UNIVERSITE NICE – SOPHIA ANTIPOLIS
EQUIPE OASIS
MASTER I

TRAVAIL D'ETUDE ET DE RECHERCHE

INTEGRATION DE COMPOSANTS SCA AU SEIN D'UN PROJET SUR BASE GCM GALAXY

ANTHONY JACQUEMIN
LIU ZHAOLONG

ENCADRANTS
FRANÇOISE BAUDE
BASTIEN SAUVAN

Table des matières

CONTEXTE

DES MODELES DE PROGRAMMATION DISTRIBUEE AU CONCEPT DE COMPOSANTS

Ce stage a pour objectif de renforcer l'intégration des composants GCM dans la plateforme Galaxy. A cet effet, le travail consiste à implémenter pour les composants GCM tous les contrôleurs, si possible, spécifiques à SCA, permettant ainsi de donner une personnalité SCA aux composants GCM et donc de considérer un assemblage hétérogène constitué aussi bien de composants SCA que GCM/SCA.

Galaxy ayant également pour objectif l'utilisation de ressources issues de cloud, la deuxième étape consisterait à valider les développements réalisés auparavant avec le déploiement d'une application ProActive/GCM sur cloud.

PROGRAMMATION DISTRIBUEE

INTRODUCTION

Historiquement, la programmation distribuée s'est développée depuis la présence des réseaux d'ordinateurs. Avec l'arrivée des processeurs multi-cœurs, elle est devenue de plus en plus couplée avec la programmation parallèle. L'idée de la programmation distribuée consiste à séparer une grosse tâche en plusieurs parties afin de les distribuer chacune sur une machine distante différente, de sorte que chaque unité soit indépendante. La communication entre chaque partie du programme passe par un réseau d'ordinateurs. L'infrastructure peut efficacement utiliser toutes les ressources existantes pour faire du calcul intensif. Les champs d'application sont très vastes, allant des prévisions météorologiques aux calculs sur l'ADN. Traditionnellement, les langages de programmation distribuée sont C et C++, avec des librairies comme MPI, qui sont d'assez bas niveau et demandent une attention particulière sur l'allocation de mémoire et sur la communication entière à travers l'envoi de messages. Le codage de tels programmes devient rapidement difficile et fastidieux et amène très facilement à une énorme quantité de lignes de code. De plus, le débogage reste problématique, tout comme la compatibilité entre différentes architectures. Ce type de programmation dépend également souvent d'un support NFS. Une plateforme plus fiable et générale était nécessaire, ceci étant d'autant plus accentué par l'apparition du cloud computing.

PROACTIVE

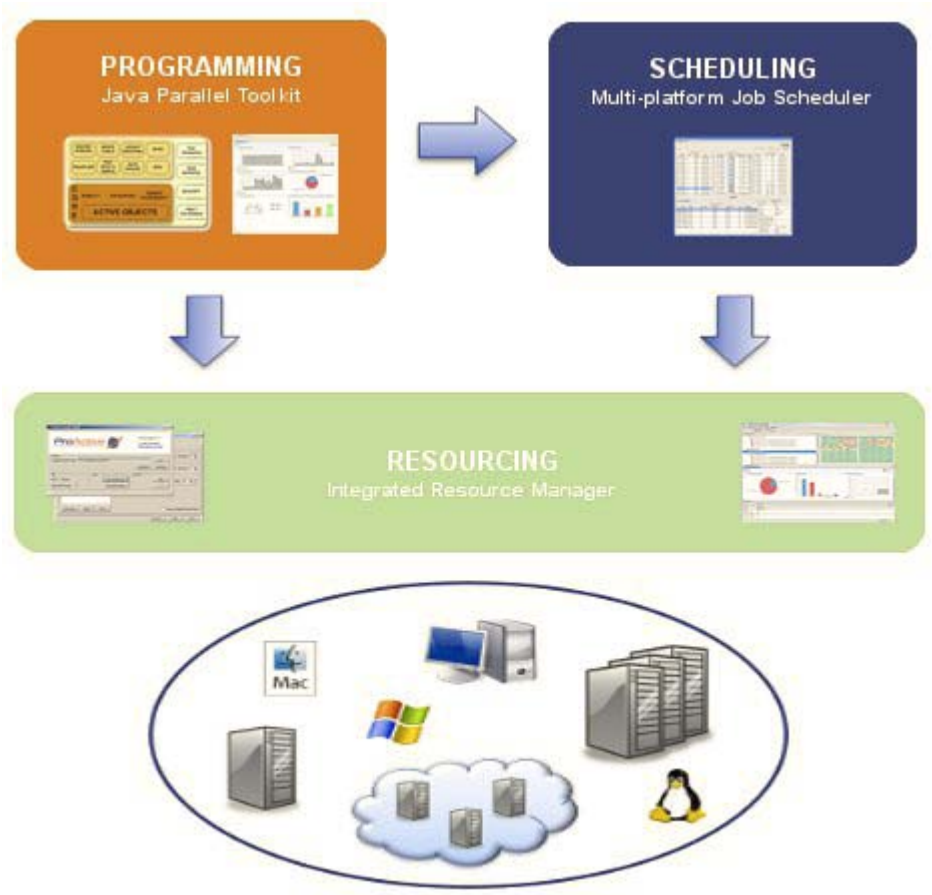
ProActive implémente une plateforme Java pour la programmation distribuée. Profitant de l'approche orientée objets de Java, il apporte une facilité de codage, des points tels que l'envoi de message et l'allocation de mémoire étant cachés au programmeur. De même, grâce à la JVM, il évite des problèmes de dépendance entre architectures, facilitant alors l'exploitation vers des clouds avec des API Java présentes.

ProActive promeut quelques principes fondamentaux :

- Les objets actifs sont des objets mono-threadés distribués et accessibles à distance
- Les interactions s'effectuent par des appels de méthodes asynchrones
- Les résultats de ces interactions sont appelés futurs et sont des entités de premier ordre
- Les demandeurs peuvent attendre les résultats si nécessaire en utilisant un mécanisme appelé wait-by-necessity

ProActive Parallel Suite se décompose en trois parties :

- La partie programmation qui offre également des outils pour le parallélisme pour fournir une base solide aux programmeurs pour construire leurs applications parallèles et distribuées
- La partie répartiteur (Scheduler) qui, à partir de commandes ou d'une interface graphique, permet la distribution et la répartition des tâches, que ce soit en interne sur un réseau local ou en externe pour un cloud par exemple
- La partie ressources fournissant un gestionnaire de ressources (Resource Manager) pour différents infrastructures existantes



Un programme ProActive n'a pas connaissance des ressources, il demande les ressources disponibles auprès du Resource Manager (l'utilisation de fichiers de configuration GCM Deployment est aussi une option porssible). Ces ressources peuvent provenir de différentes infrastructures, différents emplacements, différentes architectures de processeur. L'entière disponibilité des ressources peut être gérée par le Scheduler.

PROGRAMMATION PAR COMPOSANTS

INTRODUCTION

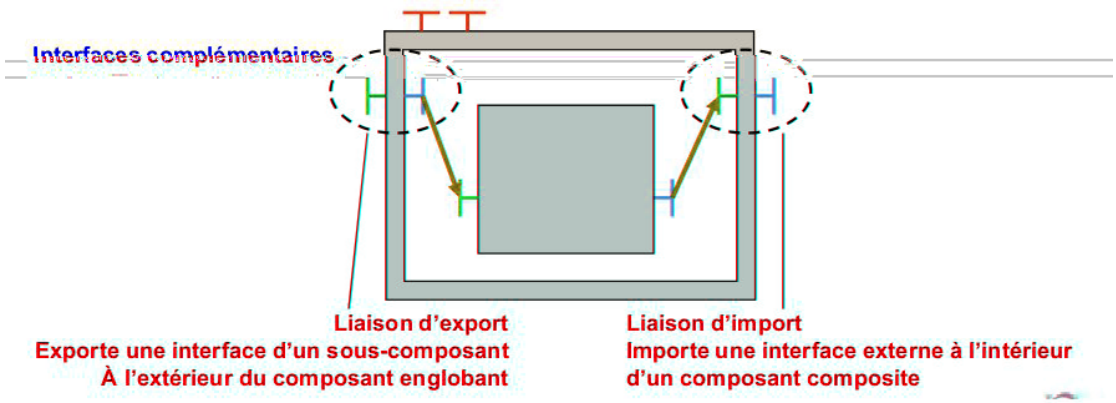
La programmation par composants a une approche modulaire de l'architecture logicielle et permet la réutilisation de code ou de certaines de ses fonctionnalités sans en connaître l'implémentation interne. L'approche composant présente les avantages d'un niveau d'abstraction plus élevé, utile pour une modélisation et une sémantique consistante (d'autant plus qu'elle dispose de capacité de réflexivité), une meilleure encapsulation permettant de reconfigurer, voire de remplacer à chaud, certaines parties si nécessaire.

COMPOSANT

Le composant est l'élément de base de ce modèle. Sa définition récursive fait qu'il peut être soit un composant primitif unitaire, soit un composant composite, i.e. un composant qui contient d'autres composants. Il peut avoir besoin d'implémenter une interface et fournit une interface. Dans tous les cas, la communication entre composants s'établit via une interface définie.

FRACTAL

Fractal est un des modèles de programmation par composants. Il existe plusieurs implémentations pour les langages C, Python, Smalltalk et Java, en particulier Julia et AOKell. Il définit un ensemble d'interfaces pour réaliser des modèles hiérarchiques de composants, c'est-à-dire que des possibilités de programmer des composants comme une composition de composants déjà existants sont présentes. Il présente les avantages d'une grande extensibilité et de connexions/déconnexions dynamiques entre composants.

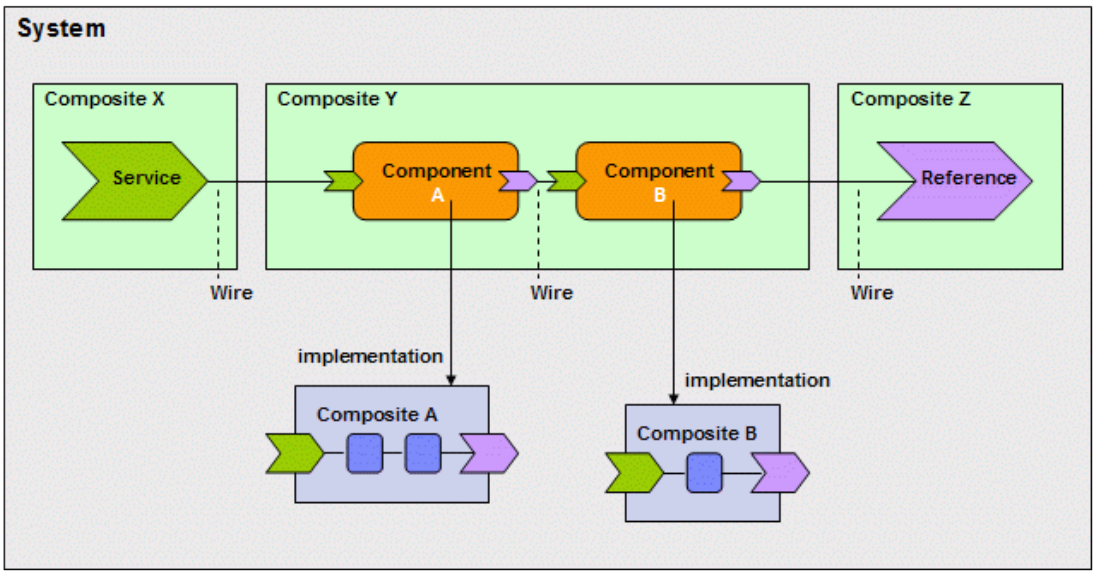


CONTROLEUR FRACTAL

Les comportements des composants sont gérés par d'autres interfaces appelées contrôleurs. Ces contrôleurs définissent les règles du composant, que ce soit pour son démarrage, son interruption ou son arrêt, les procédures de liaison ou de libération entre composants entre autres. Ce modèle assure une certaine indépendance entre blocs du programme et évite tout conflit via ce contrôle.

SCA

SCA, acronyme de Service Component Architecture, est un ensemble de spécifications qui décrivent un modèle pour construire des applications et des systèmes utilisant un SOA (Service-Oriented Architecture). Il distingue dans la construction de l'application l'assemblage d'un ensemble de composants pour former cette application de l'implémentation du composant même en tant que fournisseur de service qui consomme d'autres services. Il est plus répandu de par ses contributions du milieu professionnel dont IBM et Oracle.



SERVICES WEB

Nous évoquerons à plusieurs reprises les services Web. Ceux-ci sont des interfaces de programmation (API) qui sont accessibles via le protocole HTTP et exécutés sur un système distant hébergeant les services requis.

SERVICE WEB REST

Les services REST consistent à appeler un service sur un serveur distant sous la forme d'une URL de la forme http://adresse/serviceappele?param1=[valeur] .

WSDL

WSDL, pour Web Service Description Langage, est un langage basé sur XML qui fournit un modèle pour décrire les services Web. Il est possible d'avoir une interface Java spécifique en demandant le service disponible sur le serveur distant.

IMPLEMENTATION DES COMPOSANTS DANS DES MODELES DISTRIBUES

INTRODUCTION

Il y a de nombreux points communs entre programmation par composants et programmation distribuée. On peut imaginer le cas où différents composants tournent sur différentes machines en réseau et communiquent à travers ce dernier, chaque partie de tâche étant encapsulée dans un composant. Avec la présence de ces composants, la programmation distribuée peut agir plus indépendamment et la communication distante peut être masquée comme une liaison entre composants. Du point de vue du programmeur, son programme doit ressembler à un programme par composants traditionnel, son déploiement pouvant passer du local au distant à partir seulement de fichiers de configuration.

PROACTIVE GCM

GCM, pour Grid Component Model, est basé sur le modèle Fractal et défini comme une extension des spécifications Fractal afin de mieux cibler les infrastructures distribuées. GCM permet aux interactions entre composants d'avoir lieu avec plusieurs mécanismes distincts. *en plus des ports classiques utilise/fournit (ou client/serveur) des procédures distantes RPC, GCM permet aux ports de données, flots et évènements d'être utilisées dans des interactions entre composants. Des modèles d'interaction (mécanismes de communication) collective sont aussi supportées :

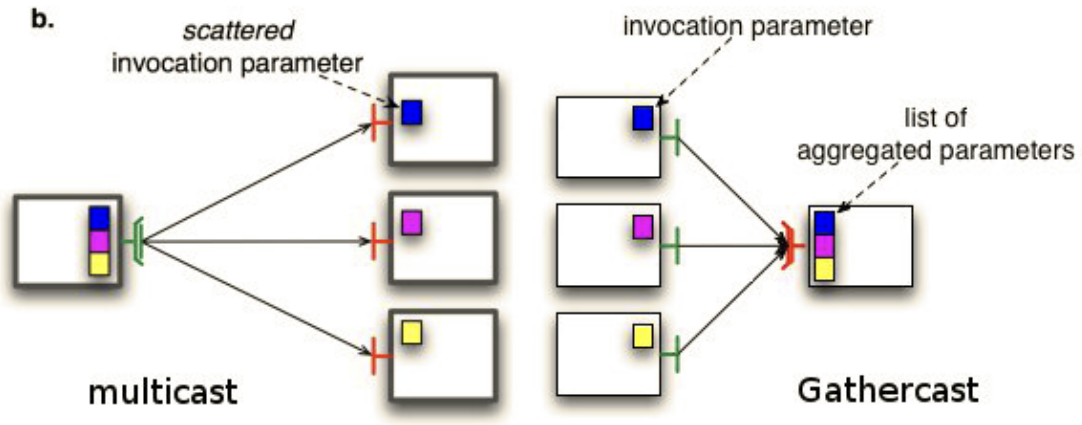
- Les ports de données permettent l'implémentation de mécanismes de partage de données. En se servant de ces ports de données, les composants peuvent être utilisés pour encapsuler des données partagées tout en préservant une certaine capacité d'interaction parmi les composants partageant les données.

- Les ports de flots permettent l'implémentation de communications à sens unique de flots de données.

- Les ports d'évènements peuvent être utilisés pour fournir des capacités d'interaction asynchrone au modèle composant. Les évènements peuvent êtres générés et utilisés pour synchroniser les composants.

- GCM supporte plusieurs genres de ports collectifs, dont ceux permettant des interactions structurées, entre une seule interface consommatrice de services et plusieurs interfaces fournisseuses de services (Multicast) ou entre plusieurs interfaces consommatrices de services et une seule interface productrice de services (Gathercast). Ces deux mécanismes d'interaction paramétrables, et ainsi personnalisables, permettent

d'implémenter tous les modèles intéressants d'interaction collective dérivant de l'utiliation de composants composites



FRASCATI

Comme son nom peut l'évoquer, FraSCAti est une combinaison des modèles Fractal et SCA. La plateforme FraSCAti supporte les spécifications SCA et l'implémentation Julia de Fractal. Un composants FraSCAti possède ainsi à la fois les aspects de Fractal et ceux de SCA, ce qui permet de créer une communication entre les deux types de composants. FraSCAti donne aussi la possibilité de lier un composant à un service Web ou d'exposer un composant comme un service Web.

ADT GALAXY

ADT (Action de Développement Technologique) Galaxy a pour objectif le développement d'une plateforme SOA de type SCA, assemblant des briques logicielles issues de différentes équipes de l'INRIA. Dans le cadre de cette ADT, l'équipe OASIS a pour charge l'intégration de composants GCM dans la plateforme Galaxy au travers de leur implémentation de référence fournie avec le middleware ProActive co-développé par l'équipe OASIS et la société ActiveEon. Une première version a déjà été réalisée avec notamment la mise en place du mécanisme de communication des composants GCM avec le reste de la plateforme Galaxy.

OBJECTIFS

DE L'INTEGRATION DE L'ASPECT SCA A L'ASSOCIATION ENTRE COMPOSANTS ET SERVICES WEB

INTEGRATION DE L'ASPECT SCA

Il s'agit ici du sujet principal du T.E.R. qui concerne l'intégration de l'aspect SCA au sein même de GCM, celui-ci ne supportant actuellement que Fractal. Nos travaux sont inspirés et influencés par le travail de l'INRIA Lille sur FraSCAti qui nous a servi de référence. Le problème majeur résidait sur comment donner un aspect SCA sur un composant et comment le manipuler. Cet objectif semblait être très vaste car nous étions néophytes sur GCM et avions absolument aucune connaissance sur SCA. Après une étude approfondie de FraSCAti et GCM, nous avons établi qu'une implémentation appropriée des contrôleurs SCA serait le meilleur moyen pour atteindre notre objectif. En voici la liste :

- SCAPropertyController : inexistant sur Fractal, nous avons compris ensuite qu'il est l'équivalent de l'AttributeController de Fractal pour les attributs
- SCAIntentController : le concept de but est totalement absent en Fractal
- SCALifeCycleController : interface identique au LifeCycleController de Fractal pour contrôler le cycle de vie d'un composant, i.e. (l'automate de) ses différents états d'exécution
- SCABindingController : interface identique au BindingController de Fractal pour les liaisons dont un composant dépend

ASSOCIATION ENTRE COMPOSANTS ET SERVICES WEB

Ce second objectif, que nous avons considéré comme optionnel, touche aux services Web. Nous avons découvert que, sur ProActive/GCM, la majeure partie des fonctionnalités de services Web concernant les composants GCM ont déjà été implémentées. Un composant peut être exposé en tant que service Web ou être connecté à un service Web. Nous avons remarqué plus tard qu'il n'y avait pas encore une implémentation pour les services RESTful, alors nous avons décidé d'étendre l'interface existante de serviceWeb pour un service RESTful spécifique : Orange SMS API. Nous pouvons ainsi appeler un service Orange à travers un programme ProActive/GCM.

REALISATION

DE LA COMPREHENSION DES CONCEPTS AU DEPLOIEMENT ET A L'IMPLEMENTATION DES COMPOSANTS

COMPREHENSION DES OBJETS ACTIFS ET PROACTIVE

Avant même de pouvoir aborder le T.E.R., il était nécessaire de comprendre en détails le fonctionnement de ProActive. Ayant déjà des bases sur la partie programmation avec les cours de Programmation distribuée, nous savions comment fonctionner un objet actif et comment ces objets communiquaient à travers le réseau. Nous avions même développé une application de vérification de nombres premiers avec cette API et déployé celle-ci sur un support NFS.

COMPREHENSION DE L'APPROCHE COMPOSANTS

De même, nous avions des bases de programmation par composants avec les cours d'Architecture logicielle. Les exercices effectués s'appuyant sur l'implémentation Julia de Fractal à partir d'annotation Java et de fichiers de configuration XML, les mécanismes de liaison et de création de composants nous étaient caché. De ce fait, nous avons commencé par implémenter plusieurs exemples à partir de l'API Java dans le but de comprendre les mécanismes internes. Nous avons réussi à les déployer et ainsi pleinement saisi le fonctionnement de Fractal.

COMPREHENSION DE PROACTIVE GCM

Vu que ProActive/GCM est une extension de Fractal, les exemples de ce dernier fonctionnent aussi avec GCM. Nous avons découvert les extensions offertes telles que les interfaces *Multicast* et *GatherCast* et réalisé plusieurs exemples décrits sur le tutoriel sans trop de difficultés. Entre-temps, nous avons obtenu notre propre branche sur le SVN de ProActive. Nous avons testé de nombreux tests JUnit existants et ainsi eu une vue d'ensemble sur la structure du projet, presque chaque ayant sa classe JUnit correspondante. A partir de là, la majorité de nos efforts était consacrée à l'intégration. Nous avions notre station de travail configuré avec Eclipse, notre compte SVN actif et étions prêts à réaliser nos objectifs.

COMPREHENSION DE FRASCATI

FraSCAti nous servait de référence, donc nous avons suivi le tutoriel sur leur site et effectué quelques exemples dans le but de mieux l'appréhender. A notre grand étonnement, FraSCAti s'est avéré être très obscur : tous les exemples étant implémentés à partir d'annotations Java et s'appuyant sur des fichiers de configuration XML, il a été impossible de trouver une API Java qui aurait pu montrer le mécanisme interne. Nous avons décidé de regarder directement leur projet à partir de SVN pour avoir un meilleur aperçu, sans succès. En effet, ce projet s'organisant avec l'outil Maven, très différent d'un projet traditionnel Ant, il nous a fallu un certain temps pour nous adapter à cette plateforme. Nous avons enfin découvert certaines de leurs classes internes et essayé de comprendre les liens entre les résultats après exécution et les appels de l'API Java. Apparemment, la plupart des classes sont générées à la volée lors de l'exécution grâce aux annotations Java. Nous avons également déployé leurs exemples de service Web bien que nous n'ayons pas vu comment cela fonctionnait en interne.

DEPLOIEMENT DE COMPOSANTS DISTRIBUES SUR RESEAU

Nous avons manipulé le Resource Manager plusieurs, en déployant dans un premier temps des composants distribués sur une machine locale puis, dans un seconde temps, sur des machines différentes communicant entre elles. Pour cela, le code devant rester identique, nous avons déployé un Resource Manager, y avons ajouté les machines souhaitées du réseau et avons créé des nœuds virtuels sur ces machines, de sorte que l'application demande au Resource Manager ces nœuds virtuels sans pour autant en connaître leur provenance.

IMPLEMENTATION DE L'ASPECT SCA DANS GCM

Presque toutes les conditions étaient réunies pour réaliser nos implémentations. Nous avons rapidement réalisé que les *LifeCycleContoller* et *BindingController* sont identique à ceux de Fractal et sont directement compatibles avec GCM. Par contre, nous nous sommes alors intéressés au *SCAPropertyController*. Après plusieurs jours de tests et de documentation, nous nous sommes rendus compte que le *property* est en fait équivalent à l'*attribute* Fractal. Pour illustrer cette notion de propriété ou attribut, considérons le composant suivant à relier à un service Web et dont nous devons donner la valeur de l'URL où il doit se connecter :

```
class ComponentToBindOnAWebService implements MyAttributeController ....  {
    .....
    private String UrlToBind;
    ....
}
```

L'affectation de l'attribut UrlToBind doît être effectué avec un *AttributeController* dont nous montrons son utilisation ci-dessous :

```
class TestAttributeController {
    .... // initialisation ....
    Component c = gf.newFcInstance(t, "primitive",
                ComponentToBindOnAWebService.class.getName()); // create component
    GCM.getGCMLifeCycleController(c).startFc();  // start component
    MyAttributeController ca = (MyAttributeController )
    GCM.getAttributeController(c);  // get attribute controller
    ca.setUrlToBind(_Http://adress_);  // set attribute value
    assertEquals(_Http://adress_, ca.getUrlToBind());
}
```

Ce que le *SCAPropertyController* effectue peut être traduit par un appel de l'*AttributeController* d'une manière ou d'une autre. Nous avons pris l'interface du *SCAPropertyController* de FraSCAti à l'identique et adapté son implémentation pour GCM. Nous avons demandé une instance *d'AttributeController* à partir du composant et converti les appels via le *SCAPropertyController* en des appels de l'*AttributeController*. Voici ci-dessous le même en utilisant cette fois-ci le *SCAPropertyController* :

```
class TestPropertyController {
    .... // initialisation ....
    Component c = gf.newFcInstance(t, "primitive",
                ComponentToBindOnAWebService.class.getName());
    GCM.getGCMLifeCycleController(c).startFc();
    SCAPropertyController scac = Utils.getSCAPropertyController(c); //get property controller
    scac.init();  //init controller
    scac.setValue("UrlToBind", _Http://adress_); //set property value
    assertEquals(new String(_Http://adress_);
    scac.getValue("UrlToBind"));
}
```

Nous avons implémenté les appels dans le *SCAPropertyController* par reflexivité. Certaines de ses caractéristiques n'ont pas pu être implémentées, comme

```
void setType(String name, Class<?> type)
```

Nous ne pensons pas qu'il soit possible de changer dynamiquement le type de propriété pendant l'exécution. Nous avons en plus écrit un test JUnit afin de prouver la validité de notre travail. La seconde étape a été d'implémenter le générateur pour le *SCAPropertyController* afin de respecter les standards des spécifications SCA en Java, ce qui nous a obligés d'utiliser les annotations. Nous avons dû l'adapter en générant automatiquement (avec l'aide de Javassist) une sous-classe qui peut obtenir un *SCAPropertyController*. Le composant peut éviter d'implémenter l'interface *AttributeController*. Le composant devrait être comme suit :

```
class ComponentToBindOnAWebService implements ...  {
    .....
    @property
```

```
    Private String UrlToBind;
    ....
}
```

Le reste du mécanisme est inchangé.

CREATION D'UNE APPLICATION UTILISANT DES COMPOSANTS SCA ET GCM AVEC DES SERVICES WEB

En fin de T.E.R., nous avons développé une petite application qui connecte un composant avec un aspect SCA sur le service REST Orange SMS. La liaison entre un service REST et un composant n'existe pas encore mais une interface générique pour les services Web est déjà présente heureusement. Nous avons étudié l'implémentation existante de WSDL et défini notre propre interface de

RESULTATS ET CONCLUSION

RESULTATS

Nous avons accompli avec succès plusieurs parties des objectifs. Parmi les quatre contrôleurs, deux d'entre eux sont supportés nativement par ProActive/GCM. Nous avons rendu possible les caractéristiques des propriétés de SCA dans un composant GCM en implémentant le SCAPropertyController. Nous avons appréhendé les services Web existants pour GCM et développé une application faisant appel à un service REST Orange. La partie que nous n'avons pas pu réalisée est l'intégration des caractéristiques de but avec les SCAIntentController. Nous ne pensons pas que ce soit un point noir de notre projet vu la difficulté de cette partie et la quantité de concepts et outils à comprendre. Nous avons fait beaucoup de recherches et d'expérimentations pour saisir les principes en profondeur et sommes satisfaits de l'accomplissement de ce projet.

CONCLUSION

Notre projet ne brille pas, certes, par un lourd développement de code, mais nous avons concentré nos efforts sur la recherche scientifique en comprenant une architecture logicielle complexe à intégrer dans un projet complexe existant. Le développement de notre code est bref mais efficace ; le seul point noir étant, nous pensons, que nous avons mal calculé la difficulté du sujet, ce qui a directement causé un travail incomplet sur les contrôleurs. Nous avons fourni de grands efforts et une grande quantité de travail qui, nous pensons, mérite une bonne note.