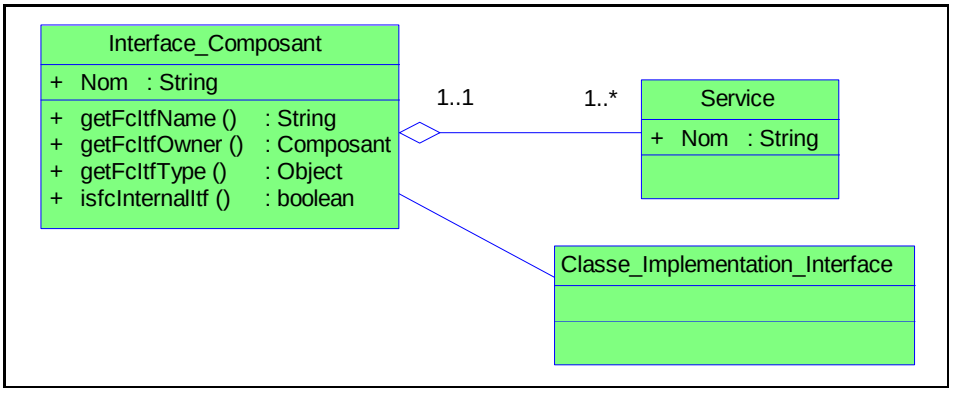

FIG. 3.5 – Représentation UML de la partie implentation

Un service correspond à une interface métier du composant. Ce service est implémenté par une et une seule classe d'implantation.

4. Finalement, il est nécessaire de représenter la partie "fabrique" de composants (voir Figure 3.6).

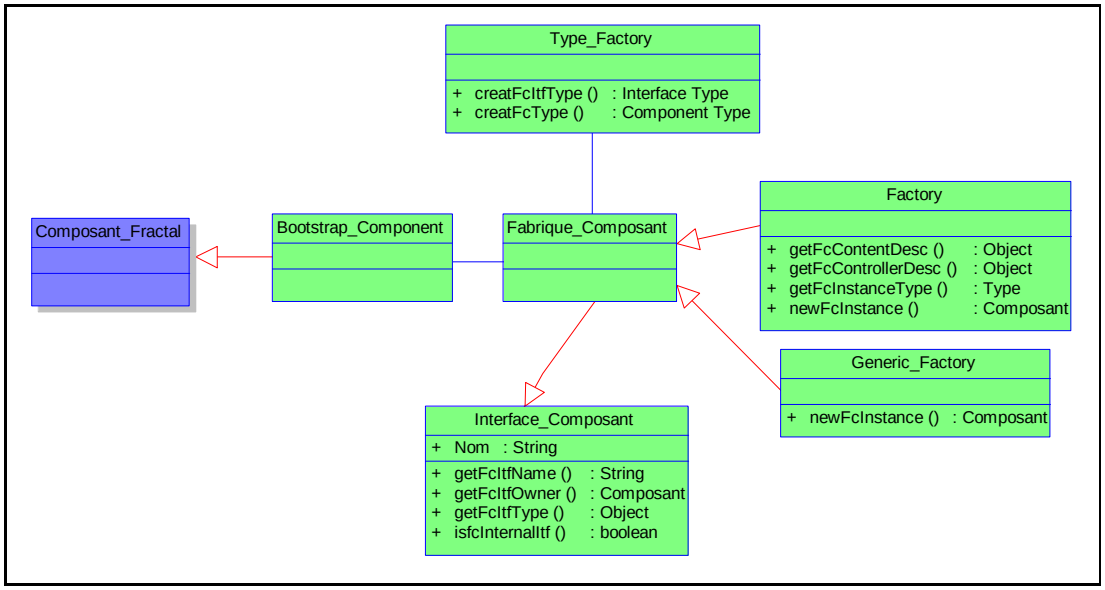

FIG. 3.6 – Représentation UML de la partie Fabrique de composants

La fabrique de composants est une entité qui gère les instances d'un composant (création, description du contenu, description du contrôleur). Cette fabrique implémente des

"constructeurs de composants" et des "constructeurs de type".

3.3 Projection du modèle conceptuel en Smalltalk

A partir du modèle UML réalisé, la projection de Fractal vers les éléments du langage de programmation Smalltalk s'avère plus structurée et plus directe. Rappelons qu'un des objectifs de notre implantation FracTalk est d'exploiter la richesse du langage Smalltalk en matière de programmation objet, et les facilités offertes en développement d'applications. Ceci explique principalement notre choix. Dans ce qui suit nous argumentons le pourquoi de ce choix.

3.3.1 Pourquoi le langage Smalltalk ?

Smalltalk est un des premiers langages de programmation orienté objet. Il a été créé en 1972. Il est inspiré par Lisp et Simula. Il a été conçu par Alan Kay,Dan Ingalls, Ted Kaehler, Adele Goldberg au Palo Alto Research Center de Xerox. Le langage a été formalisé en tant que Smalltalk-80 et est depuis utilisé par un grand nombre de personnes. Smalltalk est toujours activement développé et dispose d'une base loyale d'utilisateurs. Smalltalk a été d'une grande influence dans le developpement de nombreux langages de programmation, dont : Objective-C, Actor, Java et Ruby.
Les principaux concepts de Smalltalk sont [1] :

- "Tout est objet." Les chaînes de caractères, les entiers, les booléens, les définitions de classes, les blocs de code, les piles et la mémoire sont représentés en tant qu'objets.
- Tout est modifiable. Si vous voulez changer l'IDE[2], vous pouvez le faire en cours d'utilisation, sans recompiler et redémarrer l'application. Si vous voulez une nouvelle instruction de contrôle dans le langage, vous pouvez l'ajouter. Avec certaines implémentations, vous pouvez même changer la syntaxe du langage, ou la façon dont le ramasse-miette fonctionne.
- Les Types sont dynamiques, pas besoin de définir des types dans votre code, ce qui permet au langage d'être concis.
- Un ramasse-miettes mémoire est intégré et transparent pour le développeur.
- Les programmes Smalltalk sont généralement compilés en bytecode, exécutés par une machine virtuelle.
- La Translation Dynamique : les machines virtuelles commerciales modernes compilent le bytecode vers le code machine natif de façon à obtenir de meilleurs performances, une technique dont Smalltalk-80 a été le pionnier, developpé par ParcPlace Systems au milieu des années 80. Cette idée a été adoptée par le langage de programmation Java quelques dix ans après et renommé "compilation Just-in-time", ou JIT.

Il implémente, en plus des principes objets de bases (classe, objet, héritage, polymorphisme), des concepts originaux (métaclasse) et intodruit la notion d'objet persistant, de traitement des exceptions et le principe Modèle-Vue-Controlleur.

[1] http ://fr.wikipedia.org/wiki/Smalltalk
[2] Integrated Development Environment

Une caractéristique surprenante de Smalltalk est l'absence totale d'instructions de contrôles intégrées au langage : if-then-else, for, while, etc. Toutes ces instructions sont implémentés en utilisant des objets. Par exemple, les décisions sont prises en envoyant un message ifTrue à un objet Booléen, et en passant un fragment de code à exécuter si le Booléen est vrai. La seule chose d'intégré par défaut est la syntaxe pour envoyer un message à un objet.

Notre réalisation est développé en Squeak[3] qui est un Smalltalk libre (open source). Construit sur un langage objet homogène et très puissant (en l'occurrence Smalltalk), Squeak offre un environnement de développement efficace et très bien adapté au développement orienté objet, une très importante bibliothèque de classes, et permet une productivité et une souplesse de développement difficiles à égaler [BD01]. Totalement gratuit et open source, supporté par une importante communauté de développeurs dans le monde, facile à maintenir et à faire évoluer, Squeak a des avantages qui en font un concurrent de poids dans la compétition des langages et des environnements de développement pour divers applications (Internet, multimédia, gestion,...) pour lesquelles la modélisation objet est tout indiquée. Produit jeune, puisque son développement a commencé en 1996, Squeak bénéficie pourtant de plus de trente ans d'expérience en développment orienté-objet, et tire largement profit des travaux effectués dans les communautés Smalltalk, Self et Java.

3.3.2 Composant FracTalk

Un composant FracTalk est un composant logiciel Fractal implémenté en smalltalk comme un objet.Il est représenté par plusieurs autres objets divisés en trois groupes (voir figure 3.7) :

- les objets implémentant les interfaces de contrôles fournis par la plate forme Fractal (un objet par interface).
- les objets implémentant les interfaces métiers crées automatiquement.
- les objets implémentant la partie contenue du composant (le code fonctionnel dans le cas d'un composant primitifs ou l'ensemble des sous-composants dans le cas d'un composant composite).

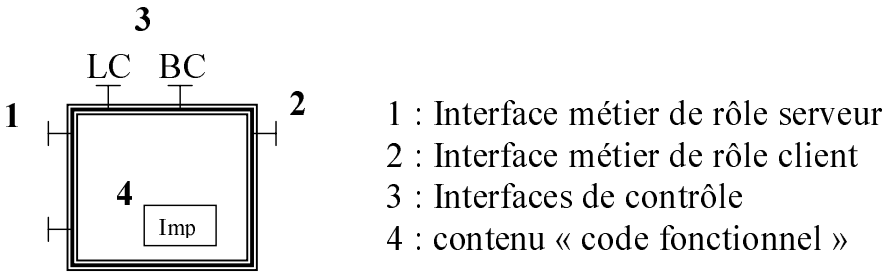

FIG. 3.7 – Exemple d'un composant FracTalk Primitif

Le fait que chaque interface Fractal soit représenté par un objet Smalltalk vient du fait d'une nécessité de respecter la spécification Fractal en ce qui concerne la séparation entre la partie fonctionnel et la partie non-fonctionnel. Le Typage dynamique offert par Smalltalk nous a permis de faire mieux que Julia.

[3]http ://www.squeak.org

L'implantation des composants Fractal est prise en charge par le programmeur de composants, ainsi que par le système de composants FracTalk. Le programmeur fournit le code fonctionnel, alors que le système se charge de la génération du code non fonctionnel. La structure de composant fait coopérer ses deux parties de code.

3.3.3 Classes implantées

L'implantation en Smalltalk des différents concepts du modèle Fractal représentés par les digrammes UML est faite à travers des catégories de classes. Dans cette section, nous ne redéfinirons pas ces concepts, nous exposerons l'ensemble des classes qui ont été créées et nous décrirons les propriétés qui les caractérisent.

Nous commencerons l'énumération de ces classes par la catégorie **FracTalk** qui regroupe trois classes : **FcComponent**, **FcInterface** et **Fractal**. Cette catégorie caractérise les deux concepts de base : le composant et ses interfaces. A chaque classe correspond un ensemble de méthodes. Le code d'une de ces méthodes écrit en Squeak est le suivant :

```
getFcInterface: name
"retourne l'objet interface qui a comme nom name"
^self getFcInterfaces
        detect: [:uneInterface | uneInterface getFcItfName = name]
        ifNone: [^ nil].
```

Cette méthode nous permet d'avoir l'objet interface qui a comme nom le paramètre "name".

FracTalk-Type symbolise le système de typage. Cette catégorie est constituée des classes suivantes : **Type**, **ComponentType** et **InterfaceType**. La classe **Type** est la super classe des deux autres. Une des méthodes clés qui correspond à la classe **InterfaceType** est la méthode *creatInterface :Class :* dont le code est le suivant :

```
createInterface: nom Class:signature
|classeInterface code|
classeInterface :=  FcInterface
        subclass:('Itf', nom) asSymbol
        instanceVariableNames: 'contentIntf'
        classVariableNames: ''
        poolDictionaries: ''
        category: 'FracTalk-lab'.
code := self methodContCreat.
classeInterface compile: code.
signature getSignatures
                do: [:elt |
                        code := self methodCreat: elt.
                        classeInterface compile: code].
^classeInterface.
```

Cette méthode de classe nous permet de créer automatiquement le canevas de la classe d'implantation de l'interface correspondante.

La catégorie **Fractalk-Factory** caractérise la partie fabrique de composants. Elle est constituée d'un ensemble de classes dont la classe **GenericFacory** qui permet de créer la partie *contenu* et *contrôleur* du composant selon qu'il soit *primitif* ou *composite*. Ci-dessous le code de la méthode *newFcInstance*[4] :

```
newFcInstance: type1 and: controller1 and: content1
| component1 |
component1 _ FcComponent new.
component1 setComponentType: type1.
component1 _ self create: controller1 controller: component1.
((controller1='primitive') and:[content1~~nil])
        ifTrue:[component1 _ self create: content1 content: component1].
component1 _ self create: type1 interface: component1.
type1 getFcInterfaceTypes.
component1 setBindingController:type1.
^ component1
```

Pour la partie implantation, elle est représenté par la catégorie de classes **ImplementationClasses** qui comporte une seule classe **InterfaceDescription**.

Enfin, la dernière catégorie de classes que nous décrirons est la catégorie **FracTalk-Control**. Elle est constitué des six classes de contrôle : **AttributeController**, **Binding-Controller**, **ContentController**, **LifeCycleController**, **NameController**,et **SuperController**.

Le schéma 3.8 représente la hiérarchie des classes implémentées ainsi que leurs catégories pour la réalisation de notre plate-forme du modèle Fractal.

3.3.4 Exécution

Le processus de création de composant Fractal à partir de FracTalk passe essentiellement par un certain nombre d'étapes qui s'effectue dans cet ordre :

1. On commence par récupérer une instance de la fabrique de type : *TypeFactory*.
2. On construit ensuite le type du composant par la définition des types de ses interfaces à partir de cette instance de type.
3. On définit une instance du composant associé au type défini précédemment en spécifiant la description du contenu et du contrôleur du composant à créer selon qu'il soit primitif ou composite. Et c'est à ce niveau qu'on définie (par la descripteur du contrôleur) les interfaces de contrôles et les classes d'implantation (par le descripteur du contenu).
4. On procède à l'assemblage, s'il y a lieu, des sous composants précedement construit en faisant appel à une instance de la classe *ContentController* du composant composite qui permet l'accès à ses méthodes.

[4]Dû à la limitation de nombre de pages, nous n'avons présenté que le code de quelques méthodes

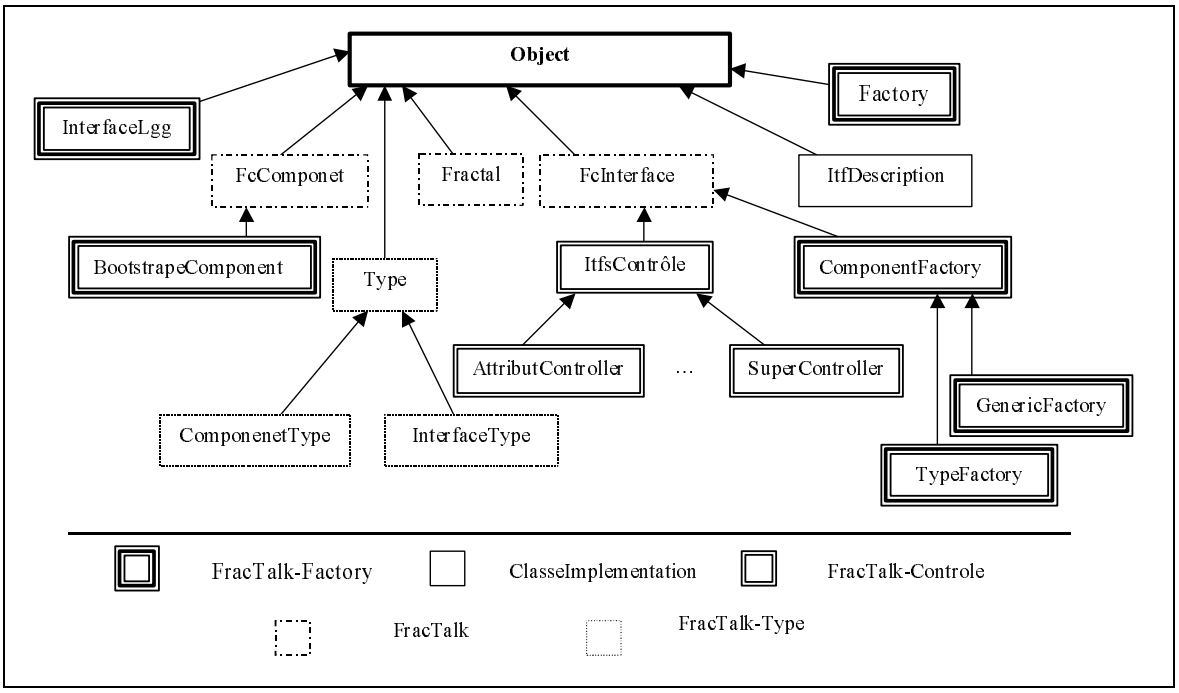

FIG. 3.8 – Hiérarchie des classes implémentées

5. On établit les différentes liaisons (binding) entre les interfaces métiers des composants concernés.
6. On démarre le composant. L'application est prête à être exécutée.

Description du contenu et du contrôleur

La méthode ***newFcInstance : type and : controller and : content*** de la classe *GenericFactory* permet la configuration du composant. Cette configuration est réalisée à travers un script de code écrit en Smalltalk et intégré dans l'implantation FracTalk. Le choix de cette solution d'intégration est motivé pour autoriser d'autres cas de figures de configuration et permettre ainsi une configuration dynamique qui rendra notre plate forme extensible et flexible. Ceci est contrairement à la configuration de la plate forme Julia qui est basé sur la manipulation d'un fichier de configuration (Julia.cfg) utilisant un format (à la LISP) peu évident à manipuler ; et surtout qui n'autorise qu'une configuration statique.

La déclaration de l'assembllage : le binding

Le binding est la liaison entre les interfaces. Il est géré par l'interface de contrôle *BindingControler* représenté par la classe *BindingController* de la catégorie de classe *ItfControle* (voir hiérarchie de classes dans le chapitre précédent). Cette interface est crée automatiquement lors de l'instantiation (création) du composant (selon qu'il est primitif ou composite). Plus exactement, à la description de contrôleur qui contiendra la liste des classes (*ItfContrôle*) implantant les contrôleurs utiles selon chaque sorte du composant (primitif ou composite).

Le binding est exécuté d'une manière automatique, sans l'intervention du concepteur du composant, lors de l'envoie du message ***bindFc : itf1 du : composant to : itf2*** à l'instance de la classe BindingControlleur du composant (composant) qui contient l'interface cliente (itf1) qui doit être liée à l'interface serveur (itf2). Ce lien signifie faire communiquer les deux interfaces, en terme de programmation objets, elles doivent avoir la même référence.

Julia donne la main au concepteur de composant Fractal d'effectuer le binding. Ceci peut générer des erreurs. L'automatisation de cette opération sécurise au mieux la conception d'application à base de composants Fractal, et préserve la sémantique de l'opération du binding qui est une opération de contrôle non-fonctionnelle (non-métier) offerte par la plate forme.

3.3.5 Diagramme d'interraction

A travers la description du diagramme représenté sur la figure 3.9, nous voulons montrer l'interaction entre les différents objets d'un composant FracTalk. Sur le diagramme figure deux

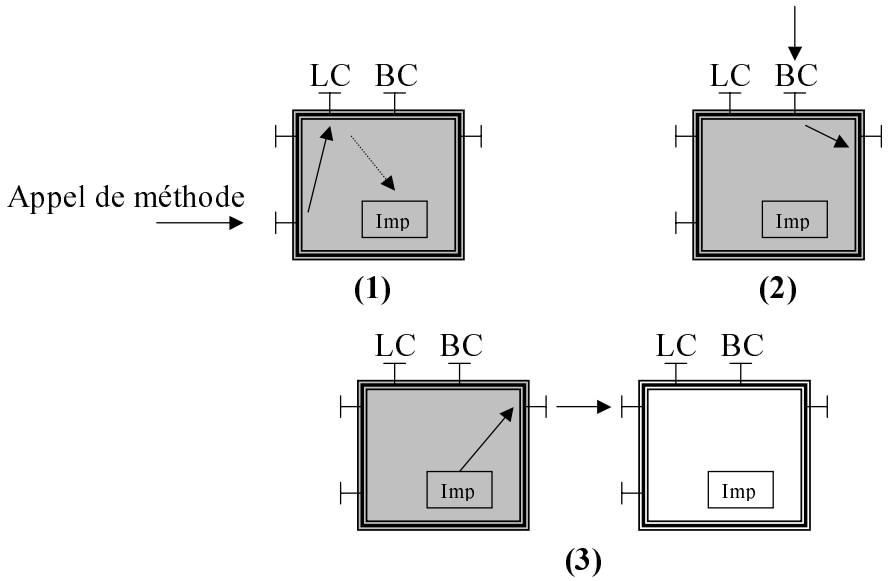

FIG. 3.9 – Diagramme d'interaction entre différents objets

composants Fractal. Chacun est composé de deux interfaces de contrôle : LC (gestion de cycle de vie) et BC (gestion de liaisons). Et de trois interfaces métiers (deux de rôle serveur et un de rôle client). L'interaction entre les différents objets du composant en général prend le scénario suivant :

- Démarrage de l'application par appel de la méthode qui désigne le service à récupérer.
- Vérification que le composant qui as reçu l'appel est démarré via l'interface de contrôle LC.
- Consulter le code fonctionnel (la classe d'implantation) encapsulé dans le contenu du composant primitif qui a reçu l'appel.
- le code fonctionnel désigne la nécessité de récupérer un service de l'autre composant ainsi il doit y avoir un binding entre l'interface client du premier composant et l'interface serveur du second composant.

- Vérification du binding via l'interface de contrôle BC.
- récupération du résultat et fin de l'application.

3.4 Conclusion

Dans ce chapitre nous avons présenté la mise en oeuvre de FracTalk. Nous avons montré et expliqué les étapes de notre conception jusqu'à la réalisation.

Parmi les avantages de FracTalk comme implantation du modéle Fractal par rapport à l'implantation de référence Julia, nous pouvons citer :

- Le respect de la spécification Fractal grâce à la fléxibilité de Smalltalk contrairement à l'implantation Julia qui été plustôt liée au langage Java.
- Grâce aux typage dynamique des objets sous smalltalk, Fractalk est mieux conçu que Julia. En effet, le programmeur n'a pas besoin de définir de types dans Smalltalk. Ce qui nous amène à un développement rapide de composant sous FracTalk.
- La génération du code non fonctionnel pour la description du contenu et du contrôleur d'un composant FracTalk permet une configuration dynamique grâce à l'intégration de ce code au corps même de l'implantation FracTalk contrairement à Julia.
- Automatisation des opérations de contrôle telle que le *binding* qui sont entièrement transparentes au développeurs de composants Fractal sous FracTalk.

Chapitre 4

Conclusion Générale

Dans ce rapport, nous avons étudié d'une part, la spécification du modèle de composants logiciels Fractal, et d'autre part, nous avons proposé une implantation de ce modèle en Smalltalk.

L'objectif visé par le présent travail était basé sur la proposition d'une plate forme, extensible et flexible, du modèle Fractal, tout en exploitant la richesse du langage de programmation Smalltalk pour mettre en œuvre tout les avantages de Fractal : le partage, la récursion, et la réflexivité.

Dans cet objectif, nous avons d'abord décrit le contexte de l'étude, celui des modèles de composants logiciels. Ensuite, comme nous avons voulu implanter le modèle Fractal en Smalltalk, les caractéristiques de ce modèle ont été présentés.

Nous pensons que le langage Smalltalk avec tout ses richesses en objets a contribué efficacement à la réalisation de l'implantation du modèle Fractal : FracTalk.

Dans cette perspective, nous avons procédé par une conception en UML du modèle Fractal qui constitue une première dans le domaine. Cette représentation nous a beaucoup aidé pour réussir la projection du modèle Fractal vers les éléments du langage Smalltalk, et obtenir enfin notre noyau FracTalk : la première implantation de Fractal en Smalltalk.

Le travail que nous avons présenté dans ce rapport nécessite d'être complété, enrichi, et étendu. Ainsi, en guise de perspectives nous suggérons de :

- Compléter la plate forme FracTalk par l'implantation de la fonction de sous-typage (*sub-type*) qui est nécessaire lors du remplacement des composants Fractal.
- Tester et évaluer les performances de l'implantation FracTalk par rapport à l'optimisation du code et de l'espace mémoire.
- Définir un ADL (*Architecture Description Languages*) pour le développement d'applications à base de FracTalk.
- Proposer une nouvelle architecture orientée service.

Ce dernier point constitue un axe de recherche qu'on as entamé mais par manque de temps on a pas pu l'achevé. En effet, un composant logiciel a besoin pour son fonctionnement d'un ensemble de services. Ces services sont offerts par d'autres composants logiciels. Ainsi, un service est considéré comme une entité associé à un ou plusieurs composants offrants ce service.

Lors de l'assemblage, un composant s'associe à une entité qui représente le service requis. Parmi les problèmes, qui constitue un souci majeur à prendre en compte, est la gestion des états des composants. Cependant, l'arrêt brusque d'un composant risque de générer un arrêt dangereux de toute l'application dont appartient ce composant du moment où le service fourni par le composant qui est tombé en panne n'est plus disponible. De ce fait, et pour éviter ce genre de scénarios et assurer un bon déroulement de l'application, nous suggérons de concevoir une nouvelle architecture orientée service, dont le principe est de regrouper des composants offrants le même service, dans un seul composite, qu'on appellera *composite service*. L'objectif de cet assemblage est d'assurer la continuité d'exécution quelque soit l'état des composants. Un *composite service* dans une applications de composants jouera ainsi le même rôle qu'un switcher dans les circuits électroniques.

Bibliographie

[BBC+04] F. Baude, E. Bruneton, T. Coupaye, P.C. David, T. Ledoux, M. Morel, A. Ocello, and M. Riveill. Projet rntl arcad - apports et limites de l'architecture. Rapport technique, Juin 2004.

[BCL+04] E. Bruneton, T. Coupaye, M. Leclercq, V. Quema, and J.B. Stefani. An open component model and its support in java. In *Proceedings of the International Symposium on Component-based Software Engineering*, May 2004.

[BCS04] E. Bruneton, T. Coupaye, and J.B. Stefani. The fractal component model : specification. Technical report, France Telecom R&D-INRIA, February 2004.

[BD01] X. Briffaut and S. Ducasse. *Squeak programmation.* ISBN :2-212-110-23-5. Eyrolles edition, 2001.

[Chi98] Y. Chikhi. *Réutilisation de structures de données dans le domaine des réseaux électroniques.* PhD thesis, Université Pierre et Marie Curie (Paris VI), Juillet 1998.

[Don01] D. Donsez. Transactions et composants d'entreprise. Cours imag/lsr/adele, Université Joseph Fourier,Grenoble1, 2001.

[GS03a] A. Gacemi and D. Seriai. La réutilisation : Concepts et techniques. Rapport technique, Dépt.GIP, Ecole des Mines de Douai, Avril 2003.

[GS03b] A. Gacemi and D. Seriai. Les composants logiciels : un état de l'art. Rapport technique, Dépt.GIP, Ecole des Mines de Douai, Avril 2003.

[Kra03] S. Krakowiak. Composants logiciels : introduction aux composants. Projet sardes inria et imag-lsr, Université Joseph Fourier, Grenoble, 2003.

[MP02] R. Marvie and M.C. Pellegrini. Modèles de composants, un état de l'art. *RSTI - L'objet*, pages 61–89, August 2002.

[Pen02] T. Penders. *Introduction à UML.* ISBN : 2-7464-0442-7. Oem edition, 2002.

[PMB00] F. Peschanski, T. Meurisse, and J.P. Briot. Les composants logiciels : evolution technologique ou nouveau paradigme ? 2000.

[Rei01] M. Reiveil. Recherches et développements autour des middlewares. LMO'2001.

[SP96] C. Szyperski and C. Pfister. International workshop on component-oriented programming (wcop'96). Workshop report, University of Linz, Austria, July 1996.

[Szy98] C. Szyperski. *Component Software.* ISBN : 0-201-17888-5. Addison-wesley edition, 1998.