HAL
open science

Une approche pour les architectures logicielles à composants multimédia

Makhlouf Derdour, Philippe Roose, Marc Dalmau, Nacira Ghoualmi-Zine, Adel Alti

To cite this version:

Makhlouf Derdour, Philippe Roose, Marc Dalmau, Nacira Ghoualmi-Zine, Adel Alti. Une approche pour les architectures logicielles à composants multimédia. Revue I3 - Information Interaction Intelligence, 2010, 10 (2), pp.67-92. hal-00572997

HAL Id: hal-00572997
https://hal.science/hal-00572997v1

Submitted on 2 Mar 2011

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Une approche pour les architectures logicielles à composants multimédia

Makhlouf Derdour[1], Philippe Roose[2]
Marc Dalmau[2], Nacéra Ghoualmi Zine[1], Adel Alti[3]

[1]Université d'Annaba
{m.derdour, ghoualmi}@yahoo.fr
[2]LIUPPA - IUT de Bayonne Pays Basque
{roose, dalmau}@ iutbayonne.univ-pau.fr
[3]Université de Sétif
altiadel2002@yahoo.fr

Résumé

Dans cet article nous proposons un méta-modèle pour les architectures à base de composants multimédia hétérogènes. Il n'existe actuellement pas de solution générique pour déployer automatiquement de telles architectures. La mise en évidence des incompatibilités d'assemblage entre composants est une nécessité dans de telles approches. En effet, les architectures logicielles valident les aspects fonctionnels, ce qui n'est pas suffisant pour garantir un assemblage réaliste et remédier aux problèmes d'hétérogénéité des flux de données échangés. Nous proposons, pour mettre en évidence ces incompatibilités et permettre de trouver des solutions, une approche basée modèle appelée MMSA (Meta-model for Multimedia Software Architecture). Elle permet la description d'architectures logicielles exprimant un système logiciel comme une collection de composants qui manipulent différents types et formats de données et qui interagissent par l'intermédiaire de connecteurs d'adaptation.

Mots-clés *: composant ; multimédia; adaptation; méta-modèle; architecture logicielle.*

Abstract

In this paper we propose a meta-model for architectures with heterogeneous multimedia components. Currently, a generic solution does not exist to automatically deploy a distributed architecture

based on multimedia components. The description of the incompatibilities between components is a need in such approaches. Indeed, software architectures validate the functional aspects, which are not sufficient to guarantee a realistic assembly. For instance, the problem of heterogeneity related to the exchanged data flows. In order to highlight these incompatibilities and to find solutions, a model-based approach called MMSA (Meta-model for Multimedia Software Architecture) is proposed. It enables the description of the software architectures expressing a software system as a collection of components which handle various types and formats of data, and interacts between them via connectors including the adaptation connectors.

***Keywords**: component; adaptation; concerns; multimedia; software architecture.*

1. INTRODUCTION

Le développement à base de composants est une approche largement utilisée pour construire des systèmes complexes. Le principe est d'affecter des exigences à des composants d'un certain type : classes, packages, services, etc. Bien que beaucoup d'exigences puissent être effectivement affectées à des composants individuels, certaines ne peuvent être limitées à un seul composant et ont des répercussions sur de nombreux composants *(Configuration)*. Les exigences expriment des préoccupations fonctionnelles et non-fonctionnelles. La conservation de telles préoccupations au cours de la conception et de la mise en œuvre rend un système difficile à comprendre et à maintenir. Il est communément admis qu'il est préférable de séparer les préoccupations fonctionnelles des préoccupations non fonctionnelles. Ceci facilite la recherche des composants métiers satisfaisant les préoccupations fonctionnelles et permet la factorisation de l'utilisation des composants assurant les préoccupations non fonctionnelles. Dans MMSA *(Meta-model for Multimedia Software Architecture)*, les deux types de préoccupations sont assurés respectivement par les composants et les connecteurs. Ainsi les connecteurs assurent la communication et la connexion des composants qui réalisent la partie métier/fonctionnelle.

La conception à base de composants connaît deux activités fondamentales : la conception pour la réutilisation, et la conception par la réutilisation. L'objectif majeur de la conception pour la réutilisation est de créer une bibliothèque complète de composants réutilisables alors que

celui de la conception par la réutilisation est de créer de nouveaux produits en réutilisant les composants déjà existants. Dans cet article nous mettons l'accent sur la deuxième activité pour construire des applications multimédia *(applications manipulant plusieurs types de médias tels que : le texte, l'image, le son et la vidéo)*. Comparées aux applications conventionnelles, ces applications souffrent de plusieurs problèmes liés à la diversité des médias *(type, format et caractéristiques)*, à leur nature hétérogène et à leur nécessaire adaptation. Ces difficultés sont accrues par le caractère pervasif de plus en plus présent des applications exécutées sur des périphériques mobiles.

L'hétérogénéité des composants en matière de mécanismes de communication *(GPRS, WIFI, Bluetooth, ZigBee, etc.)*, de vitesse de transmission, ainsi que la variété des médias *(son, vidéo, texte, image)* nécessite une prise en compte de l'adaptation à un niveau abstrait *(cf. fig 1)* afin d'éviter les solutions ad hoc non réutilisables et/ou non généralisables.

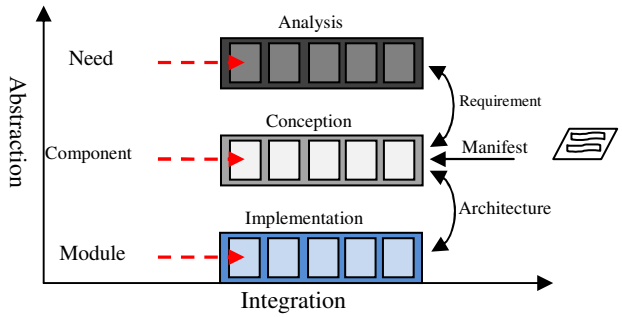

Fig 1. *Niveaux d'abstraction de l'architecture logicielle*

La phase d'analyse permet de séparer les préoccupations fonctionnelles et non-fonctionnelles de l'application. La phase de conception permet de mettre en évidence de nouvelles préoccupations non-fonctionnelles *(communication, adaptation, sécurité, etc.)* liées aux composants lors de la configuration. Pour cela, nous avons besoin d'un outil de modélisation capable de détecter l'hétérogénéité engendrée par les flux de données multimédia échangés entre composants.

Nous proposons ici un méta-modèle d'architecture logicielle pour applications intégrant les propriétés des flux de données multimédia. L'adaptation des flux de données est déportée sur les connecteurs appelés ici connecteurs d'adaptation. Ces derniers intègrent les services d'adaptation nécessaires ainsi que des extensions qualitatives de ces

services afin d'offrir une mesure de QoS reflétant l'évolution du flux de données suite aux adaptations.

Le reste de l'article est organisé comme suit : la section 2 présente les motivations de ce travail, la section 3 présente quelques travaux connexes, la section 4 présente le besoin d'adaptation pour les données multimédia, la section 5 présente notre méta-modèle d'architecture logicielle à base de composants multimédia, la section 6 aborde la prise en compte de l'adaptation dans MMSA, la section 7 détaille les propriétés et les caractéristiques des éléments d'implantation du méta-modèle. Enfin, l'article se termine par une conclusion et des perspectives.

2. MOTIVATION

Notre principale motivation est de proposer un méta-modèle pour maintenir la cohérence des architectures constituées par des composants hétérogènes *(flux multimédia, protocole de communication, etc.)* lors d'un (ré)assemblage ou d'une (re)configuration utilisant de nouveaux types d'interfaces graphiques et des connecteurs dotés d'une sémantique plus riche. L'utilisation de ces interfaces permet la détection automatique des points d'hétérogénéité entre composants tandis que l'utilisation des connecteurs d'adaptation permet la résolution de ces hétérogénéités. Les systèmes sont construits par assemblage de composants et de connecteurs où chaque élément peut être mis à la bonne place dans la configuration de l'architecture. Dans la plupart des ADL et des langages existants on trouve que :

- Le choix de connecteurs disponibles dans l'environnement se limite aux connecteurs primitifs. Les connecteurs composés ne sont pas pris en compte ;
- La gestion des préoccupations non-fonctionnelles des composants est assurée après la définition de l'architecture et de la configuration des composants ce qui engendre des problèmes d'incohérence ;
- La gestion des assemblages ne prend pas en considération les hétérogénéités comportementales des composants de l'architecture logicielle ce qui influe sur l'interopérabilité des composants ;
- Peu de modèles permettent de définir de nouveaux connecteurs avec des traitements différents assurant les préoccupations non-fonctionnelles des composants *(sécurité, communication, conversion, etc.)* ;
- Il n'y a pas de correspondance directe et automatique entre les architectures *(les modèles)* et les applications conçues suivant ces architectures *(instances)*.

Afin de répondre à ces insuffisances. Nous proposons MMSA pour décrire l'architecture logicielle basée composants multimédia. MMSA est fondé sur la définition de quatre types d'interfaces selon le type de flux de données *(son, vidéo, texte, image)* et d'une stratégie d'adaptation des flux multimédia *(type, format, propriété)* à trois niveaux. L'objectif est de résoudre le problème d'hétérogénéité des composants. Le méta-modèle est constitué pour atteindre les objectifs suivants :

- Assurer un niveau d'abstraction élevé pour les connecteurs afin de les rendre plus génériques et plus réutilisables, donc reconfigurables ;
- Prendre en compte la sémantique des liens de communication entre composants afin de détecter les points d'hétérogénéité et d'insérer des connecteurs d'adaptation en ces points ;
- Favoriser la gestion et le maintien de la qualité de service en donnant la possibilité d'ajouter, de supprimer et de substituer des services d'adaptation selon les besoins.

3. TRAVAUX CONNEXES

Les composants logiciels sont des entités réutilisables permettant une réduction des coûts liés au développement, à la maintenance et à l'évolution du logiciel. Actuellement, de nombreuses propositions se réclament du mode de développement par assemblage de composants logiciels. Malgré un vocabulaire parfois commun *(composant, port, interface, service, attachement, connecteur)*, ces propositions se différencient par leurs origines, leurs objectifs, leurs concepts ou encore leurs mécanismes.

Des ADL *(Architecture Description Language)* [CLE 96] et [BAR 05] sont utilisés pour spécifier des architectures logicielles. [MED 00] met en exergue la différence entre un ADL et une spécification formelle afin de distinguer les ADL des autres notations de modélisation. Par ailleurs, les types de connecteurs ont été étudiés par [MEH 00] qui propose une taxonomie de connecteurs permettant la prise en charge des propriétés non-fonctionnelles *(communication, sécurité, conversion, facilitation, coordination, interaction)*. Les modèles couvrent tout ou partie des besoins en termes de langage, de sémantique et d'outils. Toutefois dans [MEH 00] les auteurs relèvent des insuffisances pour la spécification des propriétés non-fonctionnelles des systèmes. De façon générale, nous avons remarqué un manque de fondement sémantique pour l'expression des contraintes et du raffinement *(composant, connecteur et*

configuration) et un manque d'outils pour la reconfiguration dynamique et l'évolution en temps réel.

Des approches comme [ALL 97a] [BER 00] [MAX 05] [ATT 09] permettant la séparation des préoccupations fonctionnelles ont été proposées dans le but de capitaliser les besoins fonctionnels dans des composants. Dans cette perspective plusieurs idées ont été proposées. On distingue principalement deux catégories d'approches pour les architectures logicielles : celles inspirées du développement de logiciels à base de composants (*CBSE*) et celles orientées services (*SOA*). Dans le premier cas [SZY 97] [ALL 97a] [BER 00] l'accent est mis sur la structure statique du système : les éléments logiciels sont des composants assemblés par des connecteurs dans des configurations. Dans le second cas [PAP 03] [MAX 05] [ATT 09] [ASR 09] l'accent est mis sur la structure fonctionnelle du système : les éléments logiciels sont des fonctionnalités *(des services)* liées par des relations de type collaboration ou composition.

Les applications modernes sont de plus en plus développées selon des processus de développement à base d'ADL [AVG 05]. Les ADL permettent d'analyser et de vérifier très tôt dans le cycle de développement les propriétés que le futur système devra satisfaire, en particulier les propriétés d'homogénéité et de compatibilité des composants manipulant divers médias. En effet, les applications actuelles telles que les systèmes embarqués incluent la notion de média comme une caractéristique importante de leur comportement [AVI 04] [BAL 03]. La plupart des ADL existants tels que SPT-UML [GRA 04], MARTE [OMG 06], fractal [BRU 04], SCA [BAR 07], Kmelia [ATT 09] et AADL [SAE 08] ne prennent en compte ni l'adaptation ni les propriétés liées au flux multimédia lors de la conception du logiciel. Certains, traitent le problème d'hétérogénéité par modification de paramètres de la configuration (ajout, retrait ou remplacement de composants) [MAR 04] ou par un méta-modèle qui vérifie l'adéquation du service à son contexte et recherche la stratégie d'adaptation [MAR 07].

[LUC 08] propose une comparaison des caractéristiques principales des langages de composants : composant, interface, port, service et connecteur. L'objectif principal de ce travail est la prise en considération de la connexion imprévue de composants développés de manière indépendante. Comme solution, il propose la production de connecteurs réutilisables et paramétrables à travers l'association d'un service de substitution aux ports fournis qui pourra être exécuté en cas d'absence du service demandé au niveau des ports. Ce travail n'offre pas de mécanismes d'intégration de nouveaux services de communication pour

assurer l'évolution de l'architecture vers de nouveaux besoins, ni de techniques de vérification de la qualité des architectures et des services fournis.

C3 *(Component Connector Configuration)* [AMI 09] est une approche centrée sur les architectures logicielles. Elle permet de décrire une vue de l'architecture logique afin de générer automatiquement l'architecture physique pour toutes les instances de l'application. L'idée est basée sur le raffinement et la traçabilité des éléments architecturaux. L'architecture des logiciels est décrite conformément aux trois premiers niveaux de la modélisation définis par l'OMG [OMG 06] [OMG 07]. Par conséquent, pour décrire l'architecture logique, trois types de connecteurs sont définis : le connecteur de connexion (CC), le connecteur de (dé)composition (CDC), et le connecteur de compression/expansion (ECC). Le typage des connecteurs permet de bien visualiser les différentes connexions entre les composants. En particulier, les connecteurs proposés n'assurent pas la connexion des composants hétérogènes et ne prennent pas en considération la sémantique des configurations et des liens entre composants.

Le tableau suivant résume les principaux concepts présents dans les architectures logicielles tels que : composant, connecteur, interface, etc. dans UML, Fractal, SCA et COSA [ALT 08].

	UML 2.0	**Fractal**	**SCA**	**COSA**
Composant	Une classe enrichie	Découpé en: Contenu & Membrane	Un ensemble de service et de références	Composé d'interfaces
Connecteur	Implicite. Il représente une relation de communication entre deux composants	/	/	Composé d'interfaces et d'une glu et Assure la communication entre deux composants
Configuration	Assemblage des composants (composants composites)	Composant composite	assemblage de composants et de services	Une composition de composants, de connecteurs et d'interfaces
Interface	- Associée à un port - Une interface peut être requise ou fournie ou les deux	− Point d'accès à un composant, − Deux types d'interface : fonctionnelle et contrôleur	On distingue : les services offerts et les services requis (références)	Composée de service et de ports.
Port	Permet d'assurer les flux fournis/requis du composant	/	/	Utilisé par les services pour communiquer avec l'extérieur
Attachement	Relation entre deux composants	Lien entre une interface client et une interface serveur.	Lien entre un service et une référence ou un autre composant Non-SCA.	Lien entre un port d'un composant et un rôle d'un connecteur.
Séparation des préoccupations	Non	Oui	Non	Oui

TAB. 1 – *Comparaison entre Fractal, SCA, COSA et UML*

Après ces différentes comparaisons, on peut dire que les architectures orientées services se rapprochent du monde des composants logiciels. Par exemple la spécification SCA (Service Component Architecture) propose un modèle de programmation pour la construction d'applications à base de composants. SCA intègre les orchestrations Web Service-BPEL comme des composants qui peuvent être assemblés avec d'autres composants SCA.

Les ADL peuvent être classés en trois catégories différentes [AMI 09] : les ADL sans connecteurs, les ADL avec un ensemble prédéfini de connecteurs et les ADL avec des types de connecteurs explicites. Dans le dernier cas, les ADL fournissent des connecteurs en tant qu'éléments du premier ordre du langage tels que : Wright [ALL 97b], [MED 99], ACME C2 [GAR 00], XADL [DAS 05], AADL [ALL 02], etc. Tous ces langages cherchent à améliorer la réutilisabilité des composants et des connecteurs en séparant le calcul et la coordination. Dans notre approche, nous optons pour une catégorie explicite de connecteurs. Ainsi, dans le méta-modèle MMSA, nous adoptons un type explicite et générique de connecteurs que le système peut spécialiser suivant les besoins de l'architecture et des composants. Nous détaillerons ce concept dans la section 7.2.

4. DONNEES MULTIMEDIA ET TECHNIQUES D'ADAPTATION

La réalisation d'applications multimédia s'appuie sur deux modèles complémentaires. Un modèle de flux multimédia permettant de représenter les différents types de médias échangés entre les composants ainsi que leurs relations structurelles et un modèle d'architecture basé sur les concepts d'ADL étendus au multimédia et intégrant les connecteurs d'adaptation. L'idée principale de cette proposition est de permettre la prise en compte des concepts standards des données multimédia ainsi que les propriétés non fonctionnelles d'un composant par des connecteurs au niveau de l'architecture logicielle (*adaptation de données, protocole de communication, sécurité, etc.*). L'objectif est ainsi de proposer une description générique, claire et complète. Dans ce qui suit, les différents concepts sont représentés à l'aide de modèles et, pour chaque modèle, nous détaillons les relations entre ses concepts.

4.1. Modèle de flux de données

Dans les environnements pervasifs (*et donc a priori hétérogènes et mobiles*), les appareils peuvent utiliser tout type de contenu allant d'un contenu textuel à des documents multimédia complexes et riches. Assurer la délivrance de données adaptées à chaque périphérique exige des techniques d'adaptation qui prennent en considération les médias et la structuration des flux. C'est pourquoi leur modélisation est nécessaire. Elle facilite le travail d'adaptation entre médias de même type (*image vers image par exemple*) ou entre médias de types différents (*texte vers son par exemple*).

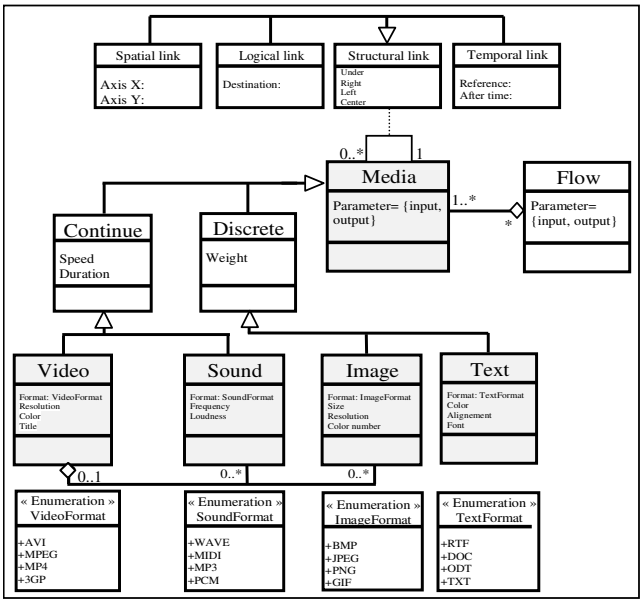

Fig 2. *Diagramme de classes de flux multimédia*

La structure hiérarchique des médias est exprimée en UML à l'aide d'un diagramme de classes (cf. fig 2). Les médias se classent en deux catégories : les médias continus, tels que la vidéo et le son, qui sont caractérisés par des dépendances temporelles et les médias discrets, tels que les images et les textes. A chaque type de média correspond un ensemble de formats d'encodage et un certain nombre de propriétés spécifiques comme la résolution (dans le cas des images/vidéos), la fréquence (dans le cas du son), etc. On distingue trois types de liens

structurels entre médias : temporel (permet de décrire les dépendances temporelles entre unités), logique (permet de décrire l'organisation logique d'un flux sous forme d'une hiérarchie de médias) et spatial (permet de décrire la disposition des éléments du flux multimédia).

Actuellement, les flux de données multimédia doivent pouvoir être utilisés sur de nombreuses plateformes matérielles (téléphones portables, PDA, ordinateurs de bureau, portables, etc.). Cette diversification des utilisations et des supports nécessite l'adaptation des flux à leur contexte d'exécution parfois imprévisible au moment de la conception des applications.

La représentation graphique des entrées/sorties offre un support de modélisation des flux de données multimédia et permet la spécification des adaptations nécessaires à la configuration des composants.

Type	Sortie	Entrée	Format		
Texte			DOC	DOCX	ODT
Image			JPEG	BMP	PNG
Son			WAVE	RM	MP3
Vidéo			MPEG	AVI	MP4

TAB. 2 – *Ports d'interface multimédia*

Pour notre méta-modèle nous proposons une notation graphique des ports d'interface multimédia. Elle permet d'identifier visuellement les entrées/sorties par type et par format de média pour chaque composant et ainsi de mettre en évidence la nécessité de la recherche de connecteurs d'adaptation en cas d'hétérogénéité (cf. tab 2).

4.2. Adaptation de flux de données

Chaque média peut subir trois types d'adaptation. La première est une conversion de format *(Transcodage)*. Elle permet la conversion en conservant le même type mais avec un format d'encodage différent *(image BMP vers image JPEG par exemple)*. La deuxième *(Transformation)* permet une manipulation des caractéristiques d'un média *(changement de résolution pour une image par exemple)* elle dépend du format du média. La troisième transformation, plus complexe, est appelée conversion de type *(Transmodage)*. Elle permet de passer d'un type de média à un autre *(texte vers son pour des non voyants par exemple)*. Cette conversion de type peut également agir sur les structures

de média en supprimant des dépendances temporelles *(vidéo vers images par exemple)*. Chacune de ces adaptations peut également avoir un impact sur la qualité des données. Ainsi, la conversion d'une image du format JPEG au format GIF implique la réduction du nombre de couleurs à 256, l'inverse implique la suppression de la composante « *transparence* » ce qui, selon le cadre d'utilisation, peut être problématique voire rédhibitoire.

Le tableau suivant présente une taxonomie des adaptations possibles entre médias :

Catégorie	Texte	Image
Transcodage	Conversion de format	Conversion de format
Transformation	Réduction de la taille de police Changement de police, couleur, etc.	Réduction de la taille Redimensionnement Réduction de profondeur de couleur Niveau de gris
Transmodage	Texte vers Son Texte vers Image	Image vers Texte
Catégorie	**Vidéo**	**Son**
Transcodage	Conversion de format	Conversion de format Changement d'échantillonnage
Transformation	Réduction de : -Taux de défilement -Résolution spatiale -Résolution temporelle -Profondeur de couleur	
Transmodage	Vidéo vers Image Vidéo vers Texte Vidéo vers Son	Son vers Texte

TAB. 3 – *Adaptations des médias*

Le diagramme de classes proposé dans la figure 3 permet la représentation des relations fonctionnelles et les dépendances entre les types d'adaptation *(Transcodage, Transmodage et Transformation)*.

Ce diagramme montre les différentes classes d'association permettant le passage d'un type de média à un autre ou d'un format de média à un autre. La relation entre la classe d'association *Transmodage* et la classe d'association *Transcodage* exprime que la classe *Transmodage* peut faire appel à la classe *Transcodage* pour accomplir sa tâche. Il en est de même entre la classe *Transcodage* et la classe *Transformation*. La classe média *(son, vidéo, texte, image)* utilise la classe *Transformation* pour changer les caractéristiques de média. Cette relation indique que chaque format de média a un ensemble de paramètres pour gérer les différentes qualités. La transformation est un type particulier d'adaptation qui conserve le type et le format de média mais provoque des changements de caractéristiques *(exemple : mise en noir et blanc d'une image JPEG)*.

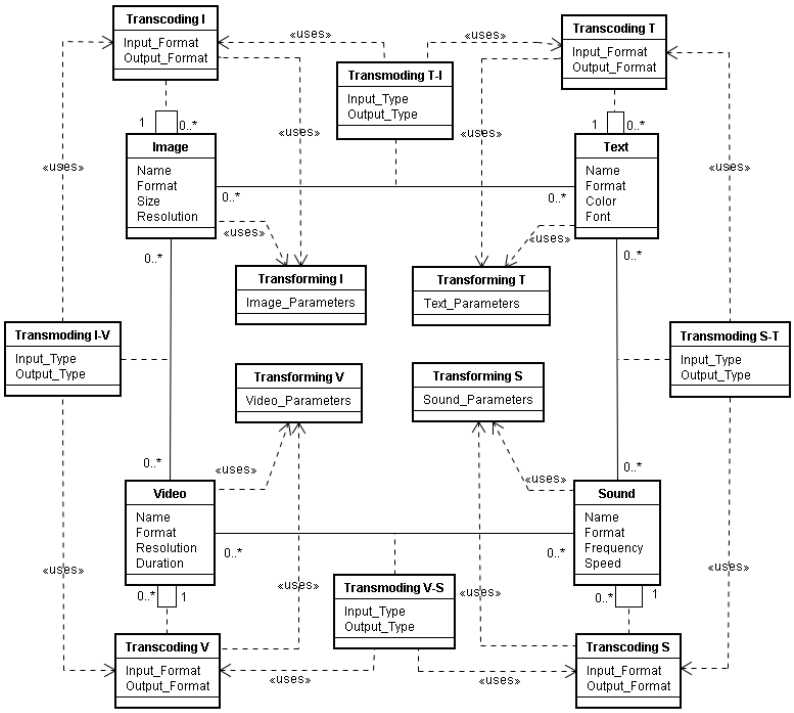

Fig 3. *Diagramme de classes des relations d'adaptation entre différents médias*

La généralisation du Transmoding et du Transcoding est impossible car les services d'adaptation n'ont ni les mêmes entrées, ni les mêmes sorties, ni les mêmes traitements.

5. LE META-MODELE MMSA

MMSA décrit l'architecture logicielle d'un système comme une collection de composants *(homogènes et hétérogènes)* qui interagissent par l'intermédiaire de connecteurs. Les composants et les connecteurs ont le même niveau d'abstraction et sont explicitement définis par la séparation de leurs interfaces et de leurs configurations internes.

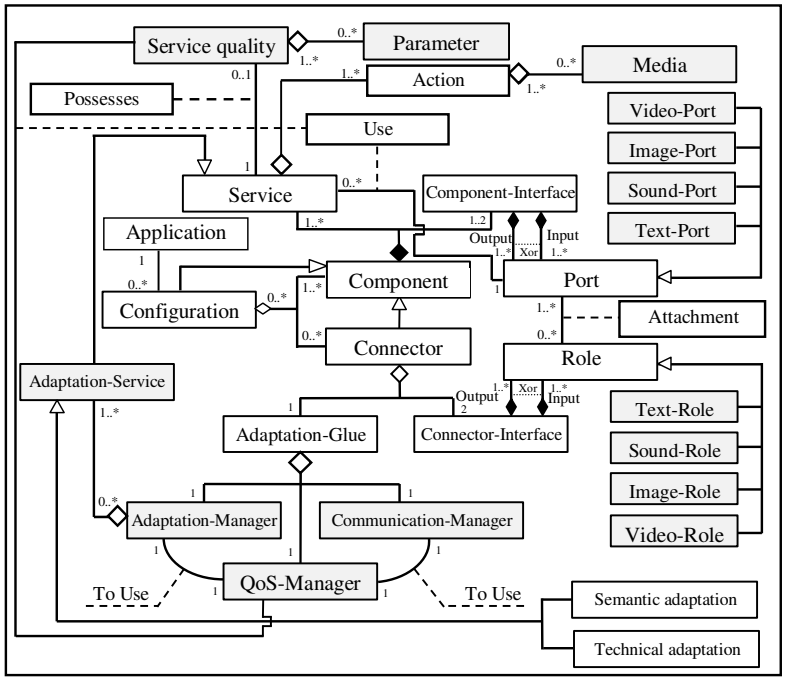

Fig 4. *Diagramme de classes du méta-modèle MMSA*

Les concepts de base de l'architecture logicielle MMSA sont les mêmes que dans la plupart des architectures logicielles, à savoir : configurations, composants et connecteurs. Le modèle d'architecture logicielle MMSA est un modèle hybride basé sur les concepts d'architectures orientées composants (CBSE) et d'architectures orientées services (SOA). Un composant est défini comme un ensemble de services interagissant pour remplir un rôle et communiquant avec l'environnement via deux interfaces requise/fournie. Généralement, les connecteurs définissent des abstractions qui encapsulent les mécanismes de communication, de coordination et de conversion *(type, nombre, fréquence et ordre des interactions)* entre les composants. Un connecteur est représenté par une interface et une glu [GOU 03] [SYL 07]. Cette description considère le connecteur comme un médiateur entre composants, ce qui limite son rôle à la communication. La spécification de la glu décrit la fonctionnalité qui est attendue d'un connecteur. Elle représente la partie cachée du connecteur. La glu peut être un simple protocole de communication liant les ports ou un protocole complexe qui

utilise diverses opérations, notamment la liaison, le transcodage/transmodage *(cf. fig 3)*, le transfert, etc.

Un composant est une unité de calcul ou de stockage à laquelle est associée une unité d'implantation et qui posséde un état. Il peut être simple ou composé- on parle alors de configuration. Les composants MMSA sont des abstractions qui encapsulent des services et manipulent des médias dans plusieurs formats via leurs interfaces. On distingue deux types d'interfaces : une interface de type « Output » qui exporte les données d'un composant et une de type « Input » qui importe les données vers le composant. Chaque interface décrit les informations nécessaires du composant incluant les points de connexion *(ports)*. On distingue un type de port par type de média identifié *(cf. Tab 2)*. Chacun fournit/requiert des médias du type correspondant. Cette distinction des ports par type de données *(son, vidéo, texte, image)* permet la vérification de l'assemblage des composants dès la conception. Cela permet de détecter les points d'hétérogénéité entre composants et de vérifier la cohérence et la validité des configurations des architectures logicielles.

Un connecteur MMSA est défini par deux interfaces «Input » et « Output » et une glu représentée par trois gestionnaires : communication, adaptation et qualité de service. Il gère le transfert des données entre les composants et permet de faire des adaptations à la volée. Une interface requise/fournie d'un connecteur se compose d'un ensemble de rôles. Chaque rôle sert de point par lequel le connecteur est relié à un composant. Ainsi, deux composants ne peuvent être liés que par un connecteur alors que deux connecteurs peuvent être liés entre eux pour créer des adaptations complexes.

Exemple 1 :

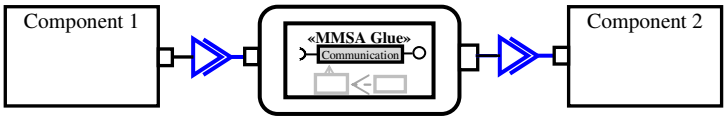

Fig 5. *Connecteur de communication*

Exemple 2 :

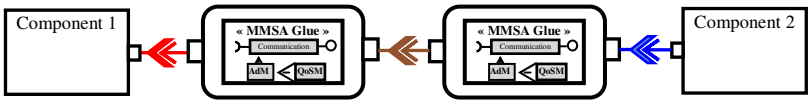

Fig 6. *Connecteurs de transcodage d'image JPEG vers BMP vers PNG*

Dans le premier exemple, le gestionnaire d'adaptation et le gestionnaire de QoS sont désactivés *(grisés)* il s'agit d'un connecteur de communication minimal pour connecter deux composants. Le second exemple montre la possibilité de connecter deux composants hétérogènes par deux connecteurs *(ou plus)* selon la complexité de l'adaptation.

Concernant les rôles de communication, nous avons ajouté des types pour clarifier les différentes liaisons entre composants en matière de flux de données. Au niveau interne nous avons enrichi la glu par un gestionnaire d'adaptation qui coopère avec un gestionnaire de qualité de service afin d'assurer la tâche d'adaptation de flux. Ce gestionnaire d'adaptation regroupe un ensemble de services d'adaptation qui coopèrent à la réalisation de l'adaptation. Deux types d'adaptation peuvent être réalisés au sein des architectures logicielles, l'adaptation sémantique (*conversion de type*) liée aux contraintes des données manipulées par les composants et l'adaptation technique *(conversion de format et ajustement des caractéristiques de média)* liée aux capacités des composants *(mémoire, écran, etc.)*. Le gestionnaire de qualité de service pilote le gestionnaire d'adaptation dans sa tâche afin de changer les paramètres des services d'adaptation pour atteindre une qualité adaptée aux besoins du composant au moment de l'exécution. Le gestionnaire de QoS participe au choix des paramètres des services d'adaptation technique des flux de données *(exemple : réduction de résolution, réduction du nombre d'images/sec)* et des services d'adaptation de type et de format lors de l'exécution (*ex : choix de taux de compression lors du passage de BMP au JPEG*).

Le but des configurations est d'abstraire les détails des différents composants et connecteurs *(encapsulation des constituants assurée par restriction de l'accès au travers des interfaces)*. Une configuration possède un nom et peut avoir une interface *(représentée par les interfaces des composants qui fournissent/requièrent des flux à/de l'environnement extérieur)* et un ensemble de services (*encapsulés dans les composants*).

La configuration est définie par des composants, des attachements et des connecteurs qui assurent les interactions entre les composants.

L'attachement est un lien de communication entre un port d'un composant et un rôle d'un connecteur *(un port de type « Output » doit être lié à un rôle de type « Input », un port de type « Input » doit être lié à un rôle de type « Output »)*. Dans notre configuration deux composants ne peuvent être liés que par un ou plusieurs connecteurs, cette exigence correspond au fait qu'un composant a besoin au minimum d'un connecteur pour communiquer avec un autre composant mais peut utiliser plus d'un connecteur suivant la complexité de la tâche d'adaptation.

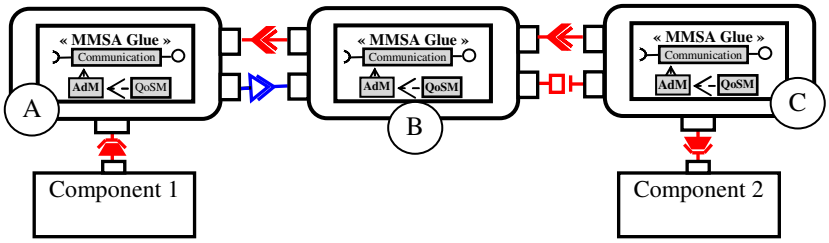

Fig 7. *Assemblage de connecteurs*

La figure 7 montre une adaptation qui fait intervenir plusieurs connecteurs : il s'agit de l'adaptation d'une vidéo *(son et image)* en une autre vidéo *(image et texte)* faisant appel à trois connecteurs d'adaptation *(Vidéo vers Son+Image (A), Son vers Texte (B) et Texte+Image vers Vidéo (C)).*

6. MMSA ET L'ADAPTATION

Au cours du processus de création de l'architecture, la prise en compte de l'adaptation pour résoudre le problème d'hétérogénéité des éléments de l'architecture *(composant, connecteur et configuration)* est faite en trois étapes successives : (I) adaptation de types (II) adaptation de formats et (III) adaptation de propriétés.

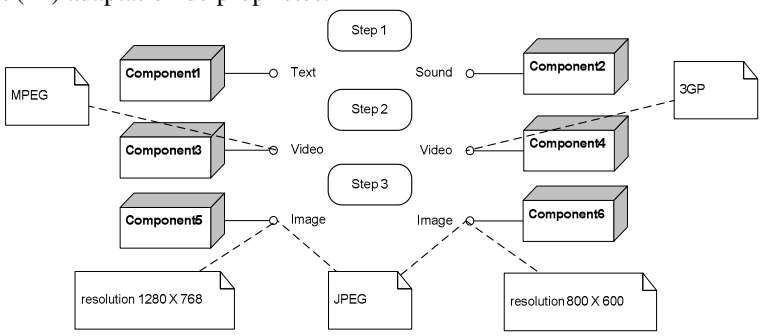

Fig 8. *Présentation des types d'hétérogénéités entre composants*

La détection d'hétérogénéités se fait automatiquement par la vérification des contraintes selon les trois étapes d'adaptation décrites ci-dessous.

6.1. Adaptation de types

L'hétérogénéité des composants qui manipulent différents types de médias est détectée par l'utilisation des différentes formes représentant les ports des composants *(étape 1, fig 8)*. Donc, deux composants qui n'ont pas les même ports *(exemple : port Texte et port Son)* ne peuvent être reliés que par l'utilisation d'un ou de plusieurs connecteurs d'adaptation de type *(cf. fig 9)*. Ce problème sera résolu, au niveau architectural par l'intégration de connecteurs de *Transmodage*.

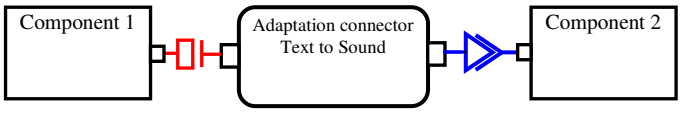

Fig 9. *Connecteur de Transmodage texte vers audio*

6.2. Adaptation de formats

L'hétérogénéité des composants qui manipulent le même type de média mais avec des formats d'encodage différents (*étape 2, fig 8*) peut être détectée par la présence de couleurs différentes. Donc, deux composants qui n'ont pas la même couleur pour le même port (*exemple : port vidéo avec couleur rouge .MPEG et bleu .3GP*) ne peuvent être reliés que par l'utilisation d'un ou de plusieurs connecteurs d'adaptation de format *(cf. fig 10)*. Ce problème sera résolu, au niveau architectural, par l'intégration des connecteurs de *Transcodage*.

Fig 10. *Connecteur de Transcodage MPEG vers 3GP*

6.3. Adaptation de propriétés

L'hétérogénéité des composants qui manipulent le même type de média avec le même format (*étape 3, fig 8*) mais avec des propriétés différentes (*exemple : résolution et couleur pour l'image, vitesse et échantillonnage pour la vidéo, etc.*) ne peut pas être exprimée visuellement dans notre architecture en raison du trop important nombre de paramètres dépendant du média et du service d'adaptation (*paramètres*

du service). Donc, deux composants ayant la même couleur pour le même port (*exemple : port Image*) peuvent être reliés par un simple connecteur de communication et ce ne sera que lors de l'exécution que les gestionnaires d'adaptation et de qualité de service gèreront l'adaptation. A ce niveau le problème d'hétérogénéité sera résolu à l'exécution par une manipulation des paramètres du service d'adaptation *(si le service est paramétrable)* en fonction du flux produit.

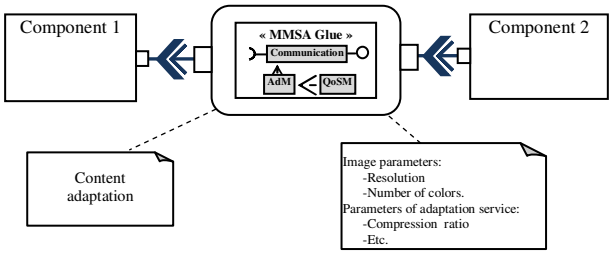

Fig 11. *Connecteur d'adaptation de contenu image*

Le service d'adaptation est paramétré pour qu'une adaptation puisse être appliquée dans différentes situations. Elle est appliquée dans plusieurs contextes, par exemple lors de l'adaptation de la résolution d'une image *(cf. fig 11)*. Si le service n'est pas paramétrable il faudra avoir recours à un changement de service d'adaptation.

7. ELEMENTS D'IMPLANTATION DE L'ARCHITECTURE MMSA

Pour la représentation graphique d'un composant nous avons choisi l'utilisation du modèle « Osagaia» [BOU 05]. En effet, le conteneur « Osagaia » intègre des unités spécifiques afin de gérer les flux de données et les buffers associés. Ce conteneur offre un certain nombre de commandes de supervision permettant de le connecter/déconnecter dynamiquement et de remplacer sa partie fonctionnelle *(le composant métier)*. Il fournit en outre tout un ensemble d'informations sur son contexte d'exécution qui permettent de décider d'éventuelles reconfigurations. Enfin, dans sa version actuelle, ce conteneur possède plusieurs implantations lui permettant d'être utilisé sur des périphériques plus ou moins contraints *(projet Kalimucho : Kalimucho.dev.java.net)*. Cet ensemble de caractéristiques en font un candidat intéressant pour nos

travaux. Néanmoins, le modèle « Osagaia » ne propose pas de typage des ports d'entrées/sorties pour représenter les divers médias, point qu'il faut impérativement résoudre dans le cadre de nos travaux.

7.1. Composant

Il est largement admis que le composant « ne peut être accessible que via des interfaces bien définies » [SZY 02]. Les interfaces lient les composants à l'environnement. Les langages basés composants proposent des concepts différents pour décrire ces interfaces : des éléments tels que les services, les ports, les protocoles, etc. avec, parfois, des significations différentes. Par exemple, dans Fractal [BRU 04] ou Enterprise JavaBeans [MON 99], les concepts de port et d'interface sont mélés c'est pourquoi on ne parle que d'interfaces. En revanche, dans les composants UML [CHE 00], les deux concepts de port et d'interface existent comme dans ArchJava [ALD 02] où les interfaces sont appelées port d'interfaces. C'est pourquoi nous allons expliquer clairement les choix que nous avons faits pour MMSA. Dans MMSA, un composant fournit ou requiert des services par les ports décrits dans l'interface fournie/requise. Ainsi, MMSA propose un typage des ports pour les différencier selon le type de média manipulé *(son, vidéo, texte, image)*.

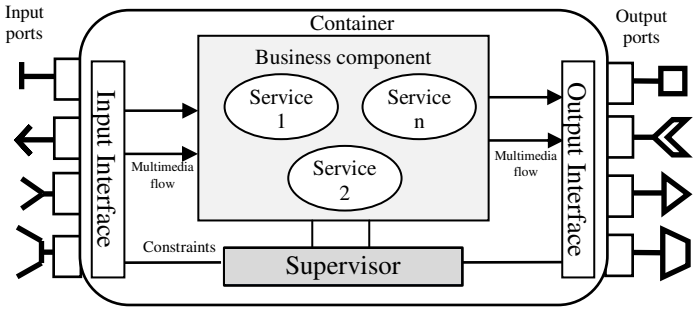

Fig 12. *Modèle de composants multimédia*

Un composant fournit des fonctionnalités qui sont nommées services. Fondamentalement, un service est un sous-programme défini dans un élément tout comme une méthode dans le modèle orienté objet. Un composant a deux catégories de services : services fournis et services requis.

« Un composant est une abstraction statique avec plugins » [NIE 95]. Les ports représentent ces plugins qui sont les points d'interaction des

composants. Cela signifie que tout passe par ces derniers, comme l'invocation de services par exemple. Le port est présent dans presque tous les modèles de composants mais avec différentes sémantiques. Dans les modèles de composants, les ports sont unidirectionnels ou bidirectionnels. Dans le cas de ports unidirectionnels comme dans ComponentJ [SEC 00] ou Fractal [BRU 04] un composant fournit ou requiert l'ensemble des services via ses ports. En ArchJava [ALD 02] ou UML 2.0 [CHE 00] les ports sont bidirectionnels et un composant requiert et fournit à la fois des services à travers chacun de ses ports.

Dans MMSA les ports sont unidirectionnels parce qu'un port peut fournir/requérir des données via/à partir des connecteurs. Ces derniers peuvent appliquer des adaptations sur les données et, généralement, le service d'adaptation est unidirectionnel ; d'un point de vue fonctionnel. La fonction inverse n'est pas toujours pertinente, voire réalisable *(cf. fig 9, le service d'adaptation du texte vers le son n'est pas le même service que celui d'adaptation du son vers le texte).*

7.2. Connecteur

Comparés à ceux des langages de description d'architectures [ALL 97a] [MEH 00] les connecteurs que nous proposons peuvent être simples ou composés et peuvent également assurer des services. Autrement dit, ces connecteurs se chargent de l'adaptation des données échangées entre composants.

Dans notre approche, le connecteur constitue l'entité de communication et d'adaptation, c'est-à-dire qu'il est capable de transférer des données multimédia entre les différents composants tout en assurant l'adaptation de ces dernières.

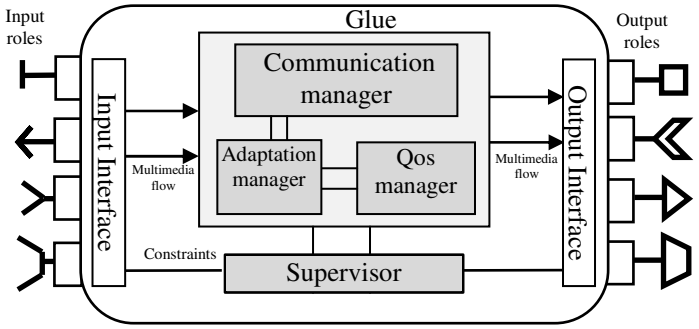

Fig 13. *Modèle de connecteurs multimédia*

Permettre à des composants hétérogènes d'interagir est une tâche non négligeable. L'adaptation est considérée comme un besoin non fonctionnel d'un composant, cette tâche doit être assurée par un autre élément. Le connecteur fournit les aspects non fonctionnels *(communication, adaptation, sécurité, etc.)* dont le composant a besoin. Le rôle du connecteur d'adaptation est de recevoir les données, de les adapter selon les directives du gestionnaire de QoS et de les acheminer vers le composant/connecteur destinataire.

Le superviseur est chargé de conserver les contraintes des flux reçus par l'interface d'entrée et émis par l'interface de sortie. Il est également chargé de la supervision du connecteur pour pouvoir demander sa reconfiguration ou le remplacement des services d'adaptation dans le cas où le gestionnaire de QoS n'arrive pas à offrir la qualité exigée.

8. CONCLUSION

Contrairement aux ADL présentées, nous considérons les connecteurs comme des unités d'interaction dotées de comportements dynamiques qui leur permettent la prise en compte de préoccupations non-fonctionnelles.

Dans cet article, nous avons présenté un méta-modèle générique pour la description des architectures logicielles tout en intégrant les concepts du multimédia. Ceci nous a permis de présenter de façon séparée les paramètres de flux multimédia en raison du fait qu'ils représentent un aspect très important des configurations et assemblages de composants. La contribution de ce travail se situe dans un contexte de description par niveaux d'abstraction intégrant de façon séparée les préoccupations fonctionnelles et non-fonctionnelles des composants. Ceci permet d'assurer une qualité d'assemblage des composants à partir des adaptations assurées par des connecteurs ainsi qu'une qualité de service d'adaptation. Les points forts de MMSA sont la prise en compte de l'aspect multimédia et la séparation entre l'aspect fonctionnel et non-fonctionnel des composants.

Notre proposition peut servir de support au développement d'applications de gestion de ressources numériques *(DAM : Digital Asset Management)*. Ce type d'application manipule une grande variété de médias et communique avec les utilisateurs au moyen de diverses plateformes *(téléphones portables, PDA, ordinateurs de bureau ou portables, etc.)*. MMSA peut apporter une solution efficace au développement des DAM surtout dans les parties : acquisition, traitement, distribution et utilisation du contenu. Il permet de résoudre les problèmes

d'incompatibilité au niveau de l'architecture par injection de connecteurs d'adaptation et au niveau de l'exécution par gestion de la QoS et reconfiguration de ces connecteurs.

Les perspectives de ce travail sont doubles. La première est de finaliser le *MMSAPlugIn* (pour Rational Software Modeler) qui permettra de créer/modifier rapidement une architecture logicielle, de construire facilement des modèles d'architecture UML 2.0 avec l'approche MMSA ainsi que de vérifier l'hétérogénéité des architectures produites. La seconde, à plus long terme, est de proposer une description des stratégies d'adaptation et de gestion de la QoS afin d'automatiser autant que possible l'adaptation et les connexions au moyen de l'implantation des différents gestionnaires *(Adaptation, QoS, Communication)*.

9. REFERENCES

[ALD 02] Aldrich J, Chambers C, Notkin D. ArchJava : Connecting software architecture to implementation. In ICSE. ACM; 2002. p. 187–97.

[ALL 97a] Allen, R. Garlan, D. A Formal Basis for Architectural Connection. ACM Transactions on Software Engineering and Methodology, vol. 6, no 3, 1997, p. 213–249.

[ALL 97b] Allen, R.J., A Formal Approach to Software Architecture, PhD Thesis, School of Computer Science, Carnegie Mellon University, 1997.

[ALL 02] Allen, R., Vestal, S., Lewis, B., Cornhill, D., Using an architecture description language for quantitative analysis of real-time systems, Proceedings of the Third International Workshop on Software and Performance, ACM Press, Rome, Italy, pages 203–210, 2002.

[ALT 08] Alti, A. Djoudi, M. Smeda, A. An automatic transformation from COSA software architecture to CORBA platform. 8th international conference on New technologies in distributed systems table of contents. Lyon, France, 2008.

[AMI 09] Amirat, A. Oussalah, M. First-Class Connectors to Support Systematic Construction of Hierarchical Software Architecture. Journal of Object Technology, vol. 8, no. 7, November-December 2009, pp. 107-130

[ASR 09] El Asri B., Kenzi A., Nassar M.et Kriouile A. Vers une architecture MVSOA pour la mise en œuvre des composants multivue, 3e Conférence francophone sur les Architectures Logicielles, 2009, pp 1-17. RNTI.

[ATT 09] Attiogbé, C. André, P. and Messabihi, M. Correction d'assemblages de composants impliquant des interfaces paramétrées. 3ième Conférence Francophone sur les Architectures Logicielles. Hermès, 2009.

[AVI 04] Avizienis A., Laprie J.-C., Randell B., Landwehr C. « Basic Concepts and Taxonomy of Dependable and Secure Computing ». IEEE Transactions on Dependable and Secure Computing. January-March 2004. pp. 11-33.

[AVG 05] Avgeriou P. Uwe Zdun. Modeling Architecture Patterns using Architecture Primitives, OOPSLA' 05, ACM, October 2005.

[BAL 03] Balsamo S., Bernado M., Simeoni M. Performance Evaluation at the Architecture Level Formal Methods for Software Architectures. LNCS 2804. Springer. Berlin, Germany. pp. 207-258. 2003.

[BAR 05] Barais O. Construire et maîtriser l'évolution d'une architecture logicielle à base de composants, PhD thesis, LIFL, Université Lille 1, novembre 2005.

[BAR 07] Barber G. (2007), What is SCA, http://www.osoa.org.

[BER 00] Bergner K., Rausch A., Sihling M., Vilbig A., Broy M. A Formal Model for Component ware. LEAVENS G. T., SITARAMAN M., Eds., Foundations of Component-Based Systems, Cambridge University Press, 2000, p. 189–210, New York.

[BOU 05] Bouix E. Dalmau, M. Roose, P. Luthon, F. A Multimedia Oriented Component Model, AINA 2005, Tamkang University, Taiwan.

[BRU 04] Bruneton E, Coupaye T, Leclercq M, Quéma V, Stefani J-B. An open component model and its support in Java. Crnkovic I, Stafford JA, Schmidt HW, Wallnau KC, editors. CBSE. Lecture notes in computer science, vol. 3054. Berlin: Springer; 2004. p. 7–22.

[CHE 00] Cheesman J, Daniels J. UML components: a simple process for specifying component-based software. Boston, MA, USA: Addison-Wesley; 2000.

[CLE 96] Clements P. C. A Survey of Architecture Description Languages. IWSSD '96: Proceedings of the 8th International Workshop on Software Specification and Design, Washington, DC, USA, IEEE Computer Society, 1996, page 16.

[DAS, 05] Dashofy, E., Hoek, A.v.d., Taylor, R.N., "A comprehensive approach for the development of XML-based software architecture description languages", Transactions on Software Engineering Methodology (TOSEM), volume 14, issue 2, pages 199–245, 2005.

[GOU 03] Goulao. M, F.B. Abreu, Bridging the gap between ACME and UML 2.0 for CBS. In: Proceedings Workshop of Specification and Verification of Component-Based Systems, Helsinki, Finland, 2003.

[GAR, 00] Garlan, D., Monroe, R.T., and Wile, D. “Acme: Architectural Description Component-Based Systems, Foundations of Component-Based Systems”. Cambridge University Press, pages 47-68, 2000.

[GRA 04] Graf S. Ober I. How useful is the UML realtime profile SPT without semantics, SIVOES 2004, associated with RTAS 2004, Toronto Canada, April 2004.

[LUC 08] Luc, F. Christophe, D. Marianne, H. Foundations of a Simple and Unified Component-Oriented Language. Journal of Computer Languages, Systems & Structures, editor Elsevier, Volume 34/2-3, 2008, p. 130-149.

[MAR 04] Marcel, C. Michel, R. Christian, M. Calin, L. Costin, M. Adaptation dynamique de services. DECOR’04, Grenoble, France. October 2004.

[MAR 07] Marcel, C. Michel, R. Christian, M. "Autonomic Adaptation based on Service-Context Adequacy Determination" in Electronic Notes in Theoretical Computer Science (ENTCS), vol. 189, pages 35-50, Elsevier, jul 2007.

[MAX 05] Maximilien, E. Singh, M. Self-adjusting trust and selection for web services, June 2005, pp. 385–386.

[MED, 99] Medvidovic, N., Rosenblum, D. S., and Taylor, R. N. A Language and Environment for Architecture-Based Software Development and Evolution, ICSE’99, Los Angeles, May 1999.

[MED 00] Medvidovic N., Taylor R. N. A Classification and Comparison Framework for Software Architecture Description Languages - IEEE Transactions on Software Engineering, vol. 26, 2000, no 1, p. 70–93.

[MEH 00] Mehta N. R., Medvidovic N., Phadke S. Towards a taxonomy of software connectors, ICSE ’00, ACM Press, 2000, p. 178–187

[MON 99] Monson-Haefel R. Enterprise JavaBeans. Sebastopol, CA, USA: O’Reilly & Associates Inc.; 1999.

[NIE 95] Nierstrasz O, Dami L. Object-oriented software composition. Englewood Cliffs, NJ: Prentice-Hall; 1995. p. 3–28.

[OMG 05] OMG. UMLTM Profile for Modeling and Analysis of Real-Time and Embedded systems (MARTE). 2005.

[OMG 06] OMG. Unified Modeling Superstructure" from, http://www.omg.org/docs/ptc/06-04-02.pdf, 2006.

[OMG 07] OMG: "Unified Modeling Language: Infrastructure" from http://www.omg.org/docs/ formal/07-02-06.pdf, 2007.

[PAP 03] Papazoglou M. P. Service-Oriented Computing: Concepts, Characteristics and Directions. WISE, IEEE CS, 2003, p. 3–12.

[SAE 08] Society of Automotive Engineers (SAE). Architecture Analysis & Design Language (AADL). SAE Standards no AS5506, November 2008.

[SEC 00] Seco JC, Caires L. A basic model of typed components. Lecture Notes in Computer Science 2000; 1850:108–29.

[SYL 07] Sylvain, M. Smeda, A. Ouassalah, M. COSA: An Architectural Description Meta-Model. ICSOFT (SE) 2007: 445-448

[SZY 97] Szyperski C. Component Software: Beyond Object-Oriented Programming – AddisonWesley Publishing Company, 1997.

[SZY 02] Szyperski C. Component software: beyond object-oriented programming. MA: Addison-Wesley; 2002.