CARI'10

Cohérence de vues dans la spécification des Architectures Logicielles

Georges Edouard KOUAMOU

Equipe ALOCO-LIRIMA

Département de Génie Informatique
Ecole Nationale Supérieure Polytechnique
BP 8390 Yaoundé
CAMEROUN

georges_edouard@yahoo.com

RÉSUMÉ. Cet article s'intéresse à la spécification des architectures logicielles. Il présente une symbiose entre l'approche conceptuelle basée sur les profils UML et la vision opérationnelle prônée par ArchJava. Actuellement, chaque langage de description se situe à une extrémité du processus, engendrant ainsi un découplage entre la spécification des Architectures Logicielles et leur implémentation et un risque d'incohérence. Nous décrivons une démarche basée sur un profil UML pour la description structurale des architectures logicielles et des règles de transformation pour générer le code source. Les expérimentations actuelles sont probantes et nous espérons poursuivre la réflexion sur les configurations et l'aspect dynamique.

ABSTRACT. This study concerns the specification of software architecture. Its presents a symbiosis between the conceptual approach based on UML profiles and the programming approach recommended by ArchJava. Currently, each ADL addresses separately the specification and the validation, thus we notice a decoupling between the two descriptions and a risk of inconsistency. We describe a step based on a UML profiles for the structural description of software architectures and the transformation rules to generate the source code. The current experiments are convincing and we hope ourselves to continue the reflexion on the configurations and the dynamic aspect.

MOTS-CLÉS : .ADL, profil UML, ArchJava, MDA

KEYWORDS: .ADL, UML profile, ArchJava, MDA.

1 Introduction

Les architectures logicielles, durant la dernière décennie, ont contribué à maîtriser la complexité sans cesse grandissante des systèmes actuels, qui sont volumineux et distribués. Les architectures logicielles favorisent une meilleure conception de ces systèmes selon une approche diviser-pour-régner en favorisant la modularité et la réutilisation des composants logiciels. Le principe directeur consiste à raisonner à un niveau d'abstraction élevé, ce qui facilite la compréhension, la maintenance ainsi que l'évolution des systèmes [1]. Egalement, l'élaboration d'une bonne architecture du logiciel peut contribuer à exhiber les propriétés cruciales d'un système telles que sa fiabilité, sa portabilité ainsi que l'interopérabilité tandis qu'au contraire, une mauvaise architecture peut avoir des conséquences désastreuses sur le système.

Considérant ce rôle central joué par les architectures logicielles, le développement des langages pour leur description de façon formelle est devenu une nécessité. Ainsi, les langages de description d'architectures (*Architecture Description Language* en abrégé ADL) sont apparus afin de favoriser une meilleure conception des architectures logicielles, de faciliter leur compréhension et fournir des outils formels pour leur analyse. Cependant, l'on constate que ces langages n'ont pas connu un grand essor industriel chez les constructeurs de logiciels pour deux raisons principales [2] :

- Ils offrent des méthodes de modélisation et des techniques d'analyse très spécialisées. Ce qui limite leur utilisation uniquement aux utilisateurs expérimentés qui disposent des connaissances dans le domaine des spécifications formelles.
- Il existe un découplage entre la description architecturale d'un système et son implantation. Ce qui peut engendrer des incohérences et ne permet pas de vérifier la non violation des contraintes architecturales dans l'implémentation.

Ce découplage entre la spécification et l'implantation des architectures logicielles a été à l'origine du langage ArchJava qui, contrairement aux ADLs, étend le langage de programmation Java en y incluant les concepts architecturaux pour unifier la description architecturale et son implantation, afin de garantir que l'implémentation faite sera conforme aux contraintes architecturales définies. Toutefois ArchJava ne permet pas de représenter des modèles conceptuels, il reste encore trop proche du code.

Cette étude a pour objectif de concilier ces deux approches de solution pour établir une symbiose entre l'approche conceptuelle proposée par UML2.0 et l'approche par programmation d'ArchJava afin de proposer une démarche permettant de guider et d'assister les activités de conception d'architectures logicielles jusqu'à leur implantation. Bien que UML, à partir de sa version 2, dispose du diagramme composite qui permet de représenter la structure des architectures logicielles, elle n'offre pas le support de raffinement capable de générer le code de l'application. Notre intention est de construire un mécanisme de transformation du modèle de composant UML vers un

langage cible afin garantir la consistance et la traçabilité entre la description conceptuelle de l'architecture et son implantation.

Dans la suite de cet article, nous allons présenter dans la section 2 les éléments de base qu'offre tout langage de description des architectures logicielles, nous illustrerons la cas de UML et ArchJava, afin de cerner les complémentarités. Dans la section 3, nous présentons la démarche de transformation d'une spécification UML en ArchJava qui s'appuie sur l'approche MDA. Nous terminerons par une discussion qui précise la position de cette recherche par rapport à l'état de la technologie

2 Description et transformation des architectures logicielles

Les travaux autour des formalismes de description d'architectures ont émergé pendant la décennie 1990. Ils ont abouti à la naissance des ADL. Ils apparaissent ainsi comme une solution pour passer de la programmation à petite échelle (ligne d'instructions) qu'offraient les langages à objets à une programmation à grand échelle (niveau d'abstraction élevé). Chacun des ADL a ses spécificités. Les uns privilégient les concepts architecturaux ainsi que leur assemblage structurel tandis que les autres s'orientent vers la configuration de l'architecture ainsi que la dynamique du système, en ayant tous pour objectif de fournir une abstraction de haut niveau plutôt que l'implantation dans un code source spécifique. Les propriétés des ADL peuvent alors être regroupées selon trois catégories [4]:

- Les exigences minimales fondamentales : il s'agit de la spécification des concepts architecturaux (Composants, Connecteurs et Configuration). Ces propriétés doivent être offertes par tous les ADLs.
- Les exigences souhaitables : il s'agit d'un ensemble de propriétés dont le support permet l'implantation des architectures correctes telles que la spécification des contraintes, la modélisation de la sémantique, la composition hiérarchique et la spécification des propriétés fonctionnelles.
- Les exigences désirables mais non fondamentales telles que le support de la réutilisation, l'évolution et le support d'outils.

Dans leur grande majorité, les langages de Description des Architectures logicielles (ADL) permettent aux architectes d'exprimer une vue abstraite des logiciels en mettant à disposition des éditeurs pour la représentation, des outils d'analyse et de vérification. Concernant le raffinement qui devrait permettre le passage d'un niveau abstrait vers des niveaux plus concrets devant déboucher sur une implémentation, ce processus n'est pas suffisamment voire pas du tout supporté par les outils, obligeant ainsi les architectes à effectuer la transformation manuellement [11][12]. Ainsi, les outils de construction des

diagrammes UML facilitent la génération du code quand il s'agit des classes, mais rien n'est fait concernant les composants qui est l'élément majeur introduit dans UML pour spécifier les architectures logicielles.

2.1 Le langage de modélisation unifié (UML)

La description architecturale doit être compréhensible et manipulable par tous les acteurs intervenant dans le processus de développement. Les travaux ont été effectués autour d'UML pour prendre en compte la description des architectures logicielles à composants pour plusieurs raisons : c'est un standard de fait par la participation des grands constructeurs de logiciels à son développement, il est utilisé à grande échelle tant dans le monde des entreprises que celui de la recherche et l'on dispose d'un grand nombre d'outils CASE.

Les qualités de UML en tant que ADL sont contenues dans sa version 2.0 [5], [6], où apparaît un modèle de composant nettement amélioré, qui fournit les concepts et mécanismes pour décrire les architectures du logiciel à savoir «composant», «connecteur» et «configuration» [7] [8].

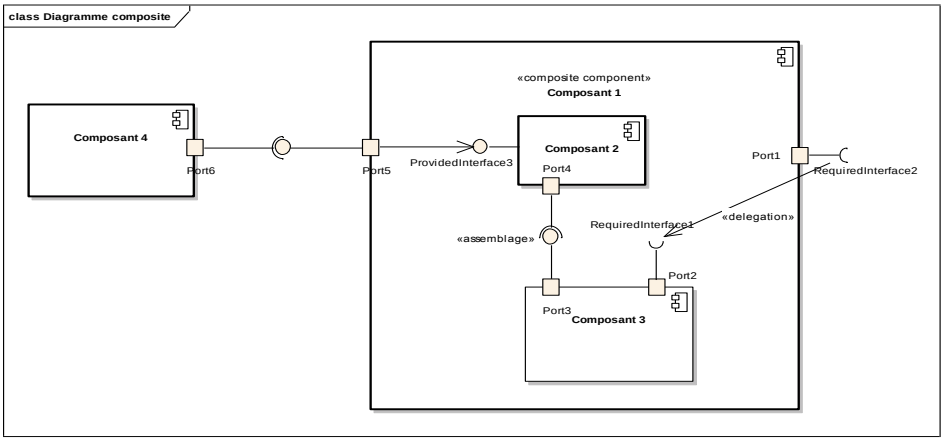

Figure 1. *Modèle de composant UML2.0*

D'après la spécification de l'OMG, un composant (Component), qui peut être atomique ou composite, est une unité instanciable qui interagit avec son environnement par l'intermédiaire des ports. Les ports sont typés par des interfaces renfermant un certain nombre de services fournis et requis. Les interactions entre composants sont définies par des connecteurs (*Connector*) pouvant être de deux types : les connecteurs d'assemblage (*AssemblyConnector*) qui permet d'assembler deux instances de composant en connectant un port fourni d'un composant au port requis de l'autre

composant, et le connecteur de délégation (*DelegationConnector*) qui permet de connecter un port externe au port d'un sous-composant interne de même type. Ce type de connecteur permet simplement de transférer une requête vers une autre interface.

2.2 Architecture for Java (ArchJava)

Comme la plupart des ADLs, UML offre une vision abstraite de la spécification des architectures logicielles, laissant le soin à l'architecte d'introduire les détails lors de son implémentation dans un langage de programmation cible. Ce découplage entre la spécification et l'implémentation des architectures logicielles est la motivation de la mise en œuvre du langage ArchJava. Ce langage de programmation se situe à la frontière des architectures logicielles et des langages à composants. Il étend le langage de programmation Java en ajoutant des concepts architecturaux dans le but de garantir la consistance entre la description architecturale et son implantation [9] afin d'offrir une vision opérationnelle des architectures logicielles. Très souvent, le programme est généré à partir d'une conception représentée selon un formalisme, à l'exemple du diagramme de classes qui permet de générer le code source dans un langage à objets.

ArchJava reprend la définition des trois concepts architecturaux de base comme éléments de structure du code de l'application. Il ajoute de nouveaux éléments de syntaxe (*Component class, connect, pattern, connect with, port, requires, provides*, etc.) au langage Java pour gérer les composants, les connecteurs et les ports.

```
public component class Webserver {
 private final Client  Browser = new  Client ( ) ;
 private final Server  ApplicationServer = new  Server( ) ;

 connect  Browser.In , ApplicationServer.Out;
  public static void main(String args[]) {
    System.out.println("ceci est juste un exemple");
  }
  }
   component class Client {
      public port In {
            requires  void Download(String url);
        }
   }
   component class Server {
      public port Out {
           provides  void Download(String url){/* TODO*/}
        }
   }
```

Figure 2 : *Architecture 2-tiers simplifiée en ArchJava*

Dans cet exemple, nous considérons une architecture 2-tiers simplifiée composée de trois composants dont deux primitifs (Client et Server) dotés chacun d'un port (respectivement *In* et *Out*) et un composite (*Webserver*) représentant la configuration ou

l'application elle-même. Une instance de Client (*Browser*) requiert un service de téléchargement fournit par une instance de serveur (*ApplicationServer*).

3 Du modèle conceptuel vers le modèle opérationnel

D'après la description précédente, nous constatons qu'il existe deux approches dans le développement des architectures logicielles. L'une formelle qui consiste à écrire des spécifications, à les analyser et les simuler. L'autre opérationnelle qui consiste à injecter des concepts architecturaux au niveau de la programmation. Dans cette contribution, nous voulons poser les bases d'une approche plus globale en conciliant les deux approches précédemment décrites.

La démarche que nous préconisons est inspirée de l'approche MDA (Model Driven Architecture), essor de l'Ingénierie Dirigée par les Modèles (IDM) autour d'UML. Cette démarche place le modèle au centre des architectures logicielles. Nos techniques correspondent aux deux artefacts de cette approche à savoir : modélisation et transformations de modèles [3]. Le principe clé de MDA étant la séparation des préoccupations à savoir la logique métier et la logique d'implémentation. Elle place le modèle dans toutes les étapes du processus de développement en suivant différents niveaux d'abstraction.

Pour rester conforme aux principes de MDA, nous avons adopté le profil UML de la Figure 3 qui définit l'abstraction des entités nécessaires pour la description des architectures logicielles [10]. Cette solution permet de raisonner au niveau des modèles, en étant indépendant d'une technologie ou d'une plateforme d'exécution particulière.

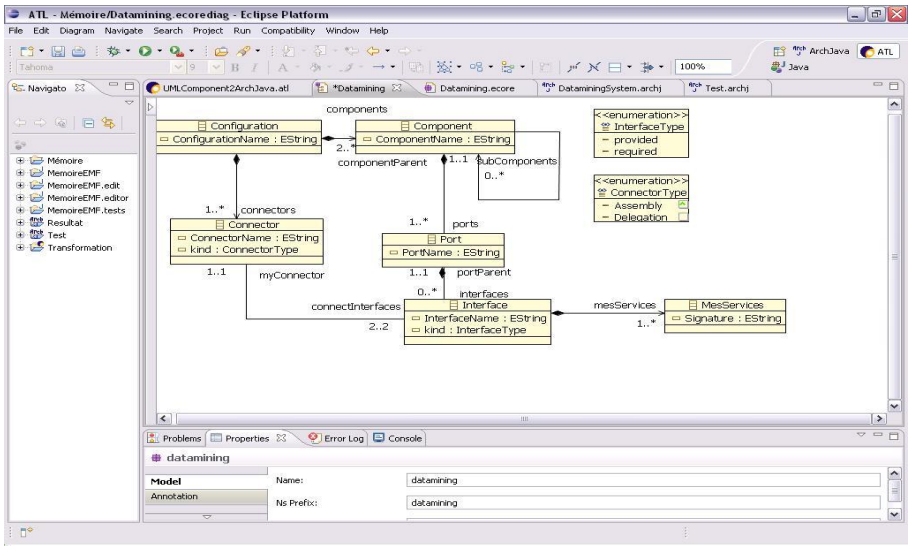

Figure 3 : *Profil de description architecturale avec Eclipse EMF*

Un *StructuralProfil* est composée par une *StructuralProfilName* indiquant son nom, une *Configuration* et un *Guards* exprimé par zéro ou plusieurs contraintes OCL. Une *Configuration* est composée par deux ou plusieurs *Components* et un ou plusieurs *Connectors* et possède un nom. Un *component* peut être composé par zéro ou plusieurs *Components* et il contient un ou plusieurs Ports.

Un *Connector* établit un lien entre deux *Components* via deux *Ports* et il peut être de type *Assembly* (assemblage) ou *Delegation*. Un *Port* doit avoir une interface de type requis et/ou fournis. L'*Interface* permet des échanges et interactions entre les composants. C'est par elle qu'un composant expose ses services ou en demande à l'environnement extérieur. Un *Component* et un *Connector* dans une architecture sont identifiés par des noms uniques. Le nom du Component est déterminé par la fonction *« ComponentName »* et celui du Connector est déterminé par la fonction *« ConnectorName ».* Un *Port* est identifié par un nom unique au sein d'un même *Component*. Au sein du même *Port*, une Interface est identifiée par un nom unique.

3.1 Démarche de modélisation

La démarche préconisée par cette approche est la transformation par modélisation qui comporte trois phases :

- La définition des règles de transformation qui est une mise en correspondance des concepts du méta-modèle source et ceux du méta-modèle cible en vue de la construction du méta-modèle de transformation.

- L'expression des règles de transformation qui consiste à définir dans un langage de transformation un modèle de transformation conforme au méta-modèle de transformation préalablement défini.

- L'exécution des règles de transformation qui consiste à exécuter le programme de transformation sur le modèle source pour produire le modèle cible.

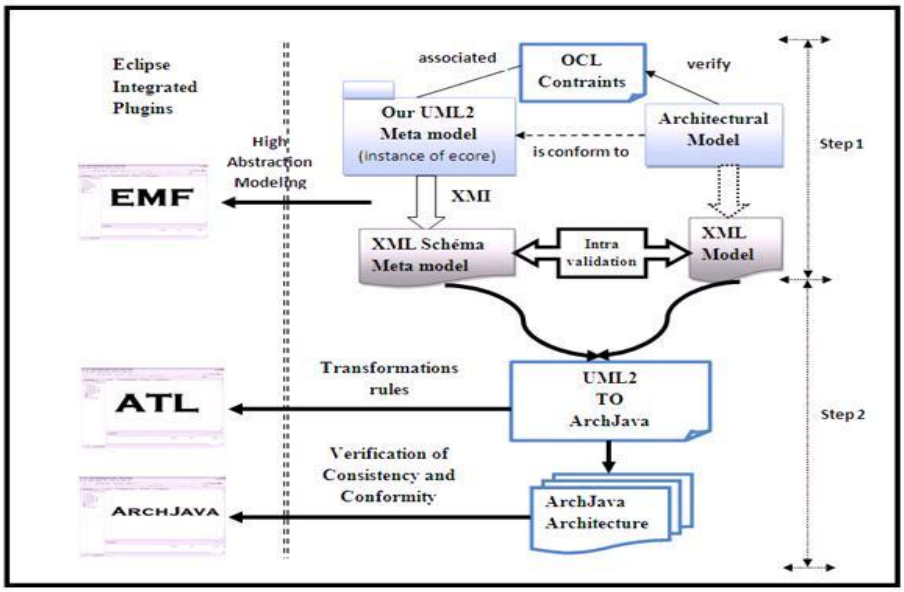

Figure 4 *: Architecture générale de notre démarche de modélisation*

3.2 Cadre d'expérimentation

Pour expérimenter notre démarche, nous avons utilisé la plateforme Eclipse en exploitant la possibilité d'extension grâce aux mécanismes de plugins. Ainsi la modélisation et la validation de notre approche ont été réalisées autour de trois plugins d'Eclipse : EMF, ATL et ArchJava.

- EMF (Eclipse Modeling Framework) est un Framework de modélisation et de génération de code destiné à la création d'outils et d'applications dirigée par les modèles [15]. Il intègre le standard XMI (XML Metadata Interchange) favorisant

l'interopérabilité des modèles ainsi qu'un langage de méta-modélisation Ecore pour l'édition et la vérification des méta-modèles.

- ATL (Atlas Transformation Language) [17] est un langage de transformation de modèles hybride (déclaratif et impératif). Il est inspiré du standard QVT de l'OMG et il est disponible en tant que plug-in dans le projet Eclipse.
- un plug-in intégrable dans Eclipse qui permet l'édition et la vérification des programmes écrits en ArchJava [18].

Le programme de transformation ATL consiste en une séquence de règles, dont chacune correspond à une entité du profil UML décrit dans la section précédente. On dénombre ainsi une dizaine de règles qui s'invoquent récursivement.

A la première étape, l'architecte spécifie son modèle d'architecture en utilisant l'éditeur arborescent généré à partir du profil UML (méta modèle) que nous avons proposé. Cette architecture est décrite par un ensemble de composants, des connecteurs qui les relient entre eux et des contraintes. Notre méta modèle peut être transformé en XML Schéma grâce au standard XMI qu'intègre EMF, permettant ainsi d'encoder le modèle d'architecture dans un document XML. Une validation conforme à l'approche de validation standard de MOF (Méta-Object Facility), est faite afin de vérifier la cohérence et la conformité entre le modèle et son méta modèle. Ainsi, le document XML correspondant au modèle d'architecture est validé par rapport au Schéma XML du méta modèle afin d'éviter des éventuelles incohérences, de détecter et corriger des erreurs de spécification.

A la seconde étape, l'architecte utilise le programme de transformation que nous avons proposé pour transformer automatiquement son modèle d'architecture UML2 en modèle d'architecture ArchJava (programme .archj). Ainsi, il pourra grâce au compilateur ArchJava, vérifier la consistance, la conformité et la traçabilité entre le modèle d'architecture UML2 et l'implantation qui en sera faite.

4 Discussion

Le développement des outils d'aide à la conception et à la transformation des architectures logicielles se pose avec acuité. La plupart des expériences menées dans ce cadre se concentre avant tout sur la définition d'un modèle de composants permettant de construire une architecture type selon le domaine d'application. Le projet *Service Component Architecture* (SCA) se situe dans la même lignée que notre préoccupation. Sauf que SCA est plus orienté vers l'implémentation des applications basées sur le principe des Architectures Orientées Service (SOA) [15]. Dans les autres cas, les outils permettent de vérifier la cohérence structurelle d'une architecture donnée et de valider sa sémantique par rapport à une approche le plus souvent formelle [14][16].

Notre préoccupation est similaire à COSA [13]. Elle consiste à développer des approches pour les architectures logicielles en s'appuyant sur les outils communément utilisés par les architectes et les développeurs afin d'offrir un environnement cohérent capable de supporter un procédé de développement des architectures logicielles à composants. Dans cette perspective, UML est populaire et grâce à ses capacités d'extension à travers les profils et à MDA qui permet de manipuler les modèles, nous pensons développer des principes directeurs pour organiser l'utilisation des diagrammes composite pour spécifier les architectures logicielles.

L'intérêt de notre démarche réside dans la réutilisation des outils existants au niveau de l'implémentation de l'environnement support de la démarche. Toutefois nous remarquons que ATL est propice pour la transformation structurelle du point de vue syntaxique. Il faudra enrichir ce mécanisme si on doit prendre en compte les aspects comportementaux et sémantiques, mais nous devons rester simple dans l'expression de ces aspects pour ne pas succomber aux critiques des approches trop formelles.

5 Conclusion et perspectives

Nous avons présenté dans cet article, une démarche pour supporter la tâche de conception des architectures logicielles utilisant le formalisme UML et, la transformation vers ArchJava et sa validation. Nous pensons que UML serait une approche complémentaire à ArchJava puisque c'est un standard de modélisation largement utilisé par les industriels et les chercheurs. De plus, il a introduit dans sa version 2.0 un modèle de composant abstrait inspiré des ADLs pour prendre en compte la description des architectures logicielles.

La description architecturale est faite suivant un profil UML et ensuite transformée automatiquement en ArchJava pour garantir la consistance entre le modèle architectural UML et son implantation. Cette démarche a pour but de résoudre le problème de découplage entre la spécification des architectures logicielles et leur implantation dans un langage de programmation cible.

Comme perspectives à nos travaux, nous envisageons d'aborder les aspects : hétérogénéité et distribution. Il s'agit de prendre en compte plusieurs modèles de composant cible y incluant les protocoles d'interaction qui seront implémentés dans les connecteurs. A cet effet l'enrichissement de notre profil UML par la définition des types de composants pour prendre en compte divers modèles de composants (EJB, CORBA, .Net, OSGI, etc.), des classes de connexion pour définir les styles d'interaction entre composants comme une entité de classe première afin de découpler l'implémentation des composants de la manière dont ils communiquent.

Sur le plan technique, l'éditeur du prototype est encore textuel. Sur ce point, le développement d'un environnement graphique de modélisation, basé sur ce Profil UML enrichi, intégrable sous forme de plugin Eclipse est une nécessité. Ceci dans le but de le coupler avec le plugin ArchJava existant afin de créer un environnement favorable, facilement manipulable par un grand nombre d'utilisateurs, pour la modélisation des architectures logicielles.

6 Bibliographie

[1] MOO MENA F. J. Modélisation des architectures logicielles dynamiques, Thèse de Doctorat, Institut National Polytechnique de Toulouse, Avril 2007

[2] Fabresse L. Du découplage à l'assemblage non-anticipé de composants : Conception et mise en œuvre du langage à composants SCL, PHD Thesis, Université de Montpellier II, Décembre 2007.

[3] Mellor S. *MDA Distilled, Principles of Model Driven Architecture*. Addison-Wesley Professional. ISBN 0-201-78891-8

[4] Ghizlane El Boussaidi Hafedh Mili, Les langages de description d'architectures, LATECE Technical Report, October 2006

[5] Medvidovic N., Redmiles D.F., Robbins J.E., Rosenblum D.S. Modeling software architectures in the unified modeling language. *ACM Transactions on Software Engineering and Methodology*, 11(1), January 2002.

[6] Garlan D., Kompanek A., Cheng S.W. Reconciling the needs of architectural description with object-modeling notations. *Science of Computer Programming Journal*, 44(1). PP 23–49, July 2002.

[7] Zarras A., Issarny V., Kloukinas C., Nguyen V. K. Towards a base UML profile for architecture description. *In 1st ICSE Workshop on Describing Software Architecture with UML*, pages PP 22–26. Canada, 2001.

[8] Baresi L., Heckel R. Thöne S., Varró D. Modeling and analysis of architectural styles based on graph transformation. In The *6th ICSE Workshop on Component Based Software Engineering*: Automated Reasoning and Prediction, May 3-4 2003.

[9] Barais O. Construire et Maîtriser l'architecture logicielle à Base de Composants. PHD Thesis, Laboratoire d'Informatique Fondamentale de Lille, Novembre 2005.

[10] HADJ KACEM M. Modélisation des applications distribuées à architecture dynamique: Conception et Validation, PHD Thesis, Université de Toulouse et l'Université de Sfax, Novembre 2008.

[11] Garlan D. Software Architecture : a road map. In *Proceedings of the conference on the future of software engineering*, may 2000.

[12] Megzari K. REFINER : un environnement logiciel pour le raffinement d'architectures logicielles fondé sur une logique de reécriture. Thèse de Doctorat, Université de Savoie, 2004.

[13] Smeda A. et al. COSAStudio : A Software Achitecture Modelling Tool, World Academy of Science, Engineering and Technology, 49, 2009.

[14] Mateescu R., Oquendo F. π-AAL: An Architecture Analysis Language for Formally Specifying and Verifying Structural and Behavioural Properties of Software Architectures. *ACM SIGSOFT Software Engineering Notes*, March 2006 Volume 31(2).

[15] Projets EMF et SOA Tools Platform de Eclipse. Disponible www.eclipse.org/stp/ and www.eclipse.org/emf/

[16] Oquendo F., Warboys B., Morrisson R., Dindeleux R., Gallo F., Garavel H. and Occhipinti C. ArchWare : Architecting Evolvable Software, in *Software Architecture, LNCS,* 2004, Vol 3047/2004, pp 257-271.

[17] Yie A., Casallas R., Deridder D. and Wagelaar D.. An Approach for Evolving Transformation Chains. In *Proceedings of the 12th International Conference on Model Driven Engineering Languages and Systems (MoDELS 2009),* Denver, CO, USA, October 4-9, 2009. LNCS 5795, pp. 551-555. Springer-Verlag.

[18] Abi-Antoun M., Aldrich J., Garlan D., Schmerl B., Nahas N., and Tseng T. Modeling and Implementing Software Architecture with Acme and ArchJava, in Proceedings of the ICSE'05, ACM 1-58113-963-2/05/005