RENPAR'16 / CFSE'4 / SympAAA'2005 / Journées Composants
Le Croisic, France, 5 au 8 avril 2005

Composition et Reconfiguration Hiérarchiques pour des Services Multimédia Auto-Adaptables

Oussama Layaïda et Daniel Hagimont

INRIA Rhônes-Alpes,
Zirst - 655 avenue de l'Europe -
Montbonnot 38334 Saint Ismier Cedex - France
Prenom.Nom@inrialpes.fr

Résumé
Avec le développement des équipements mobiles, les applications multimédia s'exécutent dans des environnements hétérogènes et instables. Une approche prometteuse pour prendre en compte cette situation est de rendre ces applications auto-adaptables, en leur fournissant la possibilité de s'auto-observer et d'observer leur environnement d'exécution, de détecter des changements significatifs et de se reconfigurer en conséquence. De nombreux travaux ont exploré cette approche, en particulier dans le domaine des applications multimédia. Cependant, les solutions proposées ont d'importantes lacunes en termes de flexibilité. Cet article présente PLASMA, un intergiciel à composants permettant la construction d'applications multimédia auto-reconfigurables. Pour atteindre ses objectifs, PLASMA repose sur les propriétés fortes de son modèle à composants : composition récursive et langage de description d'architecture (ADL) dynamique. Les expérimentations menées montrent que les reconfigurations peuvent être effectuées à très faible coût et qu'elles permettent d'améliorer significativement l'usage des ressources (et la QoS) pour des équipements mobiles.

Mots-clés : Multimédia, Auto-Adaptation, Composants, Reconfiguration, Fractal.

1. Introduction

Avec le développement des équipements mobiles, les applications multimédia s'exécutent dans des environnements hétérogènes et instables. Une approche prometteuse pour prendre en compte cette situation consiste à adapter l'application aux contraintes de son environnement d'exécution. Ces contraintes étant variables, cette adaptation est également nécessaire en cours d'exécution. Dans le cas idéal, l'application est capable de s'adapter elle-même, elle est donc auto-adaptable. Pour ce faire, les applications doivent avoir la possibilité de s'auto-observer et d'observer leur environnement d'exécution, de détecter des changements significatifs et de se reconfigurer en conséquence.

Ces reconfigurations visent en général la gestion de la qualité de service à l'exécution. En particulier, les applications multimédia réparties sont très sensibles aux fluctuations des ressources, que ce soit les ressources du terminal sur lequel les données multimédia sont présentées, ou encore les ressources du réseau de communication utilisé. Cependant, fournir un environnement pour le développement d'applications multimédia auto-adaptables n'est pas aisé. De nombreux problèmes difficiles doivent être considérés afin d'obtenir une solution applicable. Les plus importants de notre point de vue sont les suivants :

– *Adaptation aux applications* : Différentes applications peuvent avoir des contraintes de QoS variées, nécessitant ainsi des stratégies de reconfiguration très différentes. La solution recherchée doit donc permettre de répondre à un large spectre de besoin de stratégies d'adaptation et les spécialiser aux besoins de chaque application.
– *Adaptation à l'environnement* : Les environnements d'exécution peuvent varier fortement entre différentes exécutions d'une application, voire même au cours d'une même exécution. L'application doit donc être en mesure d'observer ces variations et de s'adapter en conséquence, dynamiquement.

Cet article présente PLASMA, un intergiciel à composants qui facilite la construction d'applications multimédia auto-adaptables. Nous décrivons PLASMA et montrons de quelle manière il répond aux besoins énoncés ci-dessus. La suite de cet article est organisée en 6 sections. La section suivante présente un état de l'art autours des applications multimédia adaptables. La section 3 décrit les concepts utilisés pour la construction de PLASMA et la section 4 détails son architecture et ses principaux composants. En suite, la section 5 montre une évaluation des performances effectuée dans le cadre d'applications multimédia sur terminaux mobiles. Finalement, la section 6 conclut cet article et présente nos travaux futurs.

2. Travaux Similaires

Ces dernières années, le développement des intergiciels et des systèmes à composants [1] a beaucoup contribué à la conception de systèmes adaptables. Les systèmes à composants fournissent au développeur d'application des abstractions de haut niveau, le soulageant de nombreuses fonctions récurrentes comme le déploiement des applications ou la reconfiguration d'une architecture logicielle. Dans cette lignée, les travaux autour des applications multimédia ont débouché sur le développement de plusieurs canevas à composants tels que DirectShow (Microsoft)[10], JMF (Sun)[11] et PREMO (ISO)[6]. L'idée commune à ces systèmes est d'implanter les différents traitements sur les données multimédia dans des composants séparés, qui peuvent être composés dans différents types d'applications. De même, les opérations de reconfiguration sont facilitées à travers des opérations de haut niveau telles qu'ajuster les valeurs des propriétés des composants, arrêter/démarrer ces composants, ajouter ou supprimer des composants ou modifier l'assemblage des composants ; une reconfiguration étant une combinaison de ces opérations élémentaires.

Dans ce domaine, de nombreux travaux de recherche ont exploré la conception d'applications auto-adaptables (en s'appuyant ou pas sur des intergiciels à composants). Notre étude de l'état de l'art nous a permis de dénombrer quatre grandes approches :

– Politiques de reconfiguration statiques : Les premiers travaux utilisent des politiques de reconfiguration statiques pour répondre à des changements prédéterminés dans l'environnement d'exécution. Les outils VIC [15] (Video Conferencing Tool) et RAT [9] (Robust Audio Tool), bien qu'ils ne soient pas structurées en terme de composants, utilisent un algorithme de contrôle de congestion (basé sur le taux de perte de paquets) afin d'adapter dynamiquement les flux multimédia en fonction de la bande passante disponible. La reconfiguration consiste à manipuler les paramètres d'encodage (facteur de qualité, cadence d'image) afin d'ajuster le débit de transmission à l'état du réseau. Cependant, outre la limitation à une application particulière, l'emploi d'une stratégie d'adaptation statique est très restrictif et nécessite l'anticipation de toutes les situations pertinentes lors du développement.

– Canevas à composants avec capacités de reconfiguration : Dans le prolongement des travaux cités précédemment (DirectShow, JMF, PREMO), d'autres travaux ont proposés des canevas à composants enrichis de capacités de reconfiguration. La boite à outils TOAST [7] (Toolkit for Open Adaptive Streaming Technology) s'appuie sur la réflexivité pour faciliter le développement d'applications multimédia adaptables. TOAST offre une API de haut niveau modélisant les opérations de reconfiguration décrites précédemment. La limitation de ces travaux réside dans le niveau d'abstraction fournit pour le développement des stratégies d'adaptation. L'implantation de ces stratégies est à la charge du développeur de l'application (dans le même langage de programmation) qui doit se préoccuper de l'observation des ressources et de l'application, de l'implantation de la reconfiguration. Une plus grande flexibilité peut être assurée en exploitant les architectures logicielles à composants et en spécifiant la stratégie d'adaptation comme une propriété non-fonctionnelle.

– Politiques de reconfiguration intra-composant : Dans ce sens, certains environnements à composants intègrent les fonctions de reconfiguration dans les composants applicatifs. Par exemple dans DirectShow, les composants multimédia (appelés Filtres) échangent des messages de QoS dans le sens opposé au flux des données. Avec ce mécanisme, un filtre peut indiquer à ses prédécesseurs que le débit

de données est trop élevé ("flood") ou trop bas ("famine"). Les reconfigurations se limite à ajuster la cadence de traitement par les composants, mais elle peut facilement être étendue pour fournir une gestion plus élaborée [5]. Comme dans la première approche, les opérations de reconfiguration sont codées statiquement dans les composants. Mais le principal défaut de cette approche est que les reconfigurations interviennent dans le contexte local de chaque composant, plutôt que dans le contexte global de l'application. Ceci rend notamment impossible le remplacement d'un composant, puisque la structure de l'application n'est pas connue des composants.

– Gestionnaire de reconfiguration séparé : A l'opposé de la précédente, une autre approche se base sur un gestionnaire de reconfiguration séparé de l'application. Les changements sont notifiés à ce dernier, qui se charge ensuite d'opérer les reconfiguration sur les composants de l'application. CANS (Composable Adaptive Network Services [8]) et APC (Automatic Path Creation)[16] suivent cette approche pour construire des passerelles de transcodage multimédia. QCompiler [18] et ACEEL [4] apportent plus de flexibilité en enrichissant le gestionnaire de reconfiguration avec un langage de spécification, permettant d'ajuster les paramètres de la politique de reconfiguration. Une limitation avérée de cette approche est que le gestionnaire de reconfiguration se trouve fortement couplée à l'architecture de l'application. En effet, l'exécution des reconfigurations nécessite une connaissance de l'architecture de l'application, ce qui rend impossible la généralisation de cette approche à des compositions arbitraires.

En conclusion, la conception d'une infrastructure à un haut niveau d'abstraction, permettant de construire des applications multimédia auto-adaptables, reste un problème ouvert. Notre objectif est de fournir un intergiciel à composants multimédia, permettant (1) de décrire une application en termes d'une architecture à composants, et (2) de décrire une politique d'adaptation à un haut niveau d'abstraction en termes d'opérations sur ces composants.

3. Choix de Conception

Cette section présente les principaux concepts suivis dans PLASMA, afin de combler les lacunes identifiées précédemment. Nous présentons le modèle à composant utilisé ainsi que notre approche pour la reconfiguration.

3.1. Le modèle à composants Fractal

PLASMA repose sur le modèle à composants Fractal [2], un canevas logiciel dont l'objectif est de permettre la composition, le déploiement, l'introspection et la reconfiguration de structures logicielles complexes. Comparé aux autres modèles à composants, Fractal possède les particularités suivantes :

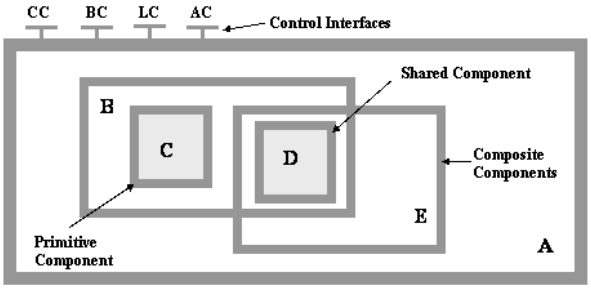

FIG. 1 – Architecture de composants Fractal

– *Composition hiérarchique* : Le modèle Fractal définit deux types de composants : les composants primitifs et les composants composites. Un composant primitif encapsule du code fonctionnel (C et D

sur la Figure 1) alors qu'un composant composite encapsule des sous-composants qui peuvent être primitifs ou composites (A, B et E). Le modèle est récursif dans le sens où des composants peuvent apparaître à des niveaux de composition arbitraires avec une structure similaire, d'où le nom Fractal. Dans un composant composite, les interfaces des sous-composants sont reliées par des liaisons pour former une architecture logicielle.
- *Modularité* : Les composants Fractal sont formés d'un contenu (code fonctionnel ou sous-composant) qui est sous le contrôle d'une membrane. Cette dernière définit quatre types d'interfaces de contrôle associées à ce composant :
  - *Binding Controller (BC)* : fournit les opérations pour lier et délier les interfaces du composant.
  - *Content Controller (CC)* : permet de lister, ajouter ou retirer des sous-composants dans le contenu du composant.
  - *Life-cycle Controller (LC)* : permet le contrôle explicite de l'exécution du composant (par exemple démarrage et arrêt).
  - *Attribute Controller (AC)* : permet d'accéder et d'affecter les attributs du composant depuis l'extérieur.
- *Partage de composant* : Une caractéristique du modèle Fractal est de permettre le partage de composants entre plusieurs composites englobants (composants D). Le partage de composant est très utile pour modéliser le partage de ressource entre composants.

Ces particularités de Fractal en font un bon candidat pour atteindre nos objectifs. La récursivité du modèle apporte une grande flexibilité pour la composition des architectures logicielles complexes ainsi que leur reconfigurabilité, comme nous le verrons dans la suite de cet article.

3.2. Approche pour la reconfiguration

Reconfiguration hiérarchique : En reposant sur le modèle Fractal, les opérations de reconfiguration sont considérées à différents niveaux de composition. Comme le montre la Figure 1, chaque composant possède son propre gestionnaire de reconfiguration responsable de la reconfiguration de son contenu. Une opération de reconfiguration est alors vue comme la modification d'un ou plusieurs attributs de ce composant. Dans le cas d'un composant primitif, ceci correspond à une modification d'une propriété (un attribut) du composant. Dans le cas d'un composite, le traitement d'une opération de reconfiguration dépend de la sémantique associée à cet attribut. L'opération peut être déléguée à un de ses sous-composants ; elle peut également intervenir dans le contexte du composite, par l'ajout/la suppression de sous-composants, la modification des liaisons entre les sous-composants ou l'arrêt/démarrage des sous-composants. Ceci permet d'intégrer les reconfigurations à différents niveaux de hiérarchie indépendamment de la structure de l'application.

ADL dynamique : PLASMA repose sur l'utilisation d'un langage de description d'architecture (ADL) dynamique. Outre la description des relations de liaisons et d'encapsulation entre les composants, un ADL dynamique coordonne les opérations de reconfiguration en termes d'observations et d'opérations de reconfiguration à appliquer sur sa structure initiale. L'intergiciel fournit les mécanismes permettant d'implanter une telle description au sein d'une architecture à composants. Ceci offre deux précieux avantages :
- Les politiques de reconfiguration peuvent être facilement modifiées ou spécialisées en fonction des besoins des applications ou du contexte d'exécution.
- Les politiques de reconfiguration peuvent être implantées en termes de composants, qui sont dynamiquement assemblés pour réaliser différents algorithmes d'adaptation.

4. Architecture de PLASMA

L'architecture de PLASMA définit quatre types de composants : des composants multimédia pour composer la partie fonctionnelle de l'application, les composants d'observation appelés *Probes*, des *Sensors* responsables de la détection des changements pertinents et des *Actuators* pour opérer les reconfigurations. Les composants multimédia sont architecturés à plusieurs niveaux de composition. Ces trois derniers ont un rôle précis et peuvent être intégrés à différents niveaux de composition. Les sections qui suivent détaillent le rôle de chacun.

4.1. Composants Multimédia

On trouve trois types de composants multimédia correspondant à trois niveaux hiérarchiques :

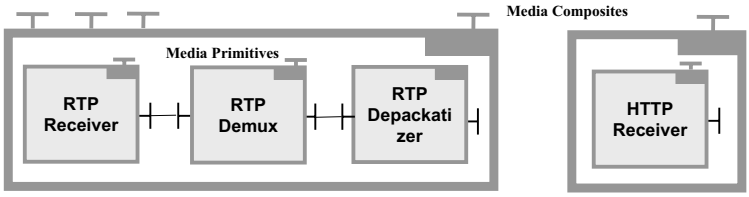

FIG. 2 – Exemples de composants média

- Media-Primitives (MP) : sont des composants primitifs qui implantent les traitements multimédia élémentaires comme un décodage MPEG, un encodage H.261 ou une transmission UDP.
- Media-Composites (MC) : sont des composites servant à définir des fonctions de plus haut niveau d'abstraction, comme le décodage, l'encodage ou la transmission. Chaque MC utilise un ensemble de MP pour réaliser une implantation spécifique de cette fonction. En tant que composant englobant, il est responsable de leur création, leur configuration et leur reconfiguration. La Figure 2 montre un exemple avec deux implantations d'un MC InputStream de réception de données. Le premier réalise un récepteur RTP avec trois MP : un RTPReceiver pour recevoir des données avec le protocole RTP, un RTPDemux pour la démultiplexage des flux et un RTPDepacketizer pour reconstituer les données multimédia originales. Le second implante un récepteur HTTP avec un seul MP : HTTPReceiver.
- Media-Session (MS) : est un composite qui encapsule une architecture de MC. Il représente l'application exécutable et fournit les opérations nécessaires à son contrôle.

Cette structuration permet de grouper les media primitives ayant le même type de fonction sous le contrôle du même composite. Ceci permet d'intégrer une gestion de configuration et de reconfiguration communes au niveau du composite. L'ensemble des composites constitue toutes les opérations qui peuvent être opérées sur l'application, et ceci indépendamment de sa structure.

4.2. Interaction entre les Composants Multimédia

La construction d'une application est réalisée en liant un ensemble de composants dans un même graphe qui représente la séquence de traitement des données. Un MP fournit un ensemble d'interfaces appelées *stream* pour recevoir/délivrer des données de/vers d'autres composants. Cette interface est typée par le format des données qui la traversent. Ce type est caractérisé par le type MIME (audio ou vidéo), le format d'encodage ainsi que des propriétés spécifiques (résolution, couleurs, etc.). Ainsi, chaque composant accepte les données dans un type particulier et produit des données dans un type particulier. La liaison entre deux interfaces de composants est donc conditionnée par la compatibilité entre les types de interfaces interconnectées. Ceci définit alors 2 types de liaisons, répondant à deux cas de figures :

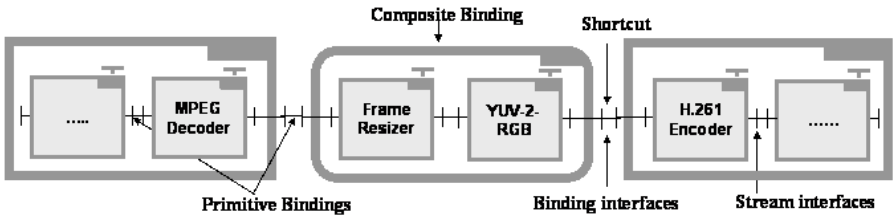

FIG. 3 – Exemples de liaisons

– *Liaisons primitives* : Les interfaces ont des types compatibles et peuvent être liées directement. Ceci signifie que le flux transite entre les composants par appels de méthodes directs entre leurs interfaces.
– *Liaisons composites* : Les interfaces ne sont pas compatibles, dans ce cas une liaison composite peut être établie afin de pallier cette incompatibilité. Une liaison composite est construite sous forme d'un MC formé d'un ensemble de MP qui réalisent les conversions adéquates. La Figure 3 montre un exemple de liaison composite entre un *Decodeur* et un *Encodeur*. Le décodeur fournit un flux vidéo en YUV alors que l'encodeur accepte seulement le format RGB et une résolution standard comme QCIF (176*144). En conséquence, la liaison composite implique deux MP : un Resizer qui transforme la résolution de la vidéo en QCIF et un YUV-2-RGB qui convertit la vidéo de YUV vers RGB.

Les *Media-Composites* exposent une seule interface en entrée et une seule interface en sortie. Contrairement aux MP, un MC ne participe pas dans l'échange des données. Au lieu de ceci, une liaison d'un MC est redirigée vers un de ses sous-composants. Ainsi, une liaison entre deux MC se traduit par une liaison entre deux MP (une extrémité de chaque de MC), évitant ainsi deux indirections inutiles (voir figure 3).

4.3. Composants d'observation : Probes

Les *Probes* sont des sondes d'observation qui peuvent être insérées à tout niveau de composition. Les données ne sont pas traitées par les probes mais seulement mis à disposition des composants voulant y accéder. En contrepartie, ils peuvent les retourner dans différentes échelles ainsi qu'appliquer les conversions inhérentes. Nous distinguons deux types de Probe :

– *QoS-Probes* : sont responsables de récupérer les informations de performances et de QoS mesurés par les composants de l'application. Par exemple, un composant RTPSender (d'émission réseau avec le protocole RTP) mesure continuellement le taux de perte de paquets, le délai de transmission, la cadence de transmission, etc.
– *Resource-Probes* : comme le nom l'indique, ces sondes sont chargées de récupérer des informations sur l'état des ressources de l'environnement d'exécution, comme la charge CPU, la consommation mémoire ou l'état de charge de la batterie. PLASMA fournit un ensemble de Resource-Probes définit dans la table 1.

Probe	paramètres observés
CPUProbe	Taux d'utilisation, temps systèmes, temps utilisateur, temps d'attente, etc.
MemoryProbe	Octets libres, Octets en cache, Pages entrée/sortie, etc.
NetworkProbe	bande passante, débit de transmission, débit de réception, taux d'erreur, nombre de connexion active, etc.
BatteryProbe	durée de vie, pourcentage restant, puissance restante, etc.
KernelProbe	Nombre de processus, Nombre de thread, nombre d'appels système, Nombre d'exceptions, etc.

TAB. 1 – Sondes d'observation de ressources dans PLASMA

4.4. Composants Sensors

Le rôle des composants *Sensors* consiste à détecter les moments de reconfiguration et de déclencher les événements conséquents. Nous distinguons deux types de sensors :

– *QoS-Sensor* et *Resource-Sensors* : il sont associés respectivement aux QoS-Probes et Resource-Probes afin de détecter les changements pertinents dans les paramètres observés. La comportement implanté par ce type Sensor est générique : il consiste à comparer les valeurs observées avec des seuils prédéfinis, et de générer des événement lors de chaque violation.
– Sensors externes : permettent de capturer des événement externes. Ils peuvent être utilisés pour des besoins très variés et nécessitent des implémentations spécifiques. Par exemple, un sensor peut jouer

le rôle d'un serveur de vidéoconférence afin de signaler les événements concernant : l'arriver/départ d'un utilisateur, la fin de la conférence, etc.

4.5. Composants Actuator

Les Actuators réagissent aux événements de reconfiguration en opérant les reconfiguration sur les différents composants de l'application. Comme évoqué au début de cet article, une telle opération corresponds à un changement d'attributs du composant cible et s'exécute à travers son interface Attribute-Controller. Par conséquent, le comportement d'un Actuator est indépendant du composant cible et peut être réalisé de façon générique. Généralement, chaque composant associe une sémantique à ses attributs qui déterminera l'exécution effective de la reconfiguration. Ainsi, nous distinguons trois formes de reconfiguration :

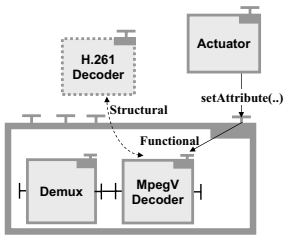

FIG. 4 – Exemples reconfigurations : fonctionelle et structurelle

– Reconfiguration fonctionnelle : qui consiste à modifier un attribut fonctionnel d'un composant primitif afin d'ajuster son comportement convenablement. Comme le montre la figure 4, un changement de la qualité d'encodage ou de la cadence des images est délégué par le MC à un sous-MP MpegV-Decoder, qui effectuera le changement effectif. Bien qu'une telle opération n'affecte qu'un seul composant, son impact sur le type de données produit par celui-ci distingue deux cas de figures :
  – Dans le premier, l'opération n'affecte pas le type des données et donc, elle s'effectue sans interruption de l'application.
  – Dans le deuxième cas, elle modifie le type de données et par conséquent, elle nécessite le re-établissement des liens entre une partie des composants (les composants qui suivent le composant cible dans le graphe), ce qui implique l'arrêt de l'application.

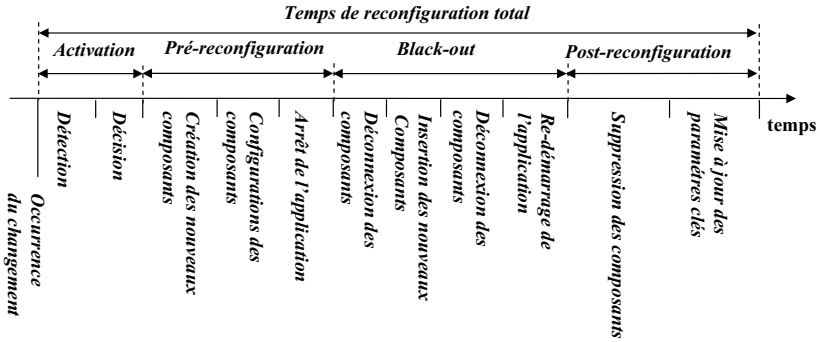

FIG. 5 – Diagram d'execution d'une reconfiguration structurelle

– Reconfiguration structurelle : correspond à la modification de la structure interne du composite par l'ajouter/suppression de composants ou la modification des liaisons entre ces sous-composants. De manière générale, cette opération s'effectue en quatre étapes, schématisées sur la figure 5 :
  – L'étape d'*activation* inclut la détection d'un changement et l'envoi d'un événement correspondant.
  – L'étape *pre-reconfiguration* inclut toutes les opérations pouvant s'exécuter avant l'interruption de l'application, parmi elles : la création des nouveaux composants et leur configuration. Ceci permet de retarder l'arrêt de l'application.
  – L'étape *blackout* représente l'intervalle de temps pendant lequel l'application est arrêtée. Ceci est nécessaire pour l'insertion de composants et l'établissement de leurs liaisons, etc.
  – L'étape *post-reconfiguration* regroupe les opérations pouvant s'exécuter après la reconfiguration, par exemple, la suppression des composants, la mise à jour des données clés, la désactivation de composants, etc.
– Reconfiguration de politiques : Les reconfigurations peuvent également viser les politiques de reconfiguration. Celles-ci se traduisent par la modification de paramètres des Probes, Sensors ou Actuators. Par exemple, le changement de la période d'observation d'un Probe, du seuil de déclenchement d'un événement dans un Sensor, ou des valeurs de l'action de reconfiguration dans un Actuator.

5. Implémentation et Evaluation des Performances

Nous avons réalisé une implantation de PLASMA en C++ sous le système d'exploitation Windows 2000 et l'environnent de développement Microsoft Visuel .Net 2003. Les fonctions multimédia reposent sur la plateforme Microsoft Direct Show dans sa version 9.0. Ce premier prototype couvre une implantation C++ du modèle Fractal conforme à la spécification de référence. Par ailleurs, les fonctions d'observations de système font partie de LeWYS [3], un canevas logiciel d'observation de systèmes. Divers scénarios d'applications ont été menés pour l'évaluation et la validation de PLASMA, parmi eux, une application de transmission vidéo pour des terminaux mobiles. Les sous-sections suivantes décrivent ce dernier et présentent ensuite une évaluation des performances sous divers aspects.

5.1. Architecture d'expérimentation

L'architecture d'expérimentation, schématisée sur la figure 6, s'intéresse à la transmission vidéo adaptative pour des terminaux mobiles. Ces derniers sont matérialisés par des PDA[1] limités à l'encodage MPEG via le protocole HTTP. Ces derniers sont susceptibles d'accéder à divers services multimédia tels qu'un serveur de Vidéo à la demande, une diffusion d'un contenu audio ou vidéo en temps réel. Ces derniers emploient différents protocoles de sessions (HTTP, RTSP, etc.), de transmission (UDP, RTP ou TCP), de formats d'encodage (MPEG-x, H.26x, MJPEG, etc.) avec différentes propriétés en terme de débit d'encodage, résolution, nombre de couleurs, etc. Pour palier les limitations des terminaux, des nœuds SPIDER interviennent dans la communication en appliquant les adaptations nécessaires entre le service multimédia fournit par le serveur et le terminal qui accède à ce service (transformations des flux audio et vidéo). En bénéficiant de la flexibilité de PLASMA, SPIDER se distingue des travaux menés dans ce domaine, par une architecture dynamique sans aucune hypothèse sur les capacités des terminaux. Ce dynamisme réside dans le fait que les terminaux sont capables de piloter un noeud SPIDER en spécifiant leurs préférences (en terme de propriétés du flux) ainsi que les politiques d'adaptation applicables en cas de changement (adaptation de l'application).

5.2. Scénario

Notre évaluation se base sur le scénario suivant : un terminal souhaite accéder à un service de diffusion TV temps réel et demande initialement un encodage MPEG avec une résolution à 320x240 pixels à 25 images/seconde. Par ailleurs, une surcharge du processeur (CPU) du terminal peut causer des pertes de données au niveau de l'application, ce qui influe négativement sur la qualité de la présentation. Cette surcharge peut être, d'une part, la conséquence d'un débit de données élevés. D'autres part, elle peut être due à une résolution des images du flux vidéo ; en effet, la limitation des capacités d'affichage oblige l'application de redimensionner les images avant l'affichage. Sachant que ces deux paramètres

[1] Avec le modèle Compaq IPaq H.3970 doté d'un processeur XScale à 400 Mhz et 64Mo de mémoire, connectés à Internet via une liaison sans fils 802.11b à 11 Mbps.

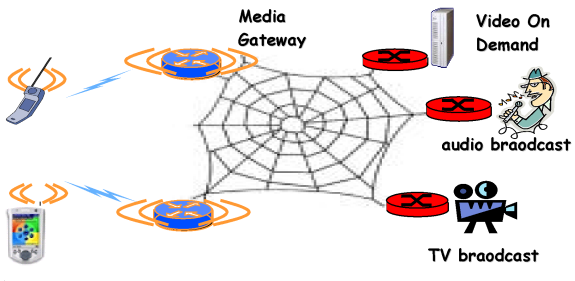

FIG. 6 – Architecture d'expérimentation.

sont fortement corrélés, l'application demande une politique de contrôle de débit qui consiste à réduire la résolution de 5 % quand le débit dépasse 512 Kbps et de l'augmenter de 5 % quand il baisse en dessous de 256 kbps. Elle est évaluée périodiquement (chaque 10 secondes) et va nous permettre d'observer le comportement du terminal dans différents états. Pour déployer un tel service, le terminal envoi une requête HTTP GET comportant la description ADL données sur la figure 7. Celle-ci décrit la structure de l'application (en termes de MC) ainsi que la politiques d'adaptation précédente en termes d'observations et d'actions de reconfiguration. Ainsi, le processus de transformation adéquat est créé par composition de composants PLASMA.

```
<TaskFlow id="Server" location="oxygene.inria.fr">
 <Task name="Input-Stream" id="C">
  <Attributes signature="InputAttributeController">
   <Attribute name="src" value="rtp://ozone.inria.fr:5000"/>
  </Attributes>
  <Binding id="b1" client="this" server="O" />
 </Task>
 ....
 <Task name="Video-Encoder" id="E">
  <Attributes signature="EncoderAttributeController">
   <Attribute name="format" value="32" />
   <Attribute name="quality" value="80" unit="%" />
  </Attributes>
  <Binding id="b4" client="this" server="O"/>
 </Task>
 <Task name="Output-Stream" id="O">
     ...
 </Task>
 <Observation id="ob1" type="Resource"
             resource="NetworkProbe@bandwidth">
  <event id="evt1" operator="exceeds" value="256"
             unit="kbps"/>
  <event id="evt2" operator="exceeds" value="512".../>
 </Observation>
 <Action-set id="set1" condition="evt1">
  <Action operation="decrease" target="id(E)@quality"
              operand="5" unit="%" status="active"/>
 </Action-set>
 <Action-set id="set1" condition="evt2">
  <Action operation="increase" .../>
 </Action-set>
</TaskFlow>
```

FIG. 7 – Description ADL d'un processus de transcodage.

5.3. Impact sur la QoS et les performances du terminal

La première évaluation concerne l'impact de reconfigurations sur la qualité de la présentation ainsi que l'utilisation des ressources du terminal. Pour cela, nous avons mesuré la cadence d'affichage de la vidéo

et le taux d'utilisation du processeur sur le terminal durant les différentes étapes de l'expérimentation. Les résultats sont donnés sur la figure 8, les lignes verticales pointillées représentent un changement de la résolution provoqué par l'occurrence d'une reconfiguration sur le noeud SPIDER. Ces opérations distinguent quatre phases :

– Durant la première phase (de 0 à 30 secondes), le flux a les propriétés de configuration initiales. Le terminal nécessite une charge CPU à 100 % ce qui donne une cadence d'affichage inférieur à 10 images/seconde. Ceci est causé principalement par la résolution du flux vidéo qui compliquent le processus de décodage et d'affichage et causent ainsi des pertes de données.
– La deuxième phase commence après l'exécution de la première reconfiguration (seconde 30) et qui réduit la résolution à 304x228. Ceci réduit la complexité du traitement sur le terminal qui atteint une cadence de 17 images/seconde. Cependant, la charge CPU reste à 100 % pour les mêmes précédentes raisons.
– Une deuxième reconfiguration (seconde 40) diminue la résolution à 288x216 et réduit ainsi la charge CPU à 85 %. La cadence d'affichage atteint 23 images/seconde, la différence par rapport au débit originale étant causée par des pertes due au débit de la vidéo.
– En effet, ceci cause une dernière reconfiguration qui diminue encore la résolution à 273x205. Ceci réduit la charge CPU à 70 % et permet ainsi d'atteindre la cadence d'encodage originale de 25 images /seconde.

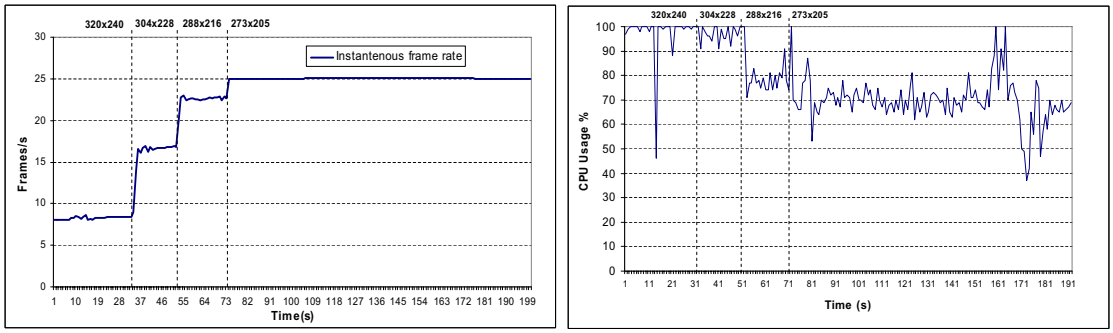

FIG. 8 – Evolution des performances de l'applications.

5.4. Coût des reconfigurations

Reconfiguration structurelle : La première exécution du précèdent algorithme se traduit par une reconfiguration structurelle de l'application par l'addition d'un composant Video-Resizer afin de réduire la résolution de la vidéo. Comme expliqué précédemment, cette opération s'exécute en plusieurs étapes ; on s'intéresse au trois dernières : pre-reconfiguration, blackout, post-reconfiguration. Le tableau 2 donne le temps moyen de cette opération ainsi que sa distribution sur ces étapes. Le temps total est d'environ 35,9 ms (milli-secondes) où la majeure partie est consacrée à l'étape de pre-reconfiguration, qui prend 24,1 ms pour l'instanciation du composant Video-Resizer. Le temps de blackout est de 11,6 ms, nécessaire pour l'insertion et la liaison de ce dernier. L'étape post-reconfiguration dure 0,2 ms. On retient de ces résultats que l'exécution de l'étape de pre-reconfiguration réduit significativement le coût d'une reconfiguration structurelle, dans notre cas par 68 % supposant qu'une exécution naïve aurait exigé l'arrêt de l'application pendant le temps total.

Reconfiguration fonctionelle : Les reconfigurations subséquentes sont effectuées par des reconfigurations fonctionnelles du composant Video-Resizer. Cependant, comme celles-ci affectent le type du média établi lors de la connexion de ce dernier, elle nécessite le re-établissement de ses liaisons (ainsi que celles des composants qui le suivent) et donc l'arrêt de l'application. Le paramétrage du composant dure 67 μs (micro-secondes) et le temps d'arrêt total 331 μs, avec 244 μs consacrés à la gestion des liaisons entre les

composantes. En dépit de cet arrêt, le temps total de reconfiguration reste limité à 398 μs et n'introduit pas un coût significatif.

Pre-reconfiguration	Instantiation	Configuration	Total
	24 ms	0,10 ms	24,1 ms
Blackout	Configuration	Liaison	Total
	11 ms	0,60 ms	11,6 ms
Post-reconfiguration	-	-	0,20 ms
Total	-	-	35,90 ms

TAB. 2 – Timing of Structural Reconfiguration.

6. Conclusion

Cet article a présenté PLASMA, un canevas à composants pour la construction d'applications multimédia auto-adaptables. L'objectif primordial consiste à fournir une solution suffisamment générale pour cibler divers type d'applications ainsi que leurs besoins en terme d'adaptabilité. Vers cet objectif, PLASMA exploite des propriétés fortes de son modèle à composants : composition et reconfiguration hiérarchiques et langage de description d'architecture (ADL) dynamique. L'étude expérimentale a prouvé que l'utilisation d'une technologie à composants n'introduit pas un surcoût significatif. D'autres part, elles per mettent une amélioration significative des performances des applications ainsi que sur l'utilisation des ressources matérielles, en particulier sur des terminaux mobiles avec des ressources limitées. Cette étude expérimentale nous permet d'identifier deux besoins qui représentent une perspective intéressante de PLASMA : la supervision et la reconfiguration distribuées :

– En effet, dans le scénario précédent, un algorithme plus efficace serait basé sur l'observation de la charge CPU sur le terminal ainsi que du comportement de l'application (e.g. cadence d'affichage, taux de pertes, etc.). Ceci permet en effet de détecter plus efficacement une baisse de la QoS ainsi que la surcharge du terminal.
– Le besoin d'opérations de reconfiguration distribuées apparait lorsqu'une reconfiguration d'un tier du système (e.g. serveur ou client) implique également une reconfiguration sur les autres tiers. Par exemple, un changement de l'encodeur sur un noeud SPIDER nécessite le changement du décodeur sur le terminal. Pour réaliser de telles opérations, une application donnée doit avoir un contrôle explicite sur les autres applications du système afin d'y opérer des reconfigurations.

Bibliographie

1. G. Blair, L. Blair, V. Issarny, P. Tuma, A. Zarras. The Role of Software Architecture in Constraining Adaptation in Component-based Middleware Platforms. Middleware Conference, April 2000.
2. E. Bruneton, T. Coupaye, M. Leclercq, V. Quéma and J-B. Stefani. An Open Component Model and its Support in Java. International Symposium on Component-based Software Engineering, May 2004.
3. E. Cecchet, H. Elmeleegy, O. Layaida, V. Quéma. Implementing Probes for J2EE Cluster Monitoring. OOPSLA Component and Middleware Performance Workshop, October 2004.
4. D. Chefrour, F. André. Auto-adaptation de composants ACEEL coopérants. 3'eme Conférence Fran¸caise sur les Systèmes d'Exploitation (CFSE'3), La Colle sur Loup, France, October 2003.
5. L.S. Cline, J. Du, B. Keany, K. Lakshman, C.Maciocco, D.M. Putzolu. DirectShow RTP Support for Adaptivity in Networked Multimedia Applications. IEEE International Conference on Multimedia Computing and Systems, 1998.
6. D. Duke and I. Herman. A Standard for Mulimtedia Middleware. ACM International Conference on Multimedia. 1998.

7. T. Fitzpatrick and al. Design and Application of TOAST : An Adaptive Distributed Multimedia Middleware. International Workshop on Interactive Distributed Multimedia Systems, 2001.
8. X. Fu and al. CANS : Composable, adaptive network services infrastructure, USITS 2001.
9. V. Hardman and al. Reliable Audio for Use over the Internet, INET'95.
10. Microsoft : DirectShow Architecture. http ://msdn.microsoft.com/directx 2002.
11. Sun : Java Media Framework API Guide. http ://java.sun.com/products/javamedia/ jmf/ 2002.
12. B. Li and K. Nahrstedt, A Control-based Middleware Framework for Quality of Service Adaptations, IEEE Journal on Selected Area on Communications (JSAC), 17(9), 1999.
13. O. Layaida, S. Ben Atallah, D. Hagimont. Adaptive Media Streaming Using Self-reconfigurable Proxies. In Proceedings of the 7th IEEE International Conference on High Speed Networks and Multimedia Communications (HSNMC'04), Toulouse, France, June 30-July 02, 2004.
14. M. Lohse, M. Repplinger, P. Slusallek, An Open Middleware Architecture for Network-Integrated Multimedia, Joint IDMS/PROMS workhop 2002.
15. S. McCanne and V. Jacobson. VIC : A flexible framework for packet video. ACM Multimedia Conference, 1995.
16. Z. Morley and al. Network Support for Mobile Multimedia using a Self-adaptive Distributed Proxy. In Proceeding of Network and Operating System Support for Digital Audio and Video, NOSSDAV-2001.
17. H. Schulzrinne and al. RFC-3550 RTP : A Transport Protocol for Real-Time Applications, 2003.
18. D. Wichadakul, X. Gu, K. Nahrstedt, A Programming Framework for Quality-Aware Ubiquitous Multimedia Applications. In Proceedings of ACM Multimedia Conference 2002.
19. D.G.Waddington and G.Coulson, A Distributed Multimedia Component Architecture, 1st International Workshop on Enterprise Distributed Object Computing, October 1997.