

# Reconfiguration d'applications réparties à base de composants

Bassem KOSAYBA, Raphaël MARVIE, Philippe MERLE, Jean-Marc GEIB

*Laboratoire d'Informatique Fondamentale de Lille*  
*UPRESA CNRS 8022*  
*59655 Villeneuve d'Ascq*  
`{kosayba,marvie,merle,geib}@lifl.fr`

17 juillet 2002

## Résumé

*L'utilisation d'intergiciels à base d'objets répartis est une première étape vers la simplification de la mise en œuvre des applications réparties : ils masquent les aspects liés à la communication.*

*L'apparition des intergiciels à base de composants répartis représentent une autre étape majeur pour permettre la production d'applications configurables : support de la phase de déploiement et expression de la connectivité.*

*Toutefois, les capacités, en termes de dynamisme, de ces applications reste limitées. Parallèlement, les besoins des applications en terme d'adaptabilité, simple reconfiguration ou ajout de fonctionnalités, s'accroît.*

*Pour répondre à ces besoins, cet article propose un environnement pour définir et exploiter des architectures reconfigurables à l'exécution. Le second plan d'adaptabilité discuté ici est la possibilité d'utiliser un même modèle abstrait d'architecture avec différentes solution technologiques.*

## 1 Introduction

Le concept d'application répartie est aujourd'hui un des éléments fondamentaux pour construire des systèmes d'information. La plupart des applications réparties sont basées sur les intergiciels qui cachent la complexité des communications. Les intergiciels les plus courants suivent le modèle objet.

Dans ce contexte, une application répartie est un groupe d'objets distants interconnectés entre eux. Toutefois ces intergiciels supportent difficilement le déploiement et la configuration des applications réparties à grande échelle [1], essentiellement à cause de deux lacunes :

- les interconnexions entre les objets répartis sont cachées dans l'implémentation et configurées au moment de la compilation. Elles ne peuvent donc pas être manipulées depuis l'extérieur par un architecte.
- il n'existe pas de phase de déploiement pour installer automatiquement les objets sur leurs sites d'exécution.

Pour corriger ces inconvénients, les intergiciels ont évolués vers les modèles de composants. Ces modèles introduisent :

- la notion de « port » qui définit explicitement les points d'accès fournis et requis par le composant en vue de son interconnexion. Ceci rend l'application configurable.
- le processus de déploiement pour installer les composants sur leurs sites d'exécution, initialiser leurs attributs par défaut et les interconnecter ensemble selon un fichier descriptif (le fichier qui configure l'application).

On peut reprocher au modèle de composant que la description de l'architecture de l'application est figée et statique. On ne peut ni ajouter des nouvelles connexions ni avoir une vision de l'application pendant l'exécution.

L'aspect distribué de l'environnement le rend variable : par exemple, ajout et suppression de sites d'exécution. Dans ce cas, on doit agir pour adapter le comportement de l'application en fonction de ces changements. L'apparition de nouveaux besoins entraîne aussi la nécessité d'adaptation. Certaines de ces adaptations sont réalisées par des changements de l'architecture de l'application (remplacement d'un composant, adjonction d'une nouvelle connexion, etc). La seule façon pour modifier l'architecture de l'application est d'arrêter l'application, de modifier les fichiers descriptifs et de relancer le processus de déploiement. Toutefois, certaines applications ne supportent pas de telles interruptions, comme les applications de télécommunication. Il nous faut donc un aspect dynamique pour reconfigurer l'application à la volée pendant l'exécution.

Dans le projet Olan [2] [3], la construction de l'application distribuée est le fait de configurer les interconnexions entre les composants de l'application. Toutefois, Olan n'

Pour les ADLs (Architecture Definition Language), l'architecture d'une application est une collection de composants interconnectés entre eux. Une architecture idéale est un groupe de composants, de connexions entre ces composants et de contraintes (comment les composants interagissent). L'intérêt des ADLs est de rendre la conception plus précise et d'assister l'architecte à bien analyser et comprendre le problème. Toutefois, la plupart de ces ADLs permettent une description statique de l'architecture et sans couplage avec des langages de mise en œuvre. On peut exclure de ces ADLs le projet ArchJava [8] [9]. Ce projet fournit un compilateur pour construire une description de l'architecture de l'application couplée avec une implémentation en Java. En plus, il présente une infrastructure dynamique qui permet de connecter et déconnecter les composants pendant l'exécution. Toutefois, il ne présente pas un outil pour vraiment exploiter cette infrastructure pendant l'exécution : la dynamique est programmée et donc figée.

Cet article se décompose comme suit. La section 2 présente nos deux principaux objectifs : l'indépendance de la plate-forme industrielle et la capacité d'adaptation. Dans cette section, on introduit les concepts de la méta-modélisation et le projet CODEX qui répond à notre premier objectif. La section 3 présente une vue d'ensemble et une proposition pour réaliser notre deuxième objectif. Enfin, la section 4 conclut cet article en indiquant ces points principaux.

## 2 Contexte et motivation

Notre souhait est que le modèle abstrait de composants prenne en compte des capacités d'adaptation, c'est-à-dire la capacité de re-configurer l'architecture de l'application pendant l'exécution en vue de changer sa structure et son comportement.

Nous voulons de plus que ce modèle soit suffisamment général pour qu'il soit utilisable avec les différentes plates-formes industrielles : l'application pourra être générée à partir d'un modèle indépendant de la plate-forme industrielle. De cette façon, le modèle est conçu et est projeté vers une plate-forme spécifique ou préservé de manière abstraite pour permettre la projection vers une future plate-forme d'exécution.

Pour que le projet CODEX (Composite Oriented Description and eXecution) puisse unifier les concepts de différentes plates-formes de composant, il se base sur la méta-modélisation MOF (Meta Object Facility) de l'OMG. Cette dernière permet la définition d'un modèle abstrait d'applications et fournit dans un niveau méta supérieur (le niveau méta-modèle) les concepts pour exprimer ce modèle.

### 2.1 L'organisation de la méta-modélisation MOF

Le MOF est organisé en quatre niveaux [5] [6]. Chaque niveau contient des informations décrivant le niveau inférieur. L'application est un groupe d'éléments qui interagissent ensemble. On décrit la sémantique de ces éléments et leurs relations (M0) au niveau du modèle (M1 ou ce qu'on appelle méta-information). Pour décrire le modèle d'une application à base de composants par

exemple, il faut disposer de la notion de composant, de connexion, de port, etc. Ces notions sont décrites au niveau du méta-modèle (M2). Ce dernier se base sur des concepts généraux (classe, association, etc) pour décrire le modèle. Ces concepts sont définis dans le niveau méta-méta-modèle (M3) qui est auto-descriptif.

La figure 1 illustre l'organisation de la pile de méta-modélisation du MOF en prenant CODEX comme un cas d'étude.

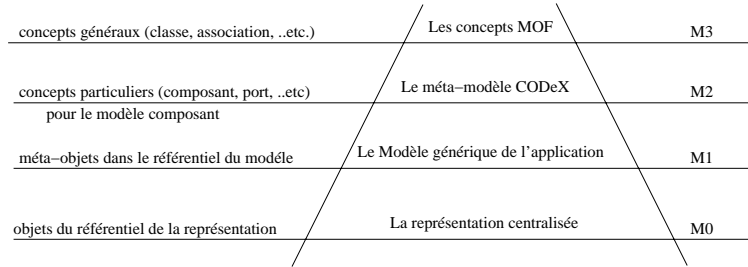


FIG. 1 – CODEX et la méta-modélisation de MOF

## 2.2 Le méta-modèle CODEX

Un des objectifs de CODEX [4] est la spécification du méta-modèle pour la définition d'architecture. En CODEX on définit le composite comme un groupe de composants, de ports et de connexions. De plus, un jeu d'opérations permet la manipulation des concepts précédents pour supporter le dynamisme des architectures. La figure 2 illustre comment le méta-modèle CODEX est défini. CODEX fournit aussi un référentiel pour définir, stocker et utiliser une version réifiée des architectures d'applications.



FIG. 2 – La définition du méta-modèle CODEX

Pour construire le modèle d'une application, il faut interagir avec le référentiel méta-modèle et invoquer les opérations comme la création d'un méta-objet qui représente un composant, la création d'un méta-objet représentant un port, l'adjonction de ce port dans ce composant, la connexion d'un port avec un autre, etc. Le référentiel contient un graphe de méta-objets mise en œuvre avec des objets CORBA. Ces méta-objets sont facilement manipulables (création, destruction, modification et utilisation) à partir de l'environnement de script Idscript [7].

## 2.3 Manipulation d'architecture

La première étape est la définition de l'architecture de l'application en définissant un modèle générique. La construction de ce modèle est réalisée par une suite d'invocations sur des opérations définies au niveau méta-modèle et mises en œuvre dans le référentiel (création de méta-objet représentant un composant, création d'un méta-objet représentant un port, etc).

Quand l'assemblage du modèle est terminé, on peut passer à l'exploitation en projetant le modèle vers une plate-forme réelle. Le résultat est l'application déployée sur les sites d'exécution et une représentation centralisée de l'architecture de l'application.

En cas de besoin, on peut modifier le comportement de notre application distribuée : on peut modifier la représentation de l'architecture pour reconfigurer l'application. Comme cette représentation est en cohérence permanente avec l'application répartie, ces changements sont répercutés sur les entités concernés de l'application.

Pour que notre travail soit accepté par les architectes des applications distribuées, on facilite les différentes étapes par une interface graphique. Cette interface graphique cache les détails compliqués pour permettre aux architectes de ne s'occuper que de la logique métier des applications.

## 3 Proposition

Dans ce paragraphe, nous présentons une proposition pour développer des applications adaptables et génériques à base de composants.

### 3.1 Vue d'ensemble

La figure 3 donne une vision complète de notre travail.

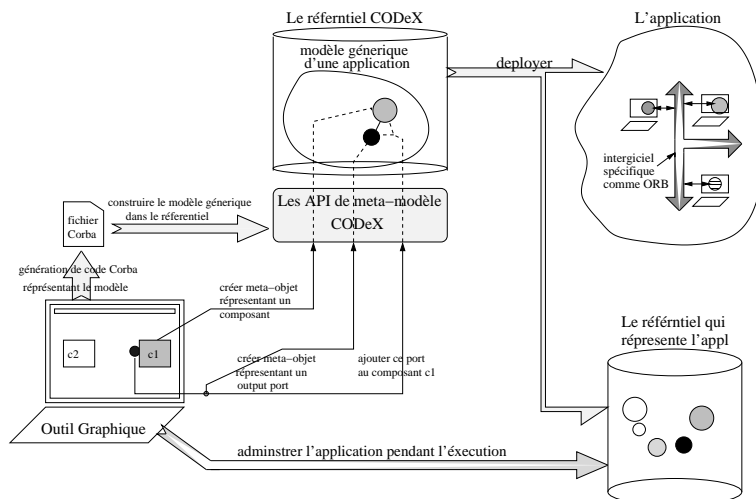


FIG. 3 – *Vision globale (conception + exploitation + adaptation)*

Le référentiel CODeX met en oeuvre le méta-modèle. Ce référentiel offre une API pour construire et manipuler le modèle de l'application.

Le référentiel de la représentation de l'application contient des objets qui offrent une représentation centralisée de l'architecture de l'application distribuée. Ces objets sont liés aux éléments de l'application et permettent le contrôle de l'architecture répartie. Le chargement de ce référentiel est simultané avec le déploiement de l'application. Ce référentiel est spécifique à la plate-forme industrielle. Toutefois, l'API de contrôle est identique. Ce référentiel répond au manque de dynamisme mentionné tant au niveau des composants que des ADLs. Il représente un moyen de re-

configuration possible pour adapter l'application à une nouvelle situation. Cette re-configuration consiste à changer les valeurs des attributs, les connexions, à ajouter un nouveau composant, etc.

L'outil graphique joue le rôle d'atelier de conception pendant la définition du modèle. Il prend en charge la vérification de la construction de l'architecture et l'invocation des opérations du référentiel CODEX en vue de créer le modèle de l'application. Cet outil joue également le rôle de console de contrôle à l'exécution de l'application et il permet la reconfiguration dynamique.

### 3.2 Expérimentation

Dans un premier temps, l'architecture d'une application était construite à l'aide d'un script de chargement. Actuellement la console permet la génération de ces scripts à partir de schémas graphiques. Une version sérialisée du schéma est également générée en vue de sa réutilisation.

Ceci permet à l'architecte de reprendre la représentation graphique d'un modèle à partir d'un fichier associé au code généré, ce qui permet la réutilisation des architectures. Enfin, cette console est en cours d'adaptation pour interagir directement avec le référentiel au travers de son API.

### 3.3 Développement à venir

Le support d'adaptation : on va définir un référentiel qui abrite des objets CORBA. Ces objets sont associés aux entités de l'application comme (composant, port, connexion, etc) et sont dotés des méthodes qui vont déléguer le travail aux opérations de telle sorte que (connecte, disConnecte, etc) au niveau des entités réels de l'application. Ces opérations sont différentes d'une plate-forme industrielle à l'autre.

L'interaction entre l'outil graphique et le support d'adaptation : la définition précédente du référentiel présente un API unique pour modifier l'architecture de l'application quoique la plate-forme exécutive. Donc, il reste que l'outil graphique présente l'interface graphique convenable à cet API.

## 4 Conclusion

Cet article a tout d'abord présenté l'évolution de la définition d'architectures à base d'objets vers la définition à base de composants logiciels. Puis, il a montré une limitation des propositions actuelles de définition d'architectures à base de composants, que ce soit sous forme de descripteurs ou d'ADL : l'absence de représentation au cours d'une exécution et la dépendance vis-à-vis de la plate-forme de mise en œuvre.

Ensuite, l'approche CODEX a été discuté pour pallier cette limitation : l'utilisation d'un référentiel comme base de la description abstraite d'une architecture et indépendant de la plate-forme industrielle. Enfin, nous avons proposé l'utilisation d'un autre référentiel utilisable pendant l'exécution comme base de la représentation et de la re-configuration de l'architecture de l'application répartie.

Les avantages de cette approche sont l'indépendance vis-à-vis de la plate-forme industrielle, la disponibilité à l'exécution, la dynamique dans l'utilisation de la description de l'architecture d'une application. En vue de garantir la flexibilité, on dote les différentes étapes de la conception à la re-configuration d'une interface graphique.

### Remerciements

Les auteurs remercient Laurence Duchien pour le temps passé à discuter de la rédaction de cet article.

## Références

- [1] R. Marvie, P. Merle *CORBA Component Model : Discussion and Use with OpenCCM*, Rapport technique, LIFL, janvier 2001.
- [2] L. Bellissard, SB. Atallah, F. Boyer and M. Riveill *Distributed Application Configuration*
- [3] L. Bellissard, SB. Atallah, A. Kerbrat and M. Riveill *Component-based Programming and Application Management with Olan*
- [4] R. Marvie *CODeX : proposition pour la description dynamique d'architectures à base de composants logiciels*, actes des journées composants, Besançon, octobre 2001.
- [5] OMG *Meta Object Facility (MOF) Specification v1.3*, Object Management Group, mars 2000.
- [6] X. Blanc *Echanges de spécifications Hétérogènes et Réparties*, thèse de doctorat, Laboratoire d'Informatique Paris 6, 2001.
- [7] P. Merle *CorbaScript-CorbaWeb : proposition pour l'accès à des objets et services distribués*, thèse de doctorat, Laboratoire d'Informatique Fondamentale de Lille, janvier 1997
- [8] J. Aldrich, C. Chambers, D. Notkin *Component-Oriented Programming in ArchJava*.
- [9] J. Aldrich, C. Chambers, D. Notkin *Architectural Reasoning in ArchJava*.