

---

# OPAD : Outils Pour Architectures Dynamiques

Olivier Barais<sup>\*,\*\*</sup> — Laurence Duchien<sup>\*</sup>

*\*LIFL UPRESA CNRS 8022, Equipe GOAL, USTL  
Bât M3 F-59655 Villeneuve d'Ascq Cedex*

*\*\*Ecole des Mines de Douai, Dept GIP, BP 838, 59508 Douai cedex  
{barais,duchien}@lifl.fr*

---

*RÉSUMÉ. Nous présentons dans ce papier un modèle abstrait qui enrichit la description de l'architecture d'une application grâce à l'extension de la notion de composite. Nous proposons d'y ajouter des connaissances globales de l'application regroupées sous le vocable de groupe. Il est la base d'un ensemble d'outils de gestion d'architectures dynamiques. Ce modèle a pour objectif d'améliorer la description de la communication, la hiérarchisation et la configuration au sein d'une application à base de composants.*

*ABSTRACT. We present in this paper an abstract model that expands the software architecture description thanks to the extension of the component composite concept. We add global knowledge of the software to the model. This model aims to improve the description of the communication, the hierarchisation and the configuration within components based softwares.*

*MOTS-CLÉS : groupe, Architecture logicielle dynamique, ADL, ArchJava*

*KEYWORDS: group, dynamic software architecture, ADL, ArchJava*

---

## 1. Introduction

Spécifier, concevoir, développer, maintenir, faire évoluer des applications réparties à base de composants demandent beaucoup d'efforts aux acteurs du génie logiciel. Ceci est essentiellement dû aux manques de méthodes et d'outils appropriés. Le développement autour du *middleware* atteint actuellement ses limites. La tendance qui se dessine est l'ingénierie des modèles avec le MDA (*Model Driven Architecture*) [SOL 00]. Ainsi, les modèles conceptuels, tels que les modèles UML produits lors des étapes de spécification ou de conception, constituent des productions pérennes et capitalisables dans le cycle de développement d'un logiciel ou d'un système d'information.

La description d'une application se fait en plusieurs étapes : de la description de l'ensemble des modules logiciels nécessaires et des interactions entre ces modules à la spécification fonctionnelle de chacun de ces modules. La première étape consiste donc à décrire le plan de l'application appelé aussi architecture. Il n'existe pas de modèle d'architecture logicielle adapté à l'ensemble des besoins des applications. Depuis les

années 80, plusieurs équipes travaillent sur ces questions et proposent de nouveaux langages appelés *Architecture Description Languages* (ADLs) permettant l'expression de l'architecture de l'application. Il existe une grande diversité sur ces ADLs. Cependant on retrouve de nombreuses caractéristiques communes telles que les deux mêmes objectifs, c'est à dire la génération de code et la validation d'architecture ainsi que les trois mêmes briques de base. Ces trois briques sont le composant qui modélise les modules logiciels, le connecteur qui modélise les interactions entre ces modules et finalement la configuration aussi appelée *composite* dans certains ADLs qui représente l'ensemble des composants et des connecteurs nécessaires au fonctionnement de l'application. Ces langages présentent plusieurs lacunes telles que la difficulté de l'expression de la dynamique et de l'évolution de l'architecture d'une application. Nous présentons dans ce papier un modèle qui enrichit la description de l'architecture d'une application grâce à l'extension de la notion de composite. Nous proposons d'y ajouter des connaissances globales de l'application regroupées sous le vocable de *groupe*. La notion de groupe met en place un service de communication entre ses membres. Il permet de modéliser la hiérarchisation de l'application et offre des services métiers de groupe composés de services métiers de ses membres. Enfin il prend en charge certains services non fonctionnels de groupe comme la tolérance au panne ou la réplication. Ce concept est la base d'OPAD (Outils Pour Architectures Dynamiques) un ensemble d'outils pour la conception et la gestion d'architectures d'applications réparties.

Ce papier s'articule en trois parties. La première section présente à la fois les travaux issus de la communauté de l'architecture logicielle et les travaux s'intéressant au paradigme de groupe. La deuxième partie décrit notre modèle prenant en compte le concept de groupe ainsi que sa mise en œuvre. Enfin nous dressons un bilan de notre étude, et discutons des futurs travaux à mener.

## **2. Travaux connexes**

### **2.1. L'architecture logicielle**

L'architecture logicielle d'une application se définit comme une spécification abstraite en termes de modules logiciels qui le constituent et des interactions entre ces modules [COU 01] et laisse en second plan la programmation des applications. La notion d'architecture véhicule donc une distinction nette entre la structure conceptuelle d'une application et ses possibles réalisations techniques sur des cibles particulières.

#### **Les Langages de Description d'Architectures**

Les langages de description d'architectures constituent le cœur des travaux sur l'architecture logicielle. Ils définissent un vocabulaire commun entre les acteurs autour de l'architecture. Ils permettent la création, la validation et l'évolution d'une architecture, la génération rapide d'un prototype, l'administration et la configuration d'une application. Ils doivent aussi permettre d'exprimer la dynamique de l'application, la hiérarchisation des composants, les interactions entre les composants et le déploiement de l'application. En ce qui concerne les notations offertes, on constate une très grande diversité entre les ADLs. Néanmoins les trois concepts de composant, connecteur et configuration sont généralement acceptés comme essentiels [MED 00].

Le **composant** logiciel [MAR 02] est une entité autonome capable d'interopérer avec son environnement à l'aide d'une interface. Deux parties définissent un composant. Une première partie, dite externe, correspond à son interface. Elle comprend la description des services fournis et requis par le composant. Elle définit les interactions du composant avec son environnement. La seconde partie, dite interne, correspond à son implantation et permet la description du fonctionnement interne du composant.

Le **connecteur** correspond à un élément d'architecture qui modélise de manière explicite les interactions entre un ou plusieurs composants. Il contient les informations concernant les règles d'interactions entre les composants.

Composants et connecteurs sont des types pouvant être instanciés et assemblés à partir de leurs interfaces pour former une **configuration**. Parmi les propriétés associées aux configurations dans les différents ADLs, la composition hiérarchique est considérée comme primordiale. Cette propriété permet de spécifier un composant comme un assemblage de sous-composants, appelé aussi composite. Celui-ci est l'élément de base de modélisation de la configuration.

Notre étude s'est portée plus précisément sur cinq ADLs, deux langages de description : UniCon [ZEL 96] et Rapide [Rap97] et trois langages de configuration Darwin [Dar], ArchJava [ALD 02], Olan [BEL 97]. Nous avons identifié certaines lacunes dans ces modèles ; la possibilité d'exprimer la dynamique et l'évolution de la configuration, c'est à dire l'expression du comportement de l'application reste un problème. Notre objectif est donc de bâtir un modèle d'architecture extensible et dynamique en retravaillant sur des extensions de la notion de composite.

## 2.2. Utilisation du concept de groupe en informatique

Le concept de groupe est déjà très utilisé en informatique, les applications ont des besoins de communication et de contrôle mettant en oeuvre non plus deux entités, mais un ensemble d'entités [DUC 99]. Le *groupe* a dès lors été très étudié par les communautés de recherche de la tolérance au panne ou du travail collaboratif. Il véhicule plusieurs idées et permet la mise en oeuvre de plusieurs services. Par exemple, le groupe inclut généralement l'implantation d'un gestionnaire de membres (*membership service*), d'un service de diffusion (*multicast service*) et d'un gestionnaire de cohérence (*consistent manager*) [PHA 01].

Notre attention s'est portée sur trois mises en oeuvre du groupe, trois visions différentes de la notion de groupe : JavaGroups [jav01], une bibliothèque Java permettant la communication fiable de groupe basée sur IP *multicast*, Jgroup [MON 00], une intégration du concept de groupe dans une application construite à base d'objets répartis, OpenORB [SAI 01], un *middleware* adaptable et réflexif intégrant le paradigme de groupe. Dans l'ensemble de ces systèmes, le groupe est utilisé comme artefact pour masquer la complexité de la communication ou de la collaboration entre plusieurs modules logiciels. Il est surprenant de constater que les outils de modélisation permettant de simplifier le travail du concepteur d'applications réparties de grande taille ne prennent pas en compte un concept fortement utilisé dans la réalisation des applications. Ainsi notre approche a pour but de montrer l'intérêt de la prise en compte du concept de groupe dans la description de l'architecture d'une application.

### 3. Notre modèle

#### 3.1. Objectif

L'objectif de l'extension de l'élément composite est d'améliorer l'expression de la communication, de la hiérarchisation et de la configuration de l'application.

##### **Un élément de communication**

Les applications ont de plus en plus de besoins de communication et de contrôle mettant en oeuvre non plus deux entités, mais un ensemble d'entités. La notion de groupe permet la mise en place d'un protocole de communication de groupe. Il permet l'envoi et la réception de messages entre membres du groupe (unicast) ou l'envoi d'un message à tous les membres du groupe (multicast).

##### **Un élément de hiérarchisation.**

Un composite fournit des services métiers qui sont décrits comme une composition des services métiers de ses membres. Il peut lui même être membre d'un groupe. La notion de groupe permet de structurer l'architecture de l'application. Une application est décrite comme un graphe où chaque noeud est un groupe et chaque feuille un composant primitif.

##### **Un élément de configuration.**

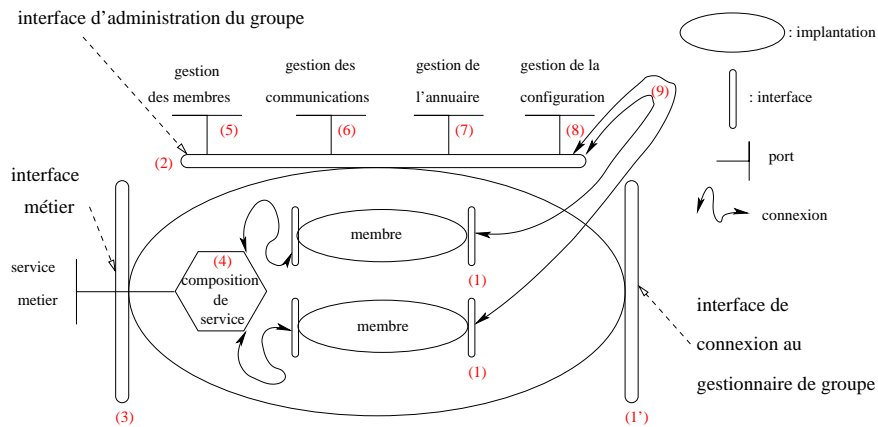
La description d'une configuration dans la plupart des ADLs est très souvent statique. La notion de groupe permet d'améliorer l'expression de la dynamique de la configuration. Il va offrir une interface standard pour connaître la configuration de ses membres et modifier sa configuration.

#### 3.2. Vue d'ensemble

Notre modèle étend la notion de composant. Un composant est structuré en deux parties : une interne, l'implantation et une externe, l'interface. L'interface est composée de ports. Un port permet l'expression de services requis et fournis par le composant. Un port est donc bi-directionnel. Une connexion se fera toujours entre deux ports dont les services sont complémentaires. Il n'y a pas de connecteur défini explicitement. Toute l'information qui caractérise une connexion entre deux composants est contenue dans les ports. Pour définir des protocoles évolués, les connecteurs sont décrits par des composants primitifs rendant des fonctions de protocole.

L'extension de ce modèle de base proche de ArchJava [ALD 02] se fait en deux étapes (voir Figure 1). On ajoute une interface de connexion au groupe à l'ensemble des composants primitifs afin qu'ils aient conscience d'appartenir à un groupe (1). On étend le composant composite en lui ajoutant une interface supplémentaire. Il possède alors trois interfaces (1',2,3). Une interface en charge de la gestion de l'administration du groupe (2), une en charge de présenter à l'extérieur les services métiers fournis et requis par le groupe (3). Enfin une interface est créée afin de pouvoir connecter le groupe à d'autres groupes (1'). L'interface d'administration de groupe possède quatre ports (5,6,7,8) strictement complémentaires aux ports de l'interface de connexion au groupe afin de pouvoir les connecter sans difficulté (9). Les services fournis par cette

interface vont permettre de gérer la communication interne au groupe, d'administrer le groupe et de faire évoluer sa configuration. Enfin l'interface métier du groupe (3) permet de présenter les services requis et fournis par le groupe. Les services métiers du groupe seront décrits comme une composition des services métiers de ces membres. Un premier niveau de langage dédié à cette description a été spécifié.



**Figure 1.** Extension du composant composite = Groupe

### 3.3. Un prototype basé sur ArchJava

Notre choix s'est porté sur ArchJava que nous avons adapté pour qu'il prenne en compte la notion de groupe. Cette extension s'est faite par l'ajout au dessus de ArchJava, d'un langage de description d'architecture décrit par une DTD XML et répondant aux contraintes de notre modèle. La compilation du code se fait alors en deux passes. Nous analysons le code XML et nous générons du code vers ArchJava. Nous compilons ensuite l'application grâce au compilateur ArchJava. L'expression de la notion de groupe n'étant pas possible par défaut dans ArchJava, nous construisons les groupes comme des composants composites enrichis de services de groupe. Les possibilités, les atouts et les limites du prototype sont détaillés dans ce mémoire de DEA [BAR 02].

## 4. Bilan et Perspectives

Ce travail nous a permis de définir un modèle qui met l'accent sur la notion de composite et de présenter les prémisses de nos outils OPAD. L'expression de la notion de groupe dans un ADL permet une spécification plus fine de l'architecture : une meilleure expression de la communication, de la hiérarchisation et de la dynamique de la configuration.

Parmi les travaux à réaliser, l'étude doit être approfondie afin de mieux prendre en compte la répartition dans notre modèle. Nous travaillons également sur les protocoles

de groupe afin d'intégrer dans le modèle des protocoles fiables, ordonnés et atomiques. Enfin une des étapes importantes sera la spécification de la composition de services. Le travail sur le prototype a permis la mise en œuvre du modèle.

La prochaine étape est de définir une chaîne de conception d'applications basée sur une transformation successive de modèles. Nous voulons par cette chaîne décrire un lien possible entre les ADLs, en particulier notre modèle d'architecture et le MDA de l'OMG. La conception d'une application se fera par définition de l'architecture globale de l'application puis par projection vers une plate-forme d'exécution. La phase de spécification de l'architecture s'intègre alors complètement dans la phase initiale de conception au même titre que UML. L'étude du lien entre un modèle d'architecture et d'autres modèles de conception comme UML nous intéresse également car elle impose la description de notre méta-modèle d'architecture afin de pouvoir décrire la transformation entre modèle UML et modèle d'architecture. Notre approche doit permettre d'améliorer notre modèle en identifiant certaines lacunes ou imprécisions, de vérifier certaines propriétés d'architecture, de décrire la transformation de notre modèle vers d'autres plate-formes, d'automatiser cette transformation, et de générer du code vers des plate-formes à composants. Par exemple la plate-forme académique OpenORB [SAI 01] nous permettrait de montrer une adéquation entre notre modèle abstrait d'architecture et un modèle technique prenant en compte la notion de groupe. Nous pensons aussi en marge de cette chaîne de conception, travailler sur l'utilisation du groupe comme concept pour décrire l'héritage d'interface de composants.

## 5. Bibliographie

- [ALD 02] ALDRICH J., CHAMBERS C., NOTKIN D., « ArchJava : Connecting Software Architecture to Implementation », *ICSE 2002*, 2002.
- [BAR 02] BARAIS O., « Approche statique dynamique et globale de l'architecture d'applications réparties », Master's thesis, Mémoire de DEA d'informatique fondamentale de l'Université de Lille1 ,LIFL - équipe GOAL, juillet 2002.
- [BEL 97] BELLISSARD L., « Construction et Configuration d'application réparties », PhD thesis, institut nationale polytechnique de Grenoble, Décembre 1997.
- [COU 01] COUPAYE T., LENGLET R., BEAUVOIS M., DECHAMBOUX P., « Composants et composition dans l'architecture des systèmes répartis », FT RD, octobre 2001.
- [Dar] « La page Web de Darwin », <http://www.doc.ic.ac.uk/igeozg/Project/Darwin/index.html>.
- [DUC 99] DUCHIEN L., « Habilitation à Diriger des Recherches : Modèles de Programmation, Services Systèmes et Reflexivité pour la Coopération de Groupes d'Objets Répartis », PhD thesis, Université Joseph Fourier - Grenoble, 1999.
- [jav01] « JavaGroups - A Reliable Multicast Communication Toolkit for Java », 2001, <http://www.cs.cornell.edu/Info/Projects/JavaGroupsNew/>.
- [MAR 02] MARVIE R., PELLEGRINI M., « Modèles de composants, un état de l'art », *RSTI-L'objet*, vol. Coopération et systèmes à objets, 2002, page 61 à 89.
- [MED 00] MEDVIDOVIC N., TAYLOR R. N., « A Classification and Comparison Framework for Software Architecture Description Languages », vol. 26, n° 1, 2000, page 23.
- [MON 00] MONTRESOR A., « Jgroup Tutorial and Programmer's Manual », rapport, Octobre 2000, University of Bolona.

- [PHA 01] PHAM N., « Interaction et communication de groupe », juin 2001, Mémoire de DEA RSD (Réseau et Systèmes Distribués), Université de Nice.
- [Rap97] Rapide Design Team Program Analysis and Verification Group Computer Systems Lab Stanford University, « Guide to the Rapide 1.0 Language Reference Manuals », 1997.
- [SAI 01] SAIKOSKI K., COULSON G., BLAIR G., « Configurable and Reconfigurable Group Services in a Component Based Middleware Environment », Distributed Multimedia Research Group, Department of Computing, Lancaster University, 2001.
- [SOL 00] SOLEY R., THE OMG STAFF STRATEGY GROUP, « Model Driven Architecture », 2000, <ftp://ftp.omg.org/pub/docs/omg/00-11-05.pdf>.
- [ZEL 96] ZELESNIK G., « The UniCon Language Language User Manual », School of Computer Science Carnegie Mellon University, Pittsburgh, Juin 1996.