Un méta-modèle pour la description de contraintes architecturales sur l'évolution des composants

Chouki Tibermacine

Laboratoire VALORIA
Centre de recherche Yves Coppens
Université de Bretagne Sud
F-56000 Vannes Cedex
Chouki.Tibermacine@univ-ubs.fr

RÉSUMÉ. *Pour préserver certaines propriétés non-fonctionnelles d'applications à base de composants lors de leur évolution, une approche par contrats peut être envisagée. Ces contrats, dits d'évolution, comportent un certain nombre de contraintes architecturales. Pour la description de ces contraintes, nous utilisons OCL et un méta-modèle de composants. Afin de définir le contrat d'évolution tout au long du cycle de vie d'une application, le méta-modèle doit abstraire aussi bien les modèles architecturaux d'analyse/conception introduits par les ADL, que les modèles d'implémentation proposés par les technologies de composants. Dans ce papier, nous présentons ce méta-modèle générique pour la description de contrats d'évolution.*

ABSTRACT. *In order to preserve certain non-functional properties during the evolution of component-based applications, a contractual approach can be envisaged. These evolution contracts consist of a set of architectural constraints. To describe the architectural constraints, we use OCL (Object Constraint Language) and a component meta-model. In order to define the evolution contract during all the application lifecycle, the meta-model should be able to abstract, the architectural models introduced by ADLs in the Analysis/Design stage, as well as implementation models proposed by component technologies. In this paper, we present this generic meta-model for describing evolution contracts.*

MOTS-CLÉS : *Méta-modèle, ADL, Composants, Contrat d'évolution, Contraintes architecturales*

KEYWORDS: *Metamodel, ADL, Components, Evolution Contract, Architectural Constraints*

... - ... – 03/2005. SE'05, pages 1 à 11

1. Introduction

Dans [FLE 05], nous avons proposé la notion de contrat d'évolution. Le rôle du contrat est de garantir le maintien des choix architecturaux, et par conséquent les attributs qualité requis dans les documents de spécification. Il englobe un certain nombre de contraintes architecturales et leurs liens avec les attributs qualité. L'un des points importants de notre approche est de pouvoir décrire des contraintes architecturales tout au long du cycle de vie du logiciel. Le contrat d'évolution doit, par conséquent, fournir un langage d'expression portant à la fois sur des modèles de conception et sur des modèles d'implémentation. Par modèle de conception, nous voulons dire infrastructure abstraite du logiciel, ou en d'autres termes, un modèle fourni par un langage de description d'architecture (ADL). Il existe en pratique une multitude d'ADL [MED 00]. Chacun introduit un modèle architectural particulier. Acme [GAR 00] constitue une tentative d'unification des langages de description d'architecture[1]. Par modèle d'implémentation, nous entendons infrastructure concrète d'un logiciel, soit un modèle dépendant d'une technologie particulière. Les technologies de composants, tels que EJB [SUN 03], COM+/.net [MIC 05] ou CCM [OMG 02], représentent des exemples de modèles d'implémentation. Chacune introduit un modèle architectural différent. Cependant, CCM tente de fournir un modèle unifié et un standard pour les technologies de composants. Malgré les tentatives d'unification intra-niveaux (conception ou implémentation), nous remarquons toujours un fossé entre les modèles inter-niveaux (conception et implémentation).

La description du contrat d'évolution repose sur un certain méta-modèle de composants. Pour pouvoir décrire des contraintes tout au long du cycle de vie du logiciel, ce méta-modèle doit être suffisamment générique pour englober, aussi bien les modèles de conception que les modèles d'implémentation. Il doit représenter toutes les abstractions présentes dans les modèles de conception ainsi que celles des modèles d'implémentation. Dans cet article, nous présentons ce méta-modèle générique pour la description de contraintes architecturales.

Dans la section 2, nous exposons l'étude faite sur les ADL et les technologies de composants. Cette étude nous a permis d'élaborer un certain nombre de méta-modèles. Puis, dans la section 3, nous présentons le méta-modèle que nous avons pu déduire après analyse des méta-modèle de la section 2. A travers quelques exemples, nous illustrons l'usage de ce méta-modèle dans le contexte du contrat d'évolution. Avant de conclure, dans la section 4, nous comparons notre méta-modèle de composants et celui d'UML2.0 [OMG 03b].

1. Ce langage a donné naissance à ADML (Architecture Description Markup Language), standard adopté par *The Open Group* [TOG 02].

2. Etat de l'art

Nous avons bâti notre méta-modèle sur la base de deux types de méta-modèles. Le premier concerne les langages de description d'architecture et le second, les technologies de composants. Une étude sur les ADL Aesop [GAR 94], UniCon [SHA 95], Rapide [LUC 95], Darwin [MAG 96], WRIGHT [ALL 97],
SADL [MOR 95], C2SADEL [MED 99], Acme [GAR 00] et Mae [ROS 04] a été menée. Cet état de l'art s'est concrétisé par l'élaboration d'un méta-modèle qui abstrait chaque modèle architectural décrit par ces ADL. Ces méta-modèles prennent en compte l'aspect statique des architectures modélisées avec ces langages. Ils représentent donc la manière dont ces langages décrivent structurellement des architectures logicielles. Il est à noter que l'aspect comportemental n'est pas pris en compte par ces méta-modèles. La deuxième catégorie de méta-modèles a été construite pour les technologies de composants : EJB (*Enterprise JavaBeans*) de Sun Microsystems [SUN 03] et CCM (*CORBA Component Model*) de l'OMG [OMG 02]. Nous avons jugé intéressant d'étendre notre étude sur le modèle de composants Fractal [BRU 04] et de définir un méta-modèle pour ce dernier[2].

Dans une première étape, il nous a fallu élaborer un méta-modèle pour chaque niveau (conception ou implémentation). Cette tâche était relativement simple dès lors que des standards dans les deux niveaux ont émergé. Nous voulons dire, par standard, une norme de fait, comme CCM dans les technologies de composants et Acme (par son extension XML, ADML) pour les ADL.

Nous nous sommes fixés deux objectifs afin de définir le méta-modèle générique à partir de ces deux types de méta-modèles :

– Le méta-modèle doit abstraire tous les concepts présents, à la fois, dans les infrastructures abstraites et les infrastructures concrètes des logiciels à base de composants. L'intersection des abstractions représentées par les différents méta-modèles fournit un méta-modèle minimal qui ne prend pas en compte toutes les propriétés et concepts architecturaux.

– Le méta-modèle doit être simple. Le développeur doit pouvoir définir son contrat d'évolution aisément et rapidement. Les contraintes à spécifier devraient être décrites en naviguant dans un méta-modèle contenant des concepts connus par le développeur. L'union des abstractions présentes dans les différents méta-modèles aura comme résultat un méta-modèle assez large et complexe ; ce qui ne répond pas à la contrainte de simplicité.

Le méta-modèle est présenté dans la prochaine section. Concentrons nous, maintenant, sur les abstractions architecturales représentées dans les ADL. Nous discuterons des méta-modèles pour les technologies de composants dans la sous-section suivante.

2. Faute de place, nous ne pouvons présenter, dans cet article, ces différents méta-modèles.

2.1. *Abstractions représentées par les ADL*

Un ADL doit, en principe, pouvoir décrire une architecture logicielle sous la forme des trois C : les Composants, les Connecteurs et les Configurations [MED 00]. Les composants représentent les unités de calcul et de stockage de données dans un système logiciel. L'interaction entre ces composants est encapsulée par les connecteurs. Dans la configuration, l'architecte instancie un nombre de composants et un nombre de connecteurs. Il les lie entre eux afin de former le système complet. Une partie des ADL répond à cette description. Cependant, une autre partie diverge. Certains ADL comme Rapide ou Darwin, par exemple, n'explicitent pas la notion de connecteur. D'autres permettent la description hiérarchique des composants. Les composants peuvent être perçus comme des boîtes blanches et peuvent donc avoir une structure interne explicite. Dans certains ADL tels que Rapide, les composants sont considérés plutôt comme des boîtes noires. Dans UniCon, WRIGHT ou encore Acme, nous pouvons définir des connecteurs composites, alors que cela est impossible dans les autres ADL.

Il est à noter que la majorité des ADL visent des objectifs particuliers et répondent aux préoccupations de leurs concepteurs. Darwin vise la reconfiguration dynamique des architectures. Rapide a pour objectif la description architecturale de systèmes événementiels. SADL se focalise sur le raffinement des architectures. C2SADEL modélise des architectures dans le style C2, etc. Cependant, il existe quelques ADL, comme Acme ou WRIGHT, à objectif général. En effet, le méta-modèle de WRIGHT reste générique et peut se projeter sur plusieurs autres ADL, malgré son objectif de formaliser les architectures logicielles. Il est donc tout à fait justifié de prendre ce méta-modèle comme référence pour les ADL. Cependant, ce dernier contient des méta-classes qui ne nous servent pas pour la définition des contrats d'évolution. Par exemple, WRIGHT, dans une description d'architecture, distingue les types des composants (ou connecteurs) de leurs instances. Cette distinction et l'existence d'une méta-classe `ComponentInstance` (ou `ConnectorInstance`) ne nous intéresse pas pour décrire les contraintes architecturales. En effet, une occurrence particulière d'un composant (ou connecteur) peut être identifiée par son nom utilisant seulement une méta-classe `Component` (ou `Connector`) avec un attribut `name`. Les mêmes remarques peuvent être faites pour Acme ainsi que pour ses extensions ADML ou xAcme [CMU01]. Le méta-modèle résultant pour les ADL constitue donc une simplification des méta-modèles de WRIGHT ou Acme.

2.2. *Abstractions représentées dans les technologies de composants*

Les technologies de composants fournissent, à la fois, un modèle abstrait de développement d'applications à base de composants et une infrastructure concrète pour leur déploiement. Ces infrastructures fournissent l'environnement nécessaire pour l'exécution de ces applications, en terme de services transactionnels, de sécurité, de répartition, etc. Les modèles abstraits définis par ces technologies permettent de définir une

application sous la forme de composants fournissant un certain nombre de services (interfaces) et explicitant leurs dépendances avec leur environnement.

Sur la base des profils UML[3] pour EJB [RSC 01] et CCM [OMG 04], nous avons établi nos propres méta-modèles pour ces deux technologies de composants. Nous avons défini nos propres méta-modèles dès lors que les profils UML ne constituent pas la solution idéale. Dans la section 4, nous discutons ce point plus en détail.

Nous remarquons sur ces deux méta-modèles que le méta-modèle CCM englobe celui d'EJB. En d'autres termes, un modèle EJB peut facilement se projeter dans CCM. Nous pouvons déduire de cela que le méta-modèle CCM peut jouer le rôle du méta-modèle générique pour les composants d'implémentation. Par contre, CCM est un modèle de composants plats. Donc, il est impossible de décrire des composants hiérarchiques. Il existe, cependant, certains modèles de composants d'implémentation hiérarchiques, comme par exemple Fractal. Ce dernier permet de définir des composants composites ayant une structure interne explicite.

Le méta-modèle résultant de cette étude abstrait des implémentations à base de composants comme ceux de CCM, mais hiérarchiques comme en Fractal.

3. Proposition d'un méta-modèle générique

Dans cette section, nous décrivons, en premier lieu, le méta-modèle générique que nous proposons. En second lieu, nous illustrons, à travers quelques exemples, l'utilisation de ce méta-modèle.

3.1. *Description du méta-modèle*

La figure 1 représente le méta-modèle générique décrit en MOF [OMG 03a]. Il définit les abstractions architecturales communes aux ADL et aux technologies de composants. Un système est décrit par un certain nombre de composants qui représentent les unités de calcul et de données. Le mode de communication et de coordination entre ces composants est encapsulé par des connecteurs. La configuration du système est représentée par un assemblage de composants par le biais de connecteurs. Ce méta-modèle met en avant les concepts suivants :

– Un composant définit un ou plusieurs ports. Ces ports représentent le point d'interaction du composant avec son environnement. Un port peut supporter aucune ou plusieurs interfaces requises par le composant et une ou plusieurs interfaces fournies.

3. Un profil UML est une extension bien formée du méta-modèle UML. Par extension bien formée, nous voulons dire une extension conforme aux spécifications du méta-modèle UML. Plus concrètement, c'est une extension du méta-modèle UML basée sur les stéréotypes, les valeurs marquées (*Tagged Values*) et les contraintes.

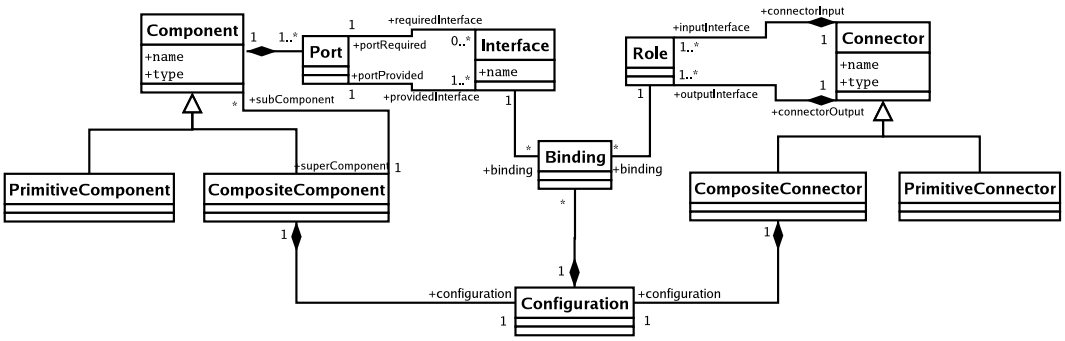

Figure 1 – Le méta-modèle générique

– Un connecteur définit au minimum deux rôles joués par les composants en interaction. L'un des rôles définit son interface d'entrée et l'autre, son interface de sortie. Un connecteur peut être vide. Il représente donc un simple lien entre deux composants (*binding* en Fractal).

– Une configuration représente plusieurs liens (*bindings*). Ces liens sont définis entre les interfaces de composants (requises ou fournies) et les rôles de connecteurs (en entrée ou en sortie). Il est évident que seules les interfaces requises peuvent être reliés aux rôles en entrée et les interfaces fournies aux rôles en sortie[4]. Il est à noter que dans ce méta-modèle (voir figure 1), une interface ou un rôle peut participer dans plusieurs liens.

– Une description hiérarchique associe à un composant ou à un connecteur composites une structure interne. Cette structure interne est décrite à l'aide d'une configuration de sous-composants et connecteurs. Nous supposons que les délégations effectuées des ports des composants aux ports de leurs sous composants sont réalisées à l'aide de connecteurs, dits de délégation. De cette manière, aucun lien direct entre les interfaces des ports de composants et les interfaces de leurs sous-composants n'est autorisé. Ceci garantit la généricité du méta-modèle.

3.2. *Exemples d'illustration*

Dans cette section, nous présentons deux exemples d'utilisation du méta-modèle pour décrire des contraintes architecturales. La première contrainte décrit la configuration à maintenir pour préserver le pattern architectural *pipeline* [SHA 96]. Un pipeline est un style architectural décrivant les quatre contraintes architecturales suivantes :

– Un composant (filtre) ne peut posséder plus d'une interface requise et une interface fournie,

4. Ceci n'est pas explicité dans le méta-modèle, parce que l'objectif de ce dernier n'est pas de vérifier la validité d'une architecture.

– Un connecteur (pipe) unique doit relier deux composants,

– Tous les connecteurs sont orientés dans le même sens,

– L'ensemble des connecteurs ne doit pas former un circuit.

Dans le contrat d'évolution d'un composant (`nomComposant`) dont la structure interne est organisée comme un pipeline, la contrainte suivante doit être définie :

```
context nomComposant : CompositeComponent inv :
(self.configuration.binding.interface.requiredPort.component =
 (self.subComponent))
and
(self.configuration.binding.interface.providedPort.component =
 (self.subComponent))
```

La première partie de cette contrainte signifie que tout sous-composant doit avoir un et un seul lien via son interface requise. La seconde fait de même pour les interfaces fournies. Ces deux contraintes garantissent ensemble la structure en pipeline de la configuration interne de `nomComposant`.

Le second exemple illustre la contrainte suivante :

Aucun connecteur ne doit traverser la structure interne d'un composant.

Cette contrainte peut être décrite, de la manière suivante, sur le méta-modèle :

```
context nomComposant : CompositeComponent inv :
self.configuration.binding
–>forAll(b| self.subComponent.interface –>includes(b.interface))
```

La contrainte ci-dessus stipule le fait que, pour tous les liens qui constituent la configuration du composant composite, chaque interface d'un port d'un des liens doit appartenir à l'un des sous-composants du composite. Celle-ci est valide pour une profondeur d'un seul niveau. En réalité, la contrainte doit être appliquée récursivement sur tous les sous-composants composites de `nomComposant` et leurs sous-composants composites pour une description hiérarchique beaucoup plus profonde.

4. Comparaison avec le méta-modèle UML 2.0

UML fournit une notation unifiée pour la modélisation et un standard supporté par plusieurs outils du génie logiciel. Dans sa spécification 2.0 [OMG 03b] (adoptée par l'OMG), il fournit les éléments nécessaires pour décrire à la fois des architectures abstraites et des implémentations à base de composants[5]. Un composant est considéré comme un élément de modélisation à part entière (tout comme une classe) et non pas comme une entité déployable et existante à l'exécution. Cependant, UML est un

5. Chapter 8 : Components. Spécification de la super-structure UML 2.0 [OMG 03b], page 133.

langage de modélisation à objectif général. Par conséquent, il existe une multitude de concepts présents dans son méta-modèle qui rendent ce dernier relativement complexe. Notre objectif (comme énoncé précédemment) est de pouvoir définir le contrat d'évolution le plus facilement et rapidement possible. Pour ce faire, le méta-modèle doit comporter des abstractions simples et connues par le développeur.

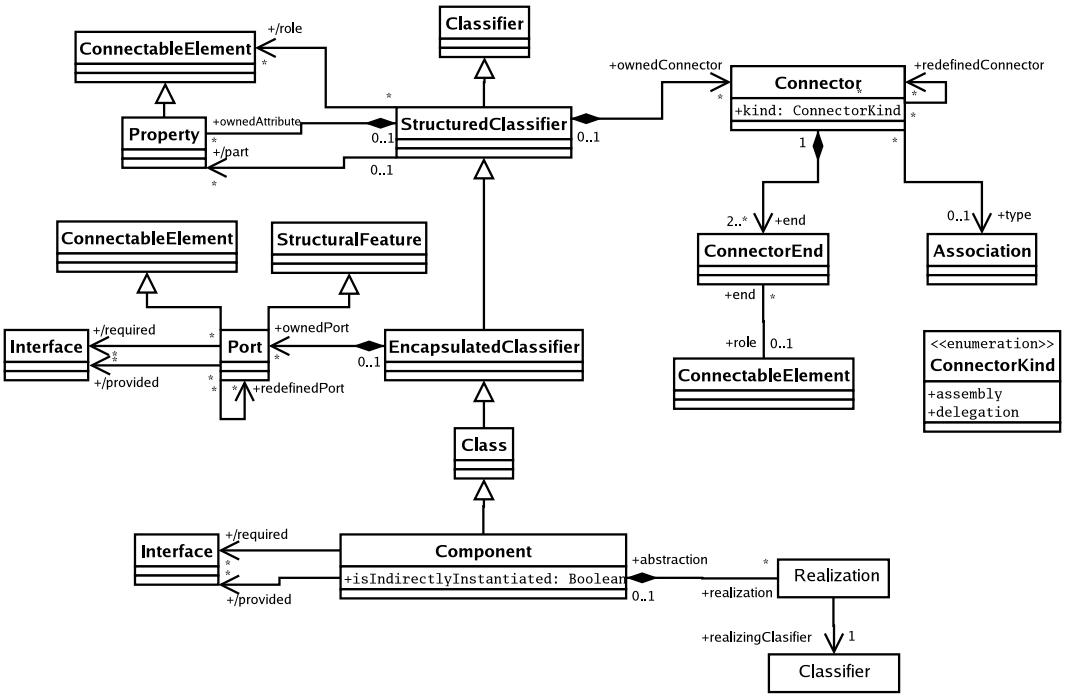

Figure 2 – Le méta-modèle de composants UML2.0

La figure 2 présente un extrait du méta-modèle UML2.0 pour la description de composants. Un composant hérite de la méta-classe `Class`. Il peut donc définir des opérations et participer dans des généralisations. Il hérite également de `EncapsulatedClassifier`. Il peut donc posséder des ports, typés par des interfaces fournies et requises. La méta-classe `EncapsulatedClassifier` hérite de `StructuredClassifier`. Par conséquent, un composant peut avoir une structure interne et peut définir des connecteurs.

Pour la description de contrats architecturaux, les méta-classes `EncapsulatedClassifier`, `StructuredClassifier` comme d'autres ne sont pas nécessaires. Dans ce cas, du refactoring du méta-modèle UML et d'une élimination de certaines méta-classes peut résulter un méta-modèle simple à utiliser pour décrire des contraintes architecturales. La même remarque concerne les profils UML pour la description des architectures logicielles [KAN 00] ou ceux pour EJB et CCM. Cependant, il est tout à fait envisageable de décrire des contraintes utilisant notre méta-modèle, évaluables,

par la suite, sur des architectures modélisées en UML2.0 ou sur les profils précédemment cités.

5. Conclusion

Dans cet article, nous avons présenté un méta-modèle générique pour la description de contraintes architecturales. Ce méta-modèle est générique dans le sens où il abstrait des modèles d'architectures au niveau conception et au niveau implémentation. Ceci nous permet de définir des contraintes tout au long du cycle de vie des applications à base de composants. Nous avons mené une étude sur les modèles d'architecture introduits par les langages de description d'architectures. Cette étude s'est concrétisée par l'élaboration d'un méta-modèle pour chaque modèle architectural. Un autre état de l'art a été effectué. Il consistait à établir des méta-modèles pour les technologies de composants. Par la suite, le méta-modèle générique a été défini à partir de ces différents méta-modèles.

L'apport de cet article et du méta-modèle proposé se situe dans un contexte de description de contraintes architecturales. Nous pouvons, également, placer le méta-modèle dans un contexte de transformation de modèles (MDA [OMG 01]). Ce méta-modèle peut jouer le rôle d'un méta-modèle pivot dans la transformation des descriptions d'architectures d'un ADL à un autre (*PIM* à *PIM*), d'un ADL à une technologie de composants (*PIM* à *PSM*) ou d'une technologie de composants à une autre (*PSM* à *PSM*). Par contre, ceci se limite à l'aspect structurel de la description des architectures.

Nous avons développé un outil pour l'évaluation des contraintes architecturales, ACE[6] (*Architectural Contract Evaluator*). Cet outil effectue, actuellement, des introspections sur des architectures Fractal. Nous projetons, dans un avenir proche, d'étendre l'outil pour effectuer des évaluations sur des architectures modélisées en UML2.0. A moyen terme, nous envisageons de restructurer ACE afin qu'il évalue des architectures définies dans un modèle pivot unique. Ce modèle intermédiaire est l'instance exacte de notre méta-modèle. Des extensions d'ACE permettant d'introspecter des architectures définies dans un nouvel ADL ou un nouveau langage de configuration d'une technologie de composants, consistent simplement à effectuer une transformation de l'architecture dans le modèle pivot. A long terme, nous projetons introduire ACE dans un atelier de génie logiciel (AGL).

6. Bibliographie

[ALL 97] ALLEN R., « A Formal Approach to Software Architecture », PhD thesis, Carnegie Mellon University, Pittsburgh, PA, USA, mai1997.

[BRU 04] BRUNETON E., COUPAYE T., STEFANI J., « The Fractal Component Model Specification. Final Release, Version 2.0-3 », http ://fractal.objectweb.org/specification/, 2004.

6. http ://www-valoria.univ-ubs.fr/Composants/se/current/Cell/ace.html

[CMU01] « xAcme : Acme Extensions to xArch », CMU Web Site : http ://www-2.cs.cmu.edu/ acme/pub/xAcme/, 2001.

[FLE 05] FLEURQUIN R., TIBERMACINE C., SADOU S., « Le contrat d'évolution d'architectures : un outil pour le maintien de propriétés non fonctionnelles », *To appear in the proceedings of LMO'05 Conference (Langages et Modèles à Objets)*, Bern, Switzerland, mars2005.

[GAR 94] GARLAN D., ALLEN R., OCKERBLOOM J., « Exploiting Style in Architectural Design Environments », *In Proceedings of the ACM SIGSOFT Symposium on the Foundations of Software Engineering*, New Orleans, Louisiana, USA, 1994, p. 175–188.

[GAR 00] GARLAN D., MONROE R. T., WILE D., « Acme : Architectural Description of Component-Based Systems », LEAVENS G. T., SITARAMAN M., Eds., *Foundations of Component-Based Systems*, p. 47-68, Cambridge University Press, 2000.

[KAN 00] KANDÉ M. M., STROHMEIER A., « Towards a UML Profile for Software Architecture Descriptions », *Proceedings of UML'2000 - The Third International Conference on the Unified Modeling Language : Advancing the Standard* -, York, United Kingdom, octobre2000.

[LUC 95] LUCKHAM D. C., KENNEY J. L., AUGUSTIN L. M., VERA J., BRYAN D., MANN W., « Specification and Analysis of System Architecture Using Rapide », *IEEE Transactions on Software Engineering*, vol. 21, n° 4, 1995, p. 336–355.

[MAG 96] MAGEE J., KRAMER J., « Dynamic Structure in Software Architectures », *In Proceedings of the fourth SIGSOFT Symposium on the Foundations of Software Engineering, San Francisco, USA*, 1996.

[MED 99] MEDVIDOVIC N., « Architecture-Based Specification-Time Software Evolution », PhD thesis, University of California, Irvine, 1999.

[MED 00] MEDVIDOVIC N., TAYLOR N. R., « A Classification and Comparison Framework for Software Architecture Description Languages », *IEEE Transactions on Software Engineering*, vol. 26, n° 1, 2000.

[MIC 05] MICROSOFT, « COM : Component Object Model Technologies », http ://www.microsoft.com/com/, 2005.

[MOR 95] MORICONI M., QIAN X., RIEMENSCHNEIDER R. A., « Correct Architecture Refinement », *IEEE Transactions on Software Engineering*, vol. 21, n° 4, 1995, p. 356–372.

[OMG 01] OMG, « Model Driven Architecture (MDA), A Technical Perspective, Document ormsc/01-07-01 », Object Management Group Web Site : http ://www.omg.org/docs/ormsc/01-07-01.pdf, juillet2001.

[OMG 02] OMG, « CORBA Components, v3.0, Adpoted Specification, Document formal/2002-06-65 », Object Management Group Web Site : http ://www.omg.org/docs/formal/02-06-65.pdf, juin2002.

[OMG 03a] OMG, « Meta Object Facility (MOF) 2.0 Final Adopted Specification, Document ptc/03-10-04 », Object Management Group Web Site : http ://www.omg.org/docs/ptc/03-10-04.pdf, 2003.

[OMG 03b] OMG, « UML 2.0 Superstructure Final Adopted Specification, Document ptc/03-08-02 », Object Management Group Web Site : http ://www.omg.org/docs/ptc/03-08-02.pdf, 2003.

[OMG 04] OMG, « UML Profile for Corba Components Final Adopted Specification, Document ptc/04-03-04 », Object Management Group Web Site :

http ://www.omg.org/docs/ptc/04-03-04.pdf, mars2004.

[ROS 04] ROSHANDEL R., VAN DER HOEK A., MIKIC-RAKIC M., N. M., « Mae - A System Model and Environment for Managing Architectural Evolution », *ACM Transactions on Software Engineering and Methodology*, vol. 11, n° 2, 2004, p. 240–276.

[RSC 01] RSC, « Rational Software Corporation. UML/EJB (TM) Mapping Specification », Java Community Process, JSR-000026. http ://www.jcp.org/aboutJava/communityprocess/review/jsr026/, juin2001.

[SHA 95] SHAW M., DELINE R., KLEIN D. V., ROSS T. L., YOUNG D. M., ZELESNIK G., « Abstractions for Software Architecture and Tools to Support Them », *IEEE Transactions on Software Engineering*, vol. 21, n° 4, 1995, p. 314–335.

[SHA 96] SHAW M., GARLAN D., *Software Architecture : Perspectives on an Emerging Discipline*, Prentice Hall, 1996.

[SUN 03] SUN-MICROSYSTEMS, « Enterprise JavaBeans(TM) Specification, version 2.1 », http ://java.sun.com/products/ejb, novembre2003.

[TOG 02] TOG, « The Open Group Consortium. The Architecture Description Markup Language Home Page », http ://www.opengroup.org/architecture/adml/adml_home.htm, 2002.