LIUPPA Année 2007

Thèse

Modèles de Flux et de Composants pour Applications Multimédias Distribuées Dynamiquement Reconfigurables

Présentée devant
Université de Pau et des Pays de l'Adour

Pour obtenir
le grade de Docteur en Sciences de l'Université de Pau et des Pays de l'Adour

Spécialité : **INFORMATIQUE**

Par
Emmanuel Bouix
Soutenue le Jeudi 29 Novembre 2007 devant la Commission d'examen

Jury

Président	Jean-Christophe Lapayre	Professeur – UFC Besançon
Rapporteurs	Didier Donsez	Professeur – UJF Grenoble
	Daniel Hagimont	Professeur – ENSEEIHT Toulouse
Examinateurs	Marc Dalmau	Maître de Conférence – UPPA Bayonne
	Stéphane Frénot	Maître de Conférence – INSA Lyon
	Franck Luthon	Professeur – UPPA Bayonne
	Philippe Roose	Maître de Conférence – UPPA Bayonne

Résumé étendu

Nos travaux antérieurs ont permis de définir une méthode globale pour la conception et le développement des applications multimédias distribuées. Ces recherches centrées sur la qualité de service permettent le respect des exigences rigoureuses que la couche réseau de l'Internet ne considère pas. En effet, ils prennent en compte la qualité requise par les utilisateurs ainsi que celle amenée par l'environnement d'exécution. L'utilisation de ces applications dans ce type d'environnement est compromise dans certains cas par les caractéristiques mouvantes et non prédictibles (bande passante des liaisons réseaux, caractéristiques des terminaux, fonctionnalités des systèmes d'exploitation mais aussi vœux des utilisateurs, etc.). Nous proposons une solution qui se divise en deux parties. La première partie définit un modèle complet de gestion de la qualité de service ainsi que la définition d'une plate-forme chargée de cette gestion. La seconde partie qui intéresse cette thèse a pour but de définir l'architecture logicielle adaptée à la conception et au développement de ces applications. Cette architecture devra pouvoir être manipulée par une plate forme chargée de la supervision de l'exécution avec pour finalité la gestion de la qualité de service. Ainsi, l'architecture globale d'une application se divise en deux parties :

- une partie applicative qui implémente les fonctionnalités des applications ; nous proposons une architecture logicielle complète chargée de leur implémentation ;
- une plate forme qui permet de superviser et de gérer l'exécution de la partie applicative en tenant compte des exigences des utilisateurs et de l'environnement d'exécution ; cette tâche de supervision est réalisée de manière dynamique.

Les spécifications des applications multimédias sont décrites à l'aide de graphes fonctionnels. Le principe utilisé est de décomposer une application en rôles ou fonctionnalités atomiques. Ces graphes sont polaires et orientés $G(V, E_S, E_c)$. Ainsi, les fonctionnalités atomiques sont décrites par les nœuds du graphe (ensemble V). Les arcs (ensemble $E_s \cup E_c$) représentent les médias échangés par ces fonctionnalités atomiques. Deux types d'arcs se distinguent alors :

- les arcs simples – ensemble E_s – permettent de spécifier que le nœud où aboutit un arc appartient à l'application quelle que soit la configuration ;

- les arcs conditionnels – ensemble E_c – permettent de modéliser les choix de configuration possibles et les contraintes liées à l'utilisation d'une configuration plutôt qu'une autre.

Ces graphes permettent de décrire l'architecture abstraite d'une application comme une connexion de fonctionnalités atomiques à l'aide de médias où les fonctionnalités manipulent et traitent des médias, plus généralement des données.

La première tâche qui nous incombe alors est de définir comment les médias sont manipulés par une telle architecture, en d'autres termes comment les fonctionnalités vont échanger des informations. Dés lors que l'on manipule ce type de données, il est nécessaire de considérer certaines de leurs propriétés qui s'avèrent cruciales pour préserver leur sémantique. En effet, certains médias intègrent des propriétés de séquence et des relations temporelles plus connues sous le nom de relations de synchronisation. Ces relations se rencontrent à deux niveaux, entre les données qui constituent un même média mais aussi entre celles de plusieurs médias, elles se nomment respectivement intra- et inter-médias. Les graphes fonctionnels permettent de spécifier les relations de type inter-médias à l'aide de liens de synchronisation. Une étude approfondie des applications existantes nous a permis d'identifier deux sources de perte possible de ces relations. Il s'agit du traitement et du transport des médias dans les applications. La première se produit lorsque plusieurs médias sont liés par des relations de synchronisation inter-médias et que certains subissent des traitements. Ces traitements sont susceptibles de modifier ces relations lorsqu'ils introduisent des retards sur les médias traités par rapport à ceux qui ne le sont pas. La seconde source se produit lorsque l'on transfère des médias par un réseau de communication. En effet, les services utilisés pour le transport des données introduisent des délais, de la gigue et même parfois des pertes de données. Afin de pallier à ces désagréments, nous fournissons un modèle unifié qui va permettre une manipulation uniforme des données dans les applications multimédias ainsi que la prise en compte de leurs principales caractéristiques.

Les médias existent sous différentes formes. On distingue, en particulier, les médias continus (audio, vidéo, etc.) et les médias discrets (texte, graphiques, images, etc.). Ces deux types se différentient sur le concept de temps et la manière dont ils sont structurés. Cette constatation implique différentes façons de les manipuler et donc de connaître a priori le type des médias que l'on veut utiliser. De plus, il est possible que des médias appartenant à ces deux types différents possèdent des relations de synchronisation inter-médias, par exemple vidéo et sous-titres. Afin de rendre possible leur utilisation, nous spécifions un modèle de données unique que l'on utilisera dans les applications pour tous les types de médias et au-delà pour tous les types de

données que les fonctionnalités sont susceptibles de s'échanger. Ce modèle doit donc intégrer les propriétés de séquence et les relations temporelles. Nous proposons que les informations existent sous la forme de flux de données. Cette structure est intéressante car elle permet de tenir compte de ces propriétés. Les médias continus sont déjà sous cette forme. Ainsi les médias discrets et autres données seront manipulés également sous la forme de flux, on va de la sorte les séquencer et leur ajouter le facteur temps. L'avantage de notre démarche est de pouvoir manipuler toutes les données de la même façon et donc de simplifier l'architecture des applications. Ainsi, les relations temporelles entre des données de différents types pourront être conservées.

Ces flux de données sont qualifiés de synchrones, ils sont définis comme une suite infinie de tranches synchrones. Une tranche synchrone est un objet qui permet de rassembler une quantité d'informations qui correspond à un même intervalle de temps. Suivant la constitution de ces dernières, on peut distinguer deux sortes de flux synchrones. Lorsqu'elles intègrent des données issues d'un même flux de données (au sens conceptuel du terme) alors on parle de flux synchrones primitifs. Lorsqu'elles rassemblent des données issues de plusieurs flux de données, on parle de flux synchrones composés. L'intérêt des flux composés est de rassembler dans une même structure des flux possédant des relations de synchronisation inter-médias et de permettre de les conserver pendant le traitement et le transfert de ces flux dans les applications.

Grâce à ces flux de données, nous donnons la possibilité de conserver la séquence et les relations temporelles en estampillant les données à l'aide de numéros de séquence et d'étiquettes temporelles délivrés respectivement par une horloge logique et une horloge physique locale. Ainsi, à l'aide de ces estampilles il nous est possible de définir des relations d'ordre strict et total entre les données d'un même flux mais aussi entre les tranches synchrones de flux synchrones. Grâce à cette approche, nous pouvons désormais manipuler les concepts de valeur temporelle, d'intervalles temporels, etc. De plus, la restitution synchrone des médias est rendue possible.

Dans notre approche, tous les médias étant rendus temporels, une question de fond se pose alors quant à l'importance de cet aspect en fonction des médias. Par exemple, un média qui intègre des données toutes les minutes ne présente pas les mêmes contraintes qu'un média qui fournit des données toutes les secondes. Nous proposons donc de les classer en fonction de ces contraintes temporelles. Ainsi, nous divisons les flux synchrones en deux catégories : ceux à contrainte temporelle faible et ceux à contrainte temporelle forte. De par leur nature, les médias continus sont considérés comme à contrainte forte quand les relations qu'ils possèdent sont strictes et se doivent d'être conservées si l'on ne veut pas dégrader leur sémantique. Suivant les cas,

les médias discrets peuvent être considérés comme à contrainte temporelle forte ou faible. Les contraintes temporelles faibles sont moins strictes que les autres, elles permettent une plus grande marge de manœuvre et leur non respect est beaucoup moins perceptible et dommageable à la sémantique des données. Cette distinction doit être faite lors de la phase de conception des applications en fonction des médias que l'on désire intégrer. Elle est réalisée en définissant une durée maximale entre deux tranches synchrones dans un flux au dessous de laquelle le flux est considéré comme étant à contrainte forte.

Enfin, ce modèle nous amène à définir des politiques de synchronisation afin de rassembler dans des flux composés les flux de données qui possèdent des relations de synchronisation entre eux. Ces politiques se basent sur tous les aspects et propriétés définis par le modèle. Ainsi, en fonction des contraintes temporelles des flux à synchroniser on va former les tranches synchrones des flux composés. Ces politiques s'appliquent aux flux synchrones et permettent d'obtenir des flux synchrones composés.

L'utilisation d'un modèle de données et les spécifications des applications multimédias à l'aide des graphes fonctionnels nous conforte dans la conception d'un modèle de composants logiciels permettant le développement de ce genre d'applications. Ces modèles nous permettent de déduire les spécifications dont nous devons tenir compte pour proposer un modèle de composants. Ainsi, les nœuds qui décrivent les fonctionnalités atomiques seront implémentés par des composants logiciels qui seront connectés entre eux par des connecteurs chargés de transporter les médias à travers les applications conformément au modèle décrit dans la section précédente.

Nous identifions deux types de composants : les composants fonctionnels et les composants non-fonctionnels. Les composants fonctionnels sont chargés de l'implémentation des fonctionnalités atomiques de l'application. Elles sont identifiées par les graphes fonctionnels. Ils consistent le plus souvent en des composants de création ou acquisition de médias, des composants de traitements et des composants de restitution ou de stockage. Les composants non-fonctionnels sont chargés d'implémenter les aspects qui ne concernent pas directement les fonctionnalités d'une application mais qui sont nécessaires pour fournir une implémentation conforme aux spécifications. Par exemple, les politiques de synchronisation décrites par le modèle de flux seront réalisées par ce type de composants pour produire un flux synchrone composé. Un composant réalisant la fonction inverse permettra de séparer des flux afin par exemple de les envoyer vers différentes destinations. De même, les graphes fonctionnels acceptent des disjonctions qui permettent le traitement ou la manipulation

parallèle de plusieurs flux. Ces disjonctions sont réalisées également à l'aide de composants non-fonctionnels.

Les composants fonctionnels sont nommés composants métier puisqu'ils recueillent le code applicatif. Les applications comportent un tel composant par fonctionnalité atomique identifiée sur les graphes. Nous les qualifions de métier car les aspects qu'ils adressent sont relatifs au domaine du multimédia. Ces composants sont capables de lire et d'écrire dans des tranches synchrones issues des flux synchrones. En raison de la synchronisation inter-flux, ils recevront donc les tranches synchrones des flux qu'ils traitent ainsi que d'autres auxquels ils ne touchent pas et donc qui doivent seulement transiter dans le composant. Cette solution nous permet de ne pas détruire les relations de synchronisation inter-flux. De plus, le cycle de vie du composant métier est supervisé par la plate-forme qui peut à tout instant de l'exécution d'une application procéder à des reconfigurations dynamiques en vue de la gestion de la qualité de service. Ces services ne sont pas directement liés aux fonctionnalités atomiques, ils font partie de l'implémentation non-fontionnelle. En ce sens, nous prévoyons d'utiliser un conteneur pour les composants métier dont le but est d'accueillir chacun d'eux (un composant par conteneur) et de leur fournir un environnement d'exécution permettant de gérer les interactions avec l'extérieur, les flux de données d'une part et la plate-forme d'autre part. Ainsi, un tel conteneur se divise en deux parties. L'unité de contrôle est chargée de la gestion du cycle de vie du composant métier et de son interaction avec la plate-forme d'exécution. L'unité d'échange se charge de la gestion des connexions en entrée et en sortie du composant métier. Le conteneur de composant métier est appelé le processeur élémentaire.

Les médias sont transportés au sein de l'application à l'aide d'un connecteur que l'on a appelé le conduit. Les conduits sont utilisés pour connecter les composants métier (et donc les processeurs élémentaires) entre eux mais aussi les composants non-fonctionnels. Leur rôle est de recevoir une à une les tranches synchrones produites par les composants et de les acheminer. Le conduit doit également être manipulable par la plate-forme : l'unité de contrôle qui est chargée de la supervision du conduit par la plate-forme et l'unité d'échange gère les connexions d'entrée/sortie. Le conduit constitue l'entité distribuée de notre modèle c'est-à-dire qu'il est capable de transférer des données entre les différents sites d'une application distribuée.

Remerciements

Ce manuscrit est dédicacé à toutes les personnes qui m'ont aidé et soutenu, de près ou de loin, pendant ces quatre dernières années.

Le travail présenté dans ce mémoire est le résultat des recherches menées au Laboratoire d'Informatique de l'Université de Pau et des Pays de l'Adour de septembre 2003 à septembre 2007. Démarrer une thèse consiste à débuter un long travail, et ce travail ne peut être effectué sans guide. En tout premier lieu, je tiens à remercier sincèrement Marc Dalmau et Philippe Roose pour m'avoir accueilli et m'avoir encadré pendant cette thèse. Je les remercie également pour la confiance accordée tout au long de ces années. Vos nombreux conseils m'ont été précieux tant au niveau de l'enseignement que de la recherche. Je vous remercie de m'avoir donné la chance d'enseigner au département informatique pendant quatre années. Je dois aussi saluer votre patience surtout lors du rush final où le temps passe à une vitesse… Merci pour le partage de vos connaissances et pour votre accompagnement tout au long de ce travail malgré vos tâches respectives de responsables de formations à l'IUT de Bayonne. Grâce à vous, j'ai vécu des expériences qui m'ont énormément apporté, tant sur le plan scientifique que sur le plan humain. A Marc pour nos nombreuses discussions sur le Rock'n'Roll et à Filou pour nos délectables escapades nocturnes avec le guide du broutard, excellente association dont la devise est : « Le bonheur, c'est simple comme un coup de fourchette ». Merci pour eux !

J'en profite également pour remercier Franck Luthon qui a accepté de diriger ma thèse. Je le remercie pour ces conseils et son suivi pendant toutes ces années. N'étant pas issu de la communauté informatique, j'ai trouvé son avis et sa critique fort intéressants. Notre collaboration ne s'arrête pas là car tu as eu l'amabilité de me proposer un poste d'enseignant pour l'année universitaire 2007-2008 au sein du département GIM de l'IUT de Bayonne fraîchement créé.

Je suis très honorée que Didier Donsez, Daniel Hagimont et Jean-Christophe Lapayre aient accepté de rapporter ce travail et donc de faire partie de mon jury de thèse.

Physiquement parlant, ma thèse s'est déroulée au département informatique de l'IUT de Bayonne : « au château… ». Bien sûr, je ne peux oublier l'ensemble du personnel du château pour la convivialité dont ils ont fait preuve. Merci à l'IUT de Bayonne pour m'avoir permis tant de rencontres exceptionnellement riches, dans une ambiance très sympathique… La liste est longue, aussi je vais essayer de n'oublier personne. Merci au PPP : Patrick (voisin de bureau gourmand de gauche), Pantxika (voi-

sine moins gourmande mais tout aussi agréable de droite) et Philippe (Lopis le voisin du dessous). Philippe (Gaba Gaba Hey!) pour nos longues discussions sur le Punk rock et la scène alternative. Christophe, le souletin bout en train. Jean-Jacques pour les escapades gourmandes du samedi matin, les champignons … bonne retraite à toi, je suis sûr que tu ne vas pas t'ennuyer. Léo (« …ça vous dirait d'être bastidots… »), Simone, Peio, Corinne, Ginette, Ghislaine, Caroline, Guy, Gilles, etc. Je n'oublie pas les retraités du département : Jacques (« The shit is deep »), Bernard et Gérard et aussi les expatriés de Darrigrand : Gérard, Manu (« … on boit un coup … ») et Mitch (échapéééé). Merci Manu pour les impressions de ce mémoire au dernier moment comme d'habitude… Une pensée pour les CTI boys : Pierre, Jean-Marc, Kevin, Serge et Gilles. Merci aux camarades de thèse et de tant d'autres, pour avoir partagé les mêmes difficultés, les mêmes motivations, et les mêmes combats durant toutes ces années de recherche, en particulier Marion (tu vois j'y suis arrivé à finir un an après toi…), Christine (ma colloc de bureau et ses délicieux gateaux, vivement le prochain !), Francis False, Eric, Brice, Sophie, etc. Une pensée pour ceux qui sont également passés au château : Fred, Clémentine, Marc, Anis, etc.

Mes remerciements se tournent également vers les autres membres du LIUPPA pour m'avoir accueilli au sein du laboratoire.

Ces quatre ans de thèse ont été aussi pour moi l'occasion d'enseigner. Je remercie l'IUT de Bayonne pour m'avoir confié cette responsabilité. J'ai aussi une pensée pour tous les étudiants que j'ai eu et avec qui j'ai eu un énorme plaisir à travailler.

Evidemment, je ne peux clore ces quelques lignes sans une pensée pour mes amis. Les agenais, les expatriés : Mathieu (Vallée FM), Nico, Jean-Philippe (l'Aiglon), Damien, Julie, Myriam, Ludo, Armelle, Hugues, Armelle (c'est pas la même), Boul, Stuph, Marion, Mike, Nico (stay sick), Julien, Laurence, Fred, Papi, Pierre, Caroline, François, Caroline (encore une autre), Pascal (ça y est j'ai enfin fini), Den's, Emilie, David, Mathilde, Thomas, Estelle, le Couleur Café (the place to be…), Hicham, Khalil, Valmi, Drix, Arantxa et les bidartars.

Enfin, le meilleur pour la fin, j'aimerais témoigner ma plus profonde reconnaissance à mes parents et à mon frère pour avoir crû en moi et m'avoir soutenu pendant ces quatre ans. Une pensée pour mon grand père qui pète la forme à 90 ans. D'autres membres de ma famille : Joëlle Rodolphe, Martin le p'tit homme, Jérôme, Lucette, Jean-Pierre, Françoise, Serge, David, Stéphane, Ludo, etc.

Pour toutes les personnes que j'ai oubliées, je ne l'ai pas fait exprès, mais merci à vous.

Bayonne le 13/09/07.

Table des Matières

Liste des Figures

Liste des Tables

I'm walkin' out for love

I'm walkin' bad really down

Like a cool breeze

I'm gonna be late again

"Driver! Wait for me please!"

I'm runnin' all in vain

Tryin' to catch this fuckin' train

"Time don't fool me no more"

And I throw my watch to the floor

(It's so lazy)

"Time don't do it again"

Now I'm stressed and strained

With anger and pain

In the subway train

Now it's half past two

Long gone the rendez-vous

Now it's half past three

Time made a fool out of me

Now it's half past four

No use in waiting no more

Baby can't you see

It's a timing tragedy

I think it's nine

When clock says ten

This girl wouldn't wait

For the out of time

OUT OF TIME MAN

Out of Time Man, Manu Chao (Mano Negra), 1991

A Aimée et Lucienne

Introduction

« Pour chaque problème, il y a une solution

Moi, je sais où se cache la clé de tous les secrets. »

Rejoins moi, Jean-Michel Poisson (Les Naufragés), 1992

Les applications multimédias distribuées se composent d'un ensemble de traitements appliqués sur plusieurs médias. Le caractère distribué implique que ces médias peuvent être transmis à l'aide d'un réseau de communication. Ils constituent des moyens de représentation de l'information basés sur les propriétés psychosensorielles humaines. Cette particularité fait qu'ils intègrent fréquemment des propriétés de temps et de séquence. Plus précisément, ces propriétés se traduisent par des relations de synchronisation qu'il est nécessaire de considérer si l'on veut donner un sens à l'information transportée. Elles existent à deux niveaux : entre les informations qui composent un même média et entre celles qui composent plusieurs médias. De plus, l'utilisation des médias à travers des réseaux de communication soulève le problème de la transmission de grands volumes de données soumises à des contraintes temporelles alors que les bandes passantes des réseaux ne sont pas toujours suffisantes. La manipulation de ce type de données devient donc délicate dans ces applications et qui plus est lorsqu'elles sont distribuées. Les travaux que nous adressons dans ce mémoire s'intéressent à la conception et au développement des applications multimédias distribuées à travers l'Internet.

L'utilisation du réseau Internet est, depuis plusieurs années, en pleine expansion. Il propose un panel de possibilités intéressant qui suscite chaque jour des intérêts croissants de la part des différents utilisateurs. Les applications multimédias distribuées à travers ce réseau reflètent une de ces facettes. Les acteurs de l'Internet s'intéressent donc à ce genre d'usage et de développement. Sur l'Internet, elle présente un grand nombre d'avantages comme par exemple l'établissement de communications ou de surveillances à distance. L'utilisation des médias est responsable de ce succès. Néanmoins, leur utilisation sur ce genre de support n'est pas une tâche si facile qu'elle semble l'être et ce, pour plusieurs raisons. Le réseau Internet fournit un service de type « best-effort » c'est-à-dire qu'il procède « au-mieux » pour transporter les données entre les différentes machines qu'il relie. Par exemple, il ne garantit pas les temps nécessaires à la transmission de ces données. Ce service est de fait inadapté pour le transport des données multimédias. Avec l'avènement de l'informatique pervasive, le

réseau Internet permet la connexion d'un ensemble de périphériques différents tant au niveau des capacités matérielles que logicielles. Internet amène donc une forte hétérogénéité dont il est difficile de connaître a priori précisément les caractéristiques. Ces caractéristiques peuvent se révéler importantes dans les applications multimédias distribuées car on ne transmettra pas une vidéo avec la même qualité sur un téléphone mobile ou sur un ordinateur portable. Un autre problème est celui du volume des données transmises à travers l'Internet. Les caractéristiques de bande passante de ce réseau ne sont pas fixes et dépendent des liaisons utilisées. Notre objectif est donc de trouver une solution acceptable répondant à l'utilisation et à la manipulation de ces applications sur l'Internet et ce quel que soit le contexte d'exécution. Nous nous tournons vers une solution d'adaptation des applications à ce contexte. Ces adaptations consistent à reconfigurer dynamiquement la structure des applications afin de les adapter aux changements de contexte. Ce choix est guidé par le fait que la réservation de ressources (autre solution pour l'adaptation) n'est pas possible sur le réseau Internet.

Le point de vue des utilisateurs est un aspect important et trop peu abordé dans les applications multimédias distribuées alors que ces développements sont sensés répondre à leurs besoins. Cette perspective implique un désintérêt rapide des utilisateurs vis-à-vis de ces applications. Des caractéristiques supplémentaires rentrent en jeu et permettent de décrire le contexte humain (handicaps, langues parlées, etc.). Elles se doivent de proposer des services différentiés selon les utilisateurs. L'inconvénient de cet aspect est qu'il est très difficile à évaluer en raison de la diversité des utilisateurs.

L'objectif des travaux présentés est de considérer ces différents aspects en proposant des implémentations adaptées. Les applications multimédias distribuées fournissent une qualité de service intrinsèque et contextuelle. La qualité de service, qui auparavant était appliquée uniquement aux entités matérielles comme le réseau, doit s'étendre afin de couvrir l'ensemble de ces aspects. Un utilisateur d'applications multimédias attend une certaine qualité dans la fourniture d'un service. Cette vision qu'il a du service doit être considérée afin de savoir s'il peut être correctement rendu et avec quel degré de qualité il peut l'être. Les capacités de l'environnement d'exécution vont être des critères déterminants quant à la fourniture de ce service. Notre démarche se propose de gérer les applications en fonction des exigences des utilisateurs et des capacités de l'environnement d'exécution. Ainsi, la qualité de service est une adéquation entre ces deux points de vue.

La solution à cette problématique se trouve dans la définition d'une architecture logicielle chargée de gérer la qualité de service des applications multimédias distribuées. Cette architecture doit fournir les moyens nécessaires à l'évaluation de la quali-

té de service fournie par l'environnement d'exécution et de celle requise par les utilisateurs. De plus, l'Internet constituent un environnement mouvant. Cette architecture doit donc être suffisamment flexible pour intégrer les changements de contexte et autoriser l'adaptation des applications par des reconfigurations dynamiques. Nous proposons donc une architecture logicielle adaptée et composée de deux parties. La première est la définition d'une entité logicielle, appelée plate-forme d'exécution, de supervision des applications multimédias qui permet de gérer leur exécution en fonction des paramètres de qualité de service qu'elle est susceptible de récupérer. Ces préoccupations ont fait l'objet de travaux antérieurs et sont succinctement présentées dans ce mémoire. La seconde partie concerne l'architecture logicielle des applications multimédias. Cette architecture doit pouvoir être supervisée afin de répondre aux exigences de qualité de service. De plus, l'intégration des médias et la prise en compte de leurs caractéristiques doivent être des pré-requis à la définition de cette architecture.

La particularité des médias nous conforte dans la fourniture d'un modèle unique de flux de données dont les objectifs sont de permettre une manipulation et une intégration uniforme et ce quel que soit leur type. L'objectif de ce modèle est double. Dans un premier temps, il doit permettre de répondre aux spécifications fonctionnelles des applications. Dans un second temps, il doit permettre la manipulation de ces médias, en accord avec ces spécifications, par les entités de l'architecture logicielle. Suite à quoi, il est nécessaire de proposer un modèle utilisable pour définir l'architecture. Les entités de ce modèle doivent manipuler les données telles qu'elles auront été définies par le modèle de flux de données. Elles doivent également être supervisables afin que la structure des applications puisse être adaptée dynamiquement par la plate-forme d'exécution chargée de la gestion de la qualité de service. Enfin, ce modèle d'architecture devra faire l'objet d'une implémentation qui va permettre de valider les concepts introduits.

La démarche adoptée pour répondre à cette problématique est une démarche d'analyse descendante qui permet de partir des spécifications fonctionnelles des applications multimédias distribuées pour aboutir à une implémentation utilisable. Cette dernière est réalisée à l'aide d'un modèle de composants spécifique adapté, à la manipulation et au traitement des flux de données, à la reconfiguration tout en étant supervisable. Afin de détailler l'architecture logicielle des applications, nous choisissons le paradigme des composants logiciels qui semble être intéressant dans la définition d'architectures flexibles. Ainsi, les spécifications fonctionnelles et le modèle de données permettent de définir un modèle de composants. Puis, le modèle de composants et les caractéristiques de déploiement permettent d'introduire une implémentation possible de ce dernier.

Le présent mémoire est structuré autour de trois parties. La première partie propose une étude des travaux concernant les applications multimédias distribuées et l'évaluation des travaux qui nous ont inspirés. Le paradigme des composants logiciels et ses multiples possibilités sera également abordé. La seconde partie décrit le contexte global de nos recherches en présentant les points importants de nos précédents travaux qui permettent de comprendre et de justifier une telle démarche. Elle constitue une partie charnière qui va permettre de justifier la nécessité des différents modèles proposés. La troisième partie décrit précisément les modèles de flux de données et de composants logiciels qui vont permettre de concevoir les architectures des applications multimédias distribuées. Enfin, la quatrième partie se propose d'étudier une implémentation possible du modèle de composants à travers l'étude d'un prototype qui va permettre de valider un certain nombre de points clés de l'approche présentée. Nous terminerons par une conclusion ainsi que des perspectives d'évolutions possibles.

Partie 1 – Etat de l'Art

L'objectif de cette première partie est d'introduire la terminologie et les concepts fondamentaux des domaines abordés. Le premier chapitre présente les notions propres aux applications multimédias distribuées. Il s'articule autour des données manipulées par ce type d'applications. Elles sont appelées médias et permettent la communication d'informations aux utilisateurs de ces applications. Certains médias diffèrent des données « traditionnelles » car ils intègrent une dimension temporelle qui contribue de façon essentielle à leur sémantique. La manipulation et l'utilisation de ces données posent un certain nombre de problèmes dans les environnements informatiques actuels qui sont distribués grâce à des réseaux de communication.

Le second chapitre propose un aperçu du paradigme des composants logiciels. Cette approche constitue une solution intéressante pour le développement d'applications informatiques car elle apporte de la flexibilité et élargit les possibilités de réutilisabilité déjà introduites avec la programmation orientée objet. Ce paradigme fournit un éventail très dense de notions, c'est pourquoi nous aborderons uniquement les concepts jugés intéressants et nécessaires pour l'ébauche d'une solution basée composants pour le développement d'applications. Nous présenterons en fin de chapitre un modèle de composants multimédia afin de faire le lien entre les deux domaines de recherche.

Chapitre 1 – Les Applications Multimédias Distribuées

« Consider a future device for individual use, which is a sort of mechanized private file and library. […] A memex is a device in which an individual stores all his books, records, and communications, and which is mechanized so that it may be consulted with exceeding speed and flexibility. […] It consists of a desk, and while it can presumably be operated from a distance, it is primarily the piece of furniture at which he works. […] Books of all sorts, pictures, current periodicals, newspapers, are thus obtained and dropped into place. »

As We May Think, Vannevar Bush, juillet 1945

Les applications multimédias distribuées sont actuellement très répandues que ce soit dans le monde de l'informatique ou dans la vie de tous les jours. Le qualificatif de « multimédia[1] » est très utilisé voire galvaudé, on parle souvent de « révolution multimédia » ou du « tout multimédia ». La puissance actuelle des ordinateurs et des différents types de périphériques disponibles sur le marché (téléphones mobiles, assistants personnels, ordinateurs portables, etc.) couplée à l'émergence des réseaux et aux débits actuels ont largement contribué à démocratiser ce terme. On peut désormais échanger tout type de médias par l'intermédiaire de ces environnements.

L'Internet, réseau de communication mondial, est le support de prédilection des applications multimédias. Son utilisation est en perpétuelle évolution puisque le nombre d'internautes a dépassé le milliard. En juin 2006, on comptait 1 043 104 886 internautes dans le monde, soit 16% de la population du globe, avec en tête les trois continents leaders : Asie (380 400 713 internautes), Europe (294 101 844), Amérique du Nord (227 470 713) [OBS06]. Par son entremise, une partie de la planète peut échanger toutes sortes d'informations au travers d'infrastructures logicielles soumises à un environnement fortement hétérogène. Cependant, en dépit des possibilités offertes, l'utilisation des applications multimédias sur de telles infrastructures reste problémati-

[1] Le terme multimédia a commencé à se répandre avec l'avènement des CD-ROM et CD-I (CD-Interactif) au milieu des années 80. Son utilisation est cependant plus ancienne, puisque l'on retrouve sa trace dans les années 60 et 70 au sein des milieux avant-gardistes de l'art et de la musique. Il désignait alors des spectacles où des films étaient projetés en même temps que des personnes dansaient sur des musiques jouées en direct par un groupe. On parlait alors de performances multimédias, intermédias ou mixed-média (voir à ce sujet le fameux Exploding Plastic Inevitable mis en scène par Andy Warhol). Néanmoins, Vannevar Bush, dans son célèbre article [BUS45], parlait déjà de multimédia sans en employer le terme lorsqu'il décrivait le « memex ».

que, surtout lorsque l'on manipule des médias comme l'audio ou la vidéo sensibles au temps. Il est donc tout à fait naturel de tenter d'améliorer le fonctionnement de ces applications pour essayer de les adapter à l'infrastructure fournie. L'objectif final étant de garantir les services proposés et donc de susciter un intérêt croissant de la part des utilisateurs.

1 Introduction

L'une des premières tâches qui nous incombent est de fixer le cadre de nos recherches en définissant de manière précise les « applications multimédias distribuées ». En effet, nous allons voir que ce terme est large et qu'il peut regrouper, selon les interprétations que l'on en fait, un grand nombre d'applications. Quelques exemples viendront ensuite illustrer cette réflexion.

1.1 Qu'est-ce qu'une Application Multimédia Distribuée ?

Définir précisément ce qu'est une application multimédia distribuée n'est pas une tâche facile car le terme multimédia est sémantiquement très large. En guise d'exemple, nous étudions la définition du dictionnaire [LAR04] :

> *« Qui utilise ou concerne plusieurs médias - adjectif. Ensemble des techniques et des produits qui permettent l'utilisation simultanée et interactive de plusieurs modes de représentation de l'information (textes, sons, images fixes ou animées) – nom masculin. ».*

Si l'on se base sur cette définition, on peut dire qu'une application multimédia manipule de manière simultanée et interactive plusieurs modes de représentation de l'information appelés médias. Néanmoins, cette tentative de définition reste trop générale, il est donc nécessaire de la préciser. De nombreuses applications actuelles manipulent plusieurs médias mais ne sont pas pour autant considérées comme des applications multimédias. La différence doit donc se faire sur un autre point que celui de la simple manipulation des médias par une application. Un aspect qui semble important est le concept de média. D'après la définition précédente, un média est un mode de représentation de l'information qui se rencontre sous diverses formes. Certains d'entre eux fournissent des informations basées sur une diffusion, dépendante du temps, d'un ensemble de données. La prise en compte du facteur temps rend leur utilisation complexe au sein des applications. L'audio et la vidéo sont des exemples de médias très répandus car ils proposent des modes de représentation de l'information intéressants et divertissants dans un contexte applicatif. Il en est pour preuve leur utilisation massive sur l'Internet due aux possibilités offertes par ces médias (réunion de personnes à

distance, téléphonie sur l'Internet, discussions de groupes, etc.). La manipulation de ces médias dans les applications est un point intéressant que nous nous proposons d'étudier en raison de ses particularités. De fait, nous restreignons le champ des applications qui nous intéresse aux applications qui manipulent ces médias. Ainsi, nous définissons les applications multimédias distribuées de la manière suivante :

> *« Une application est qualifiée de multimédia si elle manipule au moins deux médias dont au moins un à dépendance temporelle. De plus, elle est considérée comme distribuée dés lors qu'elle est capable de manipuler ces médias à travers des réseaux de communication afin que plusieurs utilisateurs puissent les échanger, soit pour des situations de communication d'informations distantes, soit pour des situations de partage de l'information. ».*

Ce domaine de recherche n'est pas récent, beaucoup de travaux se sont attardés sur la manière de définir et de classer les applications multimédias. L'approche de Blakowski *et al.* [BLA96] propose une classification de ces applications en considérant la notion de média comme un concept central. Trois critères sont définis dans ce sens. Le premier concerne le nombre de médias manipulés par l'application. Le deuxième introduit le concept de temps et distingue les médias selon deux types, ceux avec dépendance temporelle et ceux sans dépendance temporelle. Enfin, le dernier critère mesure le degré d'intégration des médias dans l'application, c'est à dire la capacité des médias à rester indépendants tout en pouvant être manipulés, traités et restitués ensembles. La Figure 1 présente cette classification. La combinaison des trois critères correspond à une classe d'applications appelée systèmes numériques intégrés définie de la façon suivante :

> *« Un système numérique intégré est une application qui supporte le traitement intégré de plusieurs médias dont au moins un avec dépendance temporelle. ».*

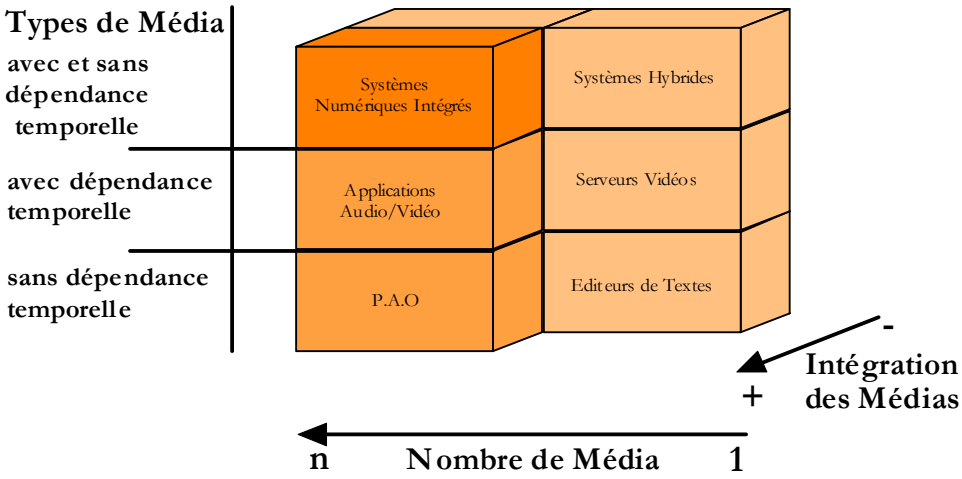


Figure 1 Classification des Applications Multimédias [BLA96][2]

D'après cette classification, notre définition rentre dans la catégorie des systèmes numériques intégrés. La différence réside dans le fait que nous considérons de façon explicite la possibilité de distribution de ces applications très pratiquée actuellement. Dans la suite de ce mémoire, ces applications sont nommées AMD pour Applications Multimédias Distribuées.

En ce qui concerne ces travaux, nous retenons deux points principaux. Le premier aborde la distinction des médias en fonction du facteur temps. Le second définit la capacité d'une application à intégrer les médias de manière indépendante sans pour autant exclure qu'ils puissent être manipulés ensemble. Bien entendu, nous n'excluons pas les autres types d'applications définies dans ces travaux car les modèles que nous définissons permettent également leur conception.

D'autres travaux classent les AMD en fonction d'autres critères. Ainsi Hafid *et al.* [HAF98] proposent un classement orienté fonctionnalité. Ils distinguent à ce titre deux catégories d'AMD : les applications de présentation et les applications conversationnelles. Les applications de présentation fournissent un accès distant à des documents multimédias. Les applications conversationnelles mettent en œuvre des communications multimédias temps réel. Ces applications utilisent des médias à dépendance temporelle afin de mettre en œuvre ces situations de communication temps réel Par conséquent, nous associons notre définition également à ce type d'applications.

---

[2] Le sigle P.A.O signifie Publication Assistée par Ordinateur. Les systèmes hybrides sont des applications qui manipulent les deux types de média mais avec un degré d'intégration faible.

1.2 Quelques Exemples d'Applications

Cette définition ne serait pas complète sans la présentation de quelques AMD représentatives et qui suscitent un réel intérêt à l'heure actuelle. Bien que toutes soient des AMD au sens de la classification précédente [BLA96], elles présentent cependant des caractéristiques différentes en ce qui concerne le type de qualité de service qu'elles se doivent d'offrir.

1.2.1 La Télésurveillance

Les applications de télésurveillance permettent la mise en place de surveillance à distance par l'intermédiaire d'un réseau de communication, l'objectif étant de détecter des événements ou phénomènes physiques à l'aide de capteurs[3]. Ainsi, une telle application doit pouvoir réagir en informant les différents utilisateurs (stations de surveillance) par transmission d'alarmes ou d'informations audiovisuelles et ce dans des délais acceptables. Ces AMD sont souvent équipées de caméras, on parle alors de vidéosurveillance. Elles sont très utilisées dans les domaines de la domotique et de la télémédecine.

La société Magys en collaboration avec le LIUPPA (Laboratoire d'Informatique de l'Université de Pau et des Pays de l'Adour) et l'ESTIA (Ecole Supérieure des Technologies Industrielles Avancées) développe un système de surveillance autoroutière dont le but est de filmer des portions critiques d'autoroute où les risques d'embouteillage sont élevés [TOT05], [TOT07]. Une caméra est utilisée pour réaliser cette tâche. Les images acquises sont transmises à une station distante équipée d'un modem GSM[4] (Global System for Mobile communications). Avant d'être émises sur ce réseau, les images sont compressées afin d'en réduire le volume. Lors de leur réception, elles sont décompressées puis restituées aux utilisateurs.

Ces AMD se caractérisent par le fait qu'elles doivent offrir des informations fiables en temps réel sans toutefois nécessiter des images d'excellente qualité. En revanche, elles sont soumises à l'utilisation de supports de transmission de faible débit et à des conditions environnementales non maîtrisables (éclairage, etc.).

---

3 Voir à ce sujet, certains travaux sur les réseaux de capteurs comme par exemple [LIN03].

4 Le GSM est un réseau utilisé pour la téléphonie mobile. Ce réseau est commuté, les ressources sont allouées pour la durée de la communication.

1.2.2 La Vidéoconférence

Les applications de vidéoconférence permettent la réunion de plusieurs personnes physiquement situées en des lieux différents par transmission en temps réel de la parole et de l'image de chaque site. Ces médias peuvent être accompagnés d'autres supports comme par exemple des transparents dans le cadre d'une conférence à distance. Les trafics engendrés restent un défi lorsque le débit des réseaux est limité ou que leur fiabilité diminue (surcharge). Des techniques empruntées au traitement du signal, comme par exemple la compression, peuvent être utilisées pour réduire le volume des données à transmettre. De plus, l'emploi de médias comme l'audio et la vidéo nécessite la mise en œuvre de mécanismes particuliers afin d'assurer leur restitution de façon synchrone. Ces systèmes couvrent un nombre important de champs d'application : réunions de travail, cours à distance, conférences à distance, etc.

Le MRCL (Multimedia Communications Research Laboratory) de l'université d'Ottawa a développé un système de vidéoconférence portable à travers le réseau Internet [ZHU01]. Il permet la diffusion de données audio et vidéo sur chaque site participant à la vidéoconférence. Les transmissions sont adaptées afin de pouvoir satisfaire le plus grand nombre d'utilisateurs. Par contre, cette AMD ne prend pas en compte la synchronisation entre l'audio et la vidéo qui est pourtant très importante.

Ces AMD sont caractérisées par des débits multilatéraux importants. Elles doivent garantir la synchronisation des médias afin d'offrir un service agréable à utiliser. Par opposition aux précédentes, elles bénéficient de conditions environnementales plus facilement maîtrisables.

1.2.3 La Vidéo à la Demande

Les applications de vidéo à la demande[5] permettent la diffusion à différents utilisateurs de médias stockés sur une ou plusieurs machines distantes et dédiées (appelées quelquefois serveurs de médias). Chaque utilisateur choisit les médias qu'il désire recevoir, le serveur transmet alors les médias sélectionnés à l'utilisateur. Cette diffusion peut être contrôlée à distance de la même façon que sur un magnétoscope ou un lecteur de DVD[6] (lecture, pause, arrêt, avance rapide, retour rapide). Le plus souvent, ce sont des techniques de streaming qui sont employées pour donner la possibilité aux utilisateurs de commencer la diffusion du média en même temps que son télécharge-

---

5 En anglais VOD pour Video On Demand.

6 DVD : Digital Versatile Disc.

ment. Ces AMD sont proches de la télévision, elles nécessitent donc des qualités d'images et de son élevées. Elles sont de plus en plus répandues du fait qu'elles suscitent un large intérêt économique. Les grands groupes de télédiffusion fournissent des systèmes de vidéo à la demande en échange d'une participation pécuniaire de la part des utilisateurs.

Début 2006, la chaîne de télévision franco-allemande ARTE a proposé un service de vidéo à la demande [ART06] qui offre la possibilité aux utilisateurs de choisir leur programme parmi un catalogue de films, de documentaires et d'émissions. Chaque programme proposé est payant et peut être visionné par le téléspectateur plusieurs fois s'il le désire. Sa diffusion est contrôlée par l'intermédiaire d'un lecteur multimédia intégré à ce service.

1.2.4 Synthèse

Ces quelques exemples permettent d'illustrer les multiples contraintes auxquelles sont soumises les AMD. En effet, elles cumulent le double inconvénient de devoir manipuler et transmettre de grandes quantités d'informations en respectant des contraintes de temps et de synchronisation extrêmement rigoureuses. De la même façon que les applications industrielles peuvent être rendues inutilisables par le non respect du temps réel, les AMD deviennent rapidement inexploitables lorsqu'elles ne parviennent pas à offrir de manière constante un service de qualité satisfaisante aux utilisateurs.

Au travers de cette rapide introduction, nous pouvons nous rendre compte de l'importance prise par les médias qu'elles manipulent, c'est pourquoi nous allons examiner plus précisément, dans le paragraphe suivant, les médias et leurs principales caractéristiques.

2 Les Médias dans les AMD

Les AMD créent, manipulent, restituent, stockent et transmettent des médias de diverses natures comme du texte, des graphiques, des images fixes, de l'audio, de la vidéo ou encore des informations de contrôle. Ces médias présentent des propriétés différentes qu'il est nécessaire d'appréhender pour les manipuler. On distingue généralement deux catégories : les médias continus et les médias discrets.

Au delà de simplement distinguer les médias par la nature des données supportées, on peut également les classer en fonction du facteur temps qu'ils intègrent. Ce sont les médias identifiés comme étant avec dépendance temporelle dans la classifica-

tion de Blakowski *et al.* [BLA96] (cf. 1.1). Le respect de ces dépendances est capital si l'on veut que les données supportées aient un sens. Des relations de synchronisation traduisent ces dépendances.

Certains médias sont volumineux et il est clair que leur échange à travers un réseau de communication comme l'Internet reste problématique. Des recherches ont donc été menées afin de proposer des techniques de compression basées sur des propriétés psychosensorielles comme par exemple le masquage de certaines fréquences considérées comme inaudibles en audio ou la prise en compte du mouvement en vidéo qui se base sur le fait que des images successives contiennent des informations redondantes.

Enfin, ce paragraphe se termine par une rapide présentation du transport des médias dans les AMD. Deux possibilités sont étudiées, à savoir le transport en local et le transport distribué à travers un réseau comme l'Internet. Nous identifions à travers cette présentation les problèmes posés par la communication des médias dans une AMD.

2.1 Médias Continus vs. Médias Discrets

Les médias sont différenciés selon la nature des informations qu'ils supportent et selon leur constitution. Habituellement, on distingue les médias continus et les médias discrets dont on va décrire les principales caractéristiques. Cette connaissance est importante pour la manipulation des médias dans une AMD. Certains travaux utilisent cette classification pour la manipulation des médias [GIB94], [GIS94].

2.1.1 Les Médias Continus

Les médias continus se composent d'une séquence d'échantillons continue et a priori infinie. Chaque échantillon décrit une partie de l'information véhiculée par le média dans un format de codage adéquat. Ils sont produits par échantillonnage périodique de données captées par des périphériques d'acquisition tels que les caméras ou les microphones. Ces médias affichent des caractéristiques de périodicité et de continuité qui impliquent qu'ils existent sous la forme de flux de données appelés également flux continus[7].

Ces flux affichent une dépendance temporelle intrinsèque qui traduit une importante partie de la sémantique de ces médias. En effet, l'information véhiculée ne peut

[7] Stream ou Flow en anglais.

se comprendre que par une diffusion rythmée des échantillons. Cette dépendance temporelle doit être respectée lors de la restitution des données à travers des périphériques adéquats comme des haut-parleurs ou un écran. Ces contraintes ne relèvent pas de contraintes de type temps-réel dur, ce qui signifie que des marges sont exploitables lors de la diffusion de ces médias. De plus, ce type de média supporte des pertes occasionnelles de données qui passent inaperçues lors de la restitution ; malgré tout, ce type de média reste quand même très sensible au temps.

Pour définir les médias continus, nous considérons que la séquence qu'ils représentent est a priori infinie. Bien entendu, cette séquence peut être finie dans le temps. Par exemple, un fichier .AVI supporte un média continu qui comporte un nombre d'échantillons finis. Par opposition, un média continu a priori infini peut concerner un média issu d'un périphérique de capture.

La vidéo est un exemple de média continu. La norme SECAM (SEquentiel Couleur A Mémoire) est une norme de codage vidéo en couleur utilisée pour la télévision en France qui impose une fréquence de diffusion de 25 images par seconde. Cette caractéristique implique donc de restituer au téléspectateur une image différente toutes les 40 millisecondes afin de donner l'illusion parfaite du mouvement. La Figure 2 montre la façon dont est structuré un flux vidéo qui respecte cette norme. Il est formé d'une suite d'images caractérisée par deux types de délai. Le premier est le temps qui sépare la production de deux images successives par le périphérique ou le processus d'acquisition. Ce délai est appelé gigue multimédia. Ce temps peut varier mais doit être inférieur au second délai, qui spécifie la contrainte temporelle de restitution, si l'on veut respecter le débit du flux.

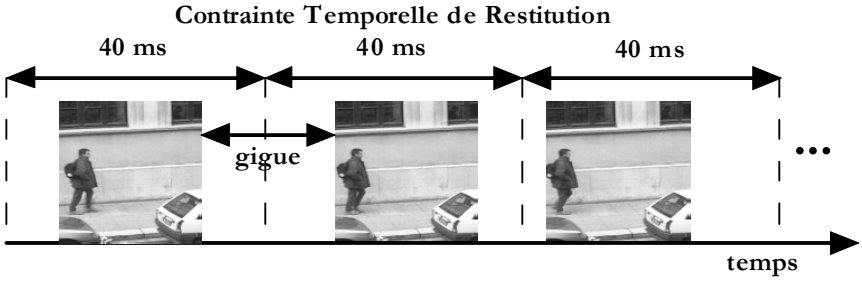


Figure 2 Caractéristiques Temporelles d'un Flux Vidéo[8]

8 Cette séquence d'image est empruntée au laboratoire TIRF de l'INP/Grenoble.

2.1.2 Les Médias Discrets

Un média discret est constitué d'un ensemble de données indivisibles. La totalité des données, représentées dans un format de codage adéquat, est nécessaire pour restituer l'information véhiculée par un média discret. Ces données sont ponctuelles, c'est à dire qu'elles existent à un moment précis et non de façon continue dans le temps comme pour les médias continus.

Le facteur temps n'est donc pas aussi prépondérant que pour les médias continus. Ces médias ne présentent pas de dépendance temporelle forte. Lors de la restitution des données, le seul point important à respecter est de disposer de l'intégralité des données. Ce type de média est donc sensible aux pertes de données.

Le mode de restitution des médias discrets se fait à travers des périphériques adéquats comme par exemple un écran. Ce processus se fait par interprétation ou décodage des données qui constituent le média.

Les médias discrets sont finis dans le temps puisqu'ils représentent un ensemble de données fini et indivisible. Comme exemple de médias discrets, on peut citer les images fixes, les graphiques, le texte, etc. La Figure 3 montre l'exemple d'une page web qui utilise différents médias discrets comme le texte et des images.

Figure 3 Une Page Web qui utilise des Médias Discrets[9]

2.1.3 Quelques remarques sur cette classification

La classification proposée entre médias discrets et médias continus est souvent utilisée car elle reflète bien la diversité des types d'informations des AMD ainsi que celle des caractéristiques qui doivent être prises en compte. Il n'en reste pas moins vrai que cette classification n'est pas parfaite. En effet, si l'on prend l'exemple d'un média constitué par les sous-titres d'une vidéo, il présente à la fois des caractéristiques propres aux médias continus (respects de délais précis) et d'autres propres aux médias discrets (sensibilité à la perte de données). De la même façon, un diaporama supportera mal la perte d'informations, il constitue un flux d'images synchronisées qui acceptent cependant une gigue plus élevée que celle d'un flux vidéo.

La frontière entre médias continus et médias discrets définie par le lien temporel entre les échantillons n'est guère précise et ne peut être définie de façon absolue. Un diaporama présentant une image toutes les 10 secondes est communément considéré comme un média discret, mais qu'en est-il si les images défilent au rythme de la seconde ?

---

9 Cette page web est empruntée à un projet d'étudiants de la licence professionnelle SIL de l'IUT de Bayonne.

Ces quelques remarques conduisent à la nécessité de définir des critères plus appropriés à nos travaux que ceux présentés ici. Il paraît également nécessaire de disposer d'une représentation unifiée de tous les types de médias de façon, en particulier, à pouvoir assurer la synchronisation entre médias a priori différents comme, par exemple, entre vidéo, son et sous-titres.

2.2 La Synchronisation des Médias

Traditionnellement, le terme de synchronisation implique la notion de temps. Par exemple, dire que deux événements sont synchrones signifie que ces événements se produisent en même temps. Ce concept est fondamental dans les AMD dés lors que l'on manipule des médias avec dépendance temporelle car la synchronisation est une de leurs propriétés.

Le terme de synchronisation est large et peut avoir plusieurs significations. En effet en ce qui concerne les médias, on peut définir trois types de synchronisation représentés sur la Figure 4. Le premier est appelé la synchronisation de contenu et met en avant des relations liées aux contenus de plusieurs médias différents. Par exemple, on peut définir une relation de synchronisation entre un tableau et un graphique représentant les données du tableau sous une autre forme. De manière générale, ce type de relations est établi explicitement afin de permettre une mise à jour automatique des différentes vues d'un ensemble de données (cf. Logiciel Microsoft EXCEL). La synchronisation spatiale permet de définir des zones de placement pour la restitution de différents médias avec éventuellement des références temporelles pour la définition de présentations multimédias. On peut, par exemple, définir de telles zones sur les interfaces graphiques d'une AMD pour spécifier le placement du texte par rapport à une image ou une vidéo. La page web de la Figure 3 est un exemple de ce type de synchronisation. Enfin, le dernier type est nommé synchronisation temporelle et s'intéresse aux relations temporelles susceptibles d'exister à l'intérieur d'un même média mais aussi entre différents médias.

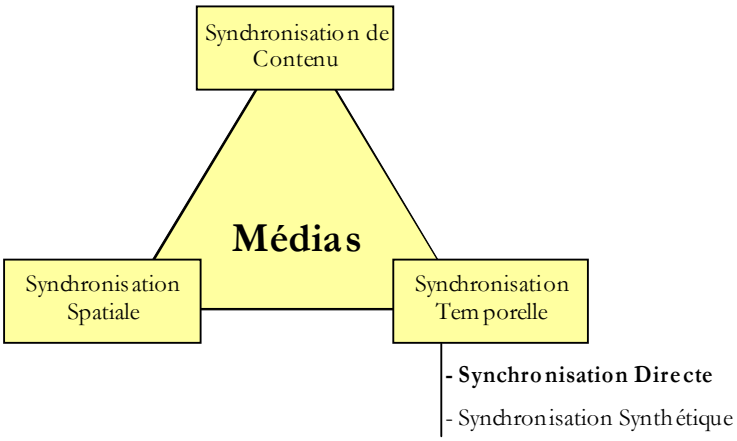


Figure 4 Différents Types de Synchronisation

La synchronisation de contenu relève de la pure sémantique des données et il appartient au concepteur de l'AMD de prendre en charge la conservation de ce lien. La synchronisation spatiale, quant à elle, est liée à la présentation de l'information et relève de ce fait de la conception des IHM[10]. La synchronisation temporelle en revanche est sensiblement plus diffuse puisqu'elle apparaît tant lors de l'acquisition ou création des informations que de leur manipulation, de leur transport et de leur restitution. La suite de ce paragraphe se focalise sur cette dernière en raison de son importance dans la manipulation des médias au sein des AMD et de l'impérieuse nécessité de la respecter à tous les niveaux de l'application.

Dans la synchronisation temporelle, on distingue généralement (cf. Figure 4) la synchronisation directe et la synchronisation synthétique. L'objectif de la synchronisation directe est de conserver les relations temporelles telles qu'elles existaient lors de l'acquisition ou de la création des médias. Par exemple, elle peut consister dans une AMD de vidéoconférence à conserver les relations temporelles entre le son et les images du locuteur. La synchronisation synthétique consiste à créer artificiellement des relations temporelles (par exemple celles définies dans [ALL83]) entre différents médias à respecter lors de leur restitution [LIT90]. Ce type de synchronisation est très utilisé dans la réalité virtuelle où il est nécessaire de faire correspondre l'évolution de plusieurs événements entre eux. Par exemple, lorsqu'un personnage ouvre une porte dans un monde virtuel alors il faut que cette ouverture coïncide avec le bruit d'ouverture et le mouvement de la main du personnage.

Le travail présenté dans ce mémoire se focalise principalement sur la synchronisation directe car notre objectif n'est pas de créer des relations de synchronisation

[10] Interfaces Homme-Machine.

mais de les conserver de sorte qu'il s'agisse de ne pas perdre ces relations qui ont existé au moment de la création des médias. Le terme de synchronisation multimédia sera désormais utilisé afin de faire référence à la synchronisation directe.

La synchronisation multimédia traduit des propriétés psychosensorielles relatives aux médias. Les relations temporelles, exprimées par ce type de synchronisation, doivent être prises en compte lors de la restitution de ces derniers. Elles existent à deux niveaux :

- entre les données issues d'un même média, on parle alors de synchronisation intra-média ;
- entre les données issues de différents médias, on parle alors de synchronisation inter-médias.

Des études menées sur la perception humaine de la synchronisation multimédia viendront en fin de partie appuyer l'importance de la synchronisation directe dans les AMD.

2.2.1 La Synchronisation Intra-Média

La synchronisation intra-média est une caractéristique propre aux médias continus. Elle se définit comme une relation temporelle intrinsèque à un média, c'est à dire aux données qui le composent. Cette synchronisation permet de restituer un aspect naturel (impression de fluidité) lors de la diffusion du média. En effet, ces données ne sont correctement perçues par un être humain que par leur diffusion à un rythme approprié. Ce rythme est connu a priori car il est défini par la fréquence d'échantillonnage utilisée par le format de codage du média.

Ce type de synchronisation peut également être pris en considération pour les médias discrets mais elle est beaucoup moins importante que dans les médias continus car elle ne constitue pas une propriété à part entière. Ainsi, elle peut permettre de traduire la distance temporelle entre deux données discrètes consécutives et de même type. Dans ce cas, cette relation n'est connue qu'a posteriori.

2.2.2 La Synchronisation Inter-Médias

La synchronisation inter-médias définit les relations temporelles entre des données appartenant à différents médias. Généralement, ce type de synchronisation existe entre plusieurs médias continus mais peut également se rencontrer entre des médias continus et des médias discrets. On peut, par exemple, sous-titrer une vidéo lorsque la langue utilisée n'est pas comprise par l'auditeur : on synchronise alors l'audio, la vidéo

et le texte utilisé pour le sous-titrage. La synchronisation inter-médias, par opposition à la précédente, est implicite. Ces relations existent lors de l'acquisition ou de la création des médias et elles doivent être conservées jusqu'à leur restitution.

2.2.3 Perception Humaine et Importance de la Synchronisation Multimédia

Le respect des relations de synchronisation est nécessaire dans les AMD si l'on veut donner une impression naturelle lors de la restitution des médias. La diffusion de médias désynchronisés paraît artificielle et peut se révéler quelquefois incompréhensible (par exemple lorsque le son et l'image sont décalés dans une vidéo). La sémantique des médias est donc directement atteinte.

Plusieurs études ont été réalisées sur la perception humaine de la synchronisation multimédia. Elles permettent de mettre en évidence son importance. Par exemple, les études présentées dans [GHI98] et [WEI98] montrent que la restitution du mouvement dans une vidéo nécessite une fréquence minimale comprise entre 12 et 15 images par seconde afin d'éviter tout effet stroboscopique lors de la restitution des mouvements rapides. Au dessous de ce seuil, des saccades commencent à apparaître sur la vidéo ce qui a pour effet de détourner l'attention de l'utilisateur.

D'autres expériences, comme celles menées à l'IBM European Networking Center [STE96], montrent l'effet des décalages temporels[11] entre différents médias. L'objectif de cette expérience est d'introduire des décalages temporels artificiels (par tranche de 40 millisecondes) entre le son et l'image d'un journal télévisé diffusé à un panel représentatif de téléspectateurs. Les résultats montrent en premier lieu que les décalages sont perçus chaque fois que le présentateur du journal commence et arrête de parler. Au delà de 80 millisecondes de décalage entre le son et l'image, ce dernier devient perceptible mais le journal télévisé reste néanmoins compréhensible. Dés que l'on dépasse 160 millisecondes, le décalage est tel qu'il en arrive à perturber les téléspectateurs et dégrader la compréhension du média à tel point que l'information fournie devient incompréhensible.

2.2.4 Synthèse

On mesure, à la vue de ces résultats, l'importance de la synchronisation multimédia dans les AMD. Le fait de ne pas la respecter amène à des situations critiques dans lesquelles les médias (donc l'information supportée) commencent à ne plus être exploitables. Dans ces cas, l'utilisation des AMD commence à devenir rédhibitoire. En

---

11 Le terme « skew » est utilisé dans l'article qui présente ces travaux publié en langue anglaise.

conséquence, nous nous fixons comme premier objectif d'assurer et de maintenir la synchronisation multimédia lors du transfert des médias dans les AMD. A ce titre, nous allons identifier les fonctionnalités des AMD qui sont susceptibles de modifier ces relations de synchronisation.

2.3 Codage, Compression et Structure des Médias Continus

Cette section aborde les principes de codage et de compression des médias continus les plus répandus dans les AMD. Cette étude nous permet de mettre en évidence la structure des médias continus. La structure des données s'avère importante lorsque l'on veut développer des AMD avec un fort degré d'intégration des médias.

Les médias continus représentent un débit d'information trop important pour pouvoir les propager sous leur forme première. Des travaux sur la compression de ces données ont naturellement fait leur apparition avec pour objectif de réduire ce volume conséquent sans pour autant perdre trop de qualité (compromis entre compression et perte de qualité). L'exemple présenté est celui de la norme MPEG (Motion Picture Experts Group) car c'est une solution largement répandue pour la transmission des médias continus de façon synchrone.

2.3.1 Principe de Codage

Nous abordons dans cette section les principes de codage des médias continus les plus utilisés.

2.3.1.1 L'Audio

Le son se concrétise par un changement répétitif de la pression d'un milieu donné, par exemple l'air ou l'eau. Ces changements de pression sont appelés ondes sonores. Un microphone permet de capter et transformer ces ondes en informations audios analogiques c'est à dire en signaux électriques. Le microphone est un périphérique d'acquisition qui produit une amplitude sonore dépendante du temps. Ces signaux sont ensuite numérisés à l'aide d'un CAN (Convertisseur Analogique Numérique) qui, à partir d'une tension électrique, génère un mot binaire[12] représentant l'amplitude instantanée du signal. Ce processus est appelé échantillonnage. Le théorème de Shannon-Nyquist énonce qu'un signal échantillonné ne peut complètement représenter un signal dont la fréquence est plus grande que la moitié de la fréquence

[12] La taille de ce mot est appelée résolution.

d'échantillonnage. En conséquence, il faut choisir une fréquence d'échantillonnage f_e respectant la propriété suivante : $f_e > 2.f_{max}$ où f_{max} représente la plus haute fréquence contenue dans le signal à échantillonner.

L'audio sous forme numérique est donc formé d'une suite de mots binaires. Par exemple, les supports de type CD (Compact Disc) supportent un signal échantillonné à une fréquence de 44100 Hertz[13] avec une résolution de 16 bits. Ce signal numérique sera donc composé de 44100 échantillons par seconde de 4 octets chacun en stéréo.

Un CNA (Convertisseur Numérique Analogique) permet, à partir des données numériques, de recréer un signal électrique qui sera ensuite envoyé vers la membrane d'un haut-parleur afin de restituer le son correspondant. Un haut-parleur est un périphérique de restitution.

2.3.1.2 La Vidéo

L'œil humain a la faculté de former des images précises sur les cellules photosensibles de la rétine pendant quelques millisecondes. Si une séquence d'images défile à 25 images par seconde ou plus, l'œil ne perçoit pas qu'il s'agit d'images distinctes. Les systèmes vidéo et cinématographiques utilisent ce principe pour restituer le mouvement.

Une caméra permet de convertir une image optique en image électronique. Cette dernière est balayée pour obtenir un signal électrique appelé signal vidéo. L'objectif est de capturer un nombre suffisant d'images par seconde afin de donner l'illusion du mouvement. Les images sont représentées par une grille rectangulaire d'éléments d'image, plus communément appelés pixels (picture elements). Chaque pixel est codé par un mot binaire représentant l'information de couleur appelée chrominance et l'information de luminance. Chacune de ces informations est une combinaison de couleurs suivant l'espace colorimétrique utilisé. Un signal vidéo numérique se compose d'une suite d'images. Ces images sont ensuite restituées au rythme approprié sur un écran.

2.3.1.3 Structure des Médias Continus

En dépit du fait que leurs processus d'acquisition et de création soient différents, les médias continus sont caractérisés par une structure commune basée sur le

---

[13] D'après le théorème de Shannon-Nyquist, un tel échantillonnage permet de capturer des sons ayant une fréquence maximale de 22050 Hertz. Cette fréquence se situe largement au-dessus du maximum perceptible par l'oreille humaine.

temps. Ils sont constitués d'une suite continue d'échantillons. Le nombre d'échantillons par seconde est lié à la fréquence d'échantillonnage (cf. § 2.3.1.1).

Un média continu est donc constitué d'une séquence continue et régulière d'unités d'information qui sont appelées unités de données logiques (LDU[14]) dans [BLA96]. Une telle unité peut être constituée d'un ou de plusieurs échantillons. Ceci dépend des caractéristiques intrinsèques du média considéré mais également des spécifications de l'AMD à développer en terme de types de traitements et de manipulations. En effet les données des médias continus peuvent être organisées en plusieurs niveaux hiérarchiques de granularité différente. Par exemple, nous pouvons représenter l'organisation hiérarchique d'une vidéo comme sur la Figure 5 (sous la forme d'un diagramme UML[15]). En effet, on peut dire qu'une vidéo se découpe en scènes qui elles-mêmes sont constituées d'images construites à partir de blocs de pixels et enfin que chaque bloc est, lui-même, composé de pixels. Ainsi, selon les manipulations que l'on désire pratiquer sur ces médias dans les AMD, les unités pourront être composées par l'un des différents niveaux hiérarchiques. On peut tout à fait imaginer de disposer d'un média vidéo où chaque unité d'information contiendrait une scène entière. Toutefois, cette structuration a un impact direct sur les relations de synchronisation intra-média. En effet, plus on descend dans ce type de hiérarchie, c'est à dire plus on choisit un grain fin pour constituer les unités, plus les relations temporelles entre ces dernières seront difficiles, voire impossible, à respecter de manière logicielle[16]. Ce type de structuration correspond à la notion de flux de données. Certains médias existent sous la forme de flux constitués d'une séquence d'unités d'information avec des propriétés de continuité et de régularité différentes.

---

14 Logical Data Unit dans l'article écrit en langue anglaise.

15 Unified Modeling Language. Nous essayerons d'utiliser ce langage tout au long de ce mémoire puisqu'il est très utilisé et également reconnu comme un standard [BOO01].

16 Il est clair qu'il est impossible de respecter les contraintes temporelles d'une vidéo qui serait alors constituée d'une suite de pixels de manière logicielle. Cet exemple induirait des contraintes temporelles intra-flux entre les pixels de la vidéo de l'ordre de la microseconde. Dans ce sens, certains travaux se sont intéressés à des solutions matérielles qui travaillent sur des unités plus petites que l'image afin d'obtenir des performances temps réel pour des traitements très gourmands en calculs. Voir à ce titre, les travaux présentés dans [DUM99] (Thématique AAA du GDR ISIS Adéquation/Algorithmique/Architecture).

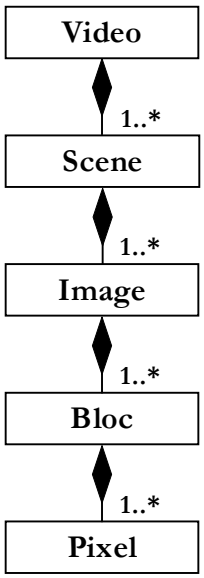


Figure 5 Décomposition Hiérarchique d'une Vidéo

2.3.2 Principe de Compression

La taille conséquente de certains médias a engendré des recherches sur la compression de ce type de données dont le principal objectif est de réduire de manière significative le volume d'information nécessaire à leur représentation. Les techniques employées se basent sur des caractéristiques psychosensorielles humaines. La compression peut se faire avec ou sans perte de qualité. Toutefois, il s'avère que les compressions les plus efficaces s'opèrent en dégradant la qualité des données.

A titre d'exemple, nous présentons la norme MPEG [RAN96] qui permet de compresser des signaux audio et vidéo analogiques ou numériques afin de faciliter leur manipulation sur un réseau. L'un des avantages de cette norme est que la synchronisation inter-médias est assurée.

La compression MPEG tire profit des analogies qui existent entre les images successives d'une séquence vidéo par la mise en œuvre de méthodes de prédiction et d'interpolation dans le but de réduire les redondances d'information contenues dans ces séquences, cette méthode est connue sous le nom de compression temporelle. MPEG utilise également la compression spatiale qui consiste à compresser chaque image d'une séquence vidéo sans tenir compte des autres. Le principe de MPEG est basé sur le multiplexage des flux. Les données audio et vidéo sont encodées séparément, empaquetées avec des références à l'horloge système puis fusionnées pour obtenir un flux multiplexé.

Ce type de compression constitue une solution intéressante pour la manipulation des médias dans les AMD car elle permet d'obtenir des débits intéressants. De

plus, grâce au multiplexage, la norme MPEG assure de conserver les liens de synchronisation entre les flux. Beaucoup de travaux sur les AMD ont adopté cette solution [BAI96], [BLA02], [CHA00], [CHE03], [HEM99], [MUN99]. Beaucoup d'applications de vidéo à la demande proposent des flux au format MPEG.

Malgré ces constatations, cette solution est discutable, et ce sur plusieurs points. Nous avons vu que l'intégration des médias est un critère important dans la conception des AMD que nous visons. Ce critère traduit la capacité d'une AMD à intégrer des médias indépendants tout en permettant de les traiter et de les manipuler seuls ou en groupe. Le fait d'utiliser une solution basée sur le multiplexage des données est un frein pour l'intégration des données. En effet, ce genre de technique fait perdre l'opportunité de pouvoir traiter séparément les différents flux sauf à mettre en œuvre des architectures complexes qui intègrent, pour chaque traitement, un processus de décompression du flux multiplexé, un processus permettant de conserver les relations temporelles entre les flux démultiplexés puis un processus de compression utilisable après chaque traitement. Malheureusement, les processus de compression et décompression introduisent des temps de traitement non négligeables et il n'est pas souhaitable de les multiplier dans l'architecture d'une AMD. De plus, le multiplexage des flux impose une qualité de service combinée. On ne peut donc plus gérer indépendamment la qualité de chaque média en la dégradant ou en l'augmentant, voire même en supprimant l'un des médias selon les besoins en qualité de service d'une AMD. Enfin, un autre inconvénient réside dans la difficulté d'intégration des données de diverses natures dans la norme. En effet, lorsque l'on conçoit une architecture logicielle, il est difficile de décrire de manière exhaustive tous les types de données qui sont susceptibles d'exister dans une AMD. Il semble donc difficile de les intégrer aisément dans la norme. Ce type de solution réduit les degrés de liberté quant à l'intégration des médias au sein des AMD, ce qui a pour conséquence une importante perte de flexibilité. Lorsque l'objectif d'une modélisation est basé sur la gestion de la qualité de service des AMD, nous pensons que cette perte de flexibilité est dommageable dés lors que l'on considère que la gestion des flux de manière indépendante peut participer à l'amélioration ou à la dégradation du service fourni par une AMD. Nous nous rallions en cela aux arguments exposés dans [TEN90] qui s'opposent aux solutions basées sur le multiplexage.

2.4 Le Transport des Médias

Afin que les médias puissent être utilisés dans les AMD, il est nécessaire de les diffuser à l'intérieur de celles-ci. Nous nous intéressons donc au transport des médias dans les AMD afin d'identifier les problèmes posés par ce transport et les sources de

désynchronisation des médias. A l'intérieur d'une AMD, le transport est réalisé à deux niveaux :

- le transport local des médias c'est à dire entre les différents modules d'une AMD ;
- le transport distribué des médias c'est à dire entre les différents sites d'une AMD à l'aide d'un réseau de communication.

2.4.1 Le Transport Local

Le transport local consiste à distribuer les médias entre les différents modules (unités d'implémentation) d'une AMD se trouvant sur la même machine. Nous illustrons ce point en nous appuyant sur un exemple d'AMD (un exemple comparable est donné dans [BOU05]).

Nous considérons une AMD qui a pour objectif la diffusion d'un bulletin météo en direct sur l'Internet dans le cadre d'une WebTV[17]. L'architecture de cette AMD est donnée sur la Figure 6. Cette application est structurée autour de plusieurs modules chargés de manipuler et de traiter les médias qui transitent dans l'AMD. On distingue sur la gauche de la Figure 6 les quatre modules chargés de l'acquisition et de la création des médias. Les acquisitions audio et vidéo permettent de filmer et d'enregistrer le présentateur du bulletin météo. Le module d'images permet d'envoyer les cartes météorologiques et les informations de l'éphéméride qui serviront d'arrière plan. Enfin, le dernier module permet de proposer une musique de fond. Cette AMD utilise également deux modules de traitement des médias que nous avons représentés en bleu sur la Figure 6. Ce bulletin est réalisé en utilisant la technique du bluescreen qui consiste à filmer le présentateur sur un fond monochrome uniforme (par exemple bleu ou vert). Un module d'incrustation (technique de compositing) permet, à partir de la vidéo ainsi filmée et du flux d'images, d'obtenir une nouvelle vidéo sur laquelle on peut voir le présentateur avec, comme arrière plan, les cartes météorologiques. Un module de mixage permet de mélanger les deux flux audio afin d'obtenir en sortie un flux composé de la musique atténuée et de la voix du présentateur.

---

[17] Une WebTV est une chaîne de télévision qui diffuse ses programmes à travers l'Internet.

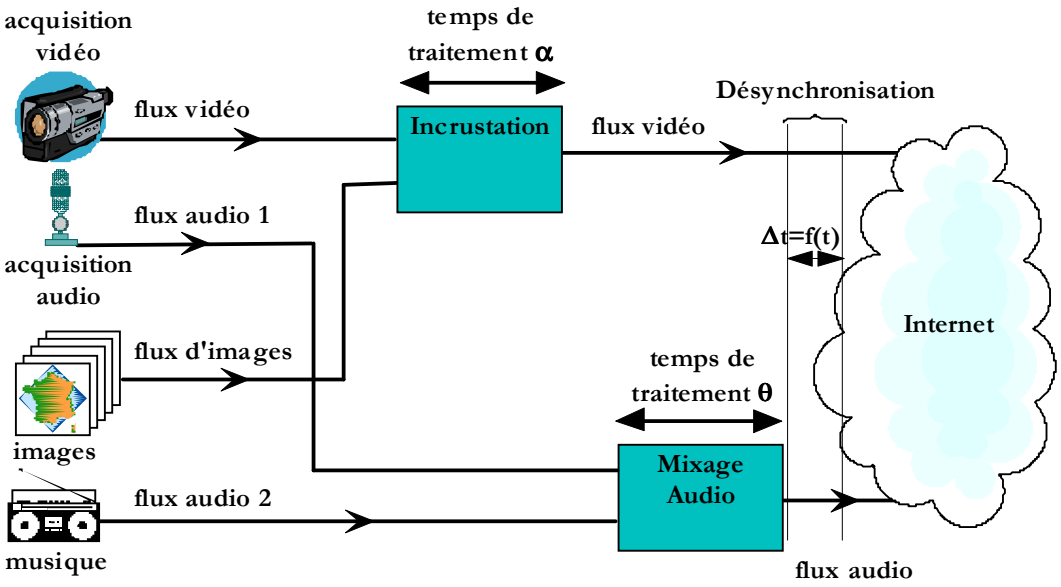


Figure 6 Diffusion d'un Bulletin Météo

Le flux vidéo et le flux audio 1 sont synchrones entre eux lors de leur acquisition respective puisqu'ils représentent le son et l'image du présentateur. Le flux vidéo et le flux d'images sont tous les deux récupérés par le module d'incrustation qui procède à ce traitement au fur et à mesure de l'arrivée des données. Le problème qui se pose est que ce traitement induit un temps α qui agit sur la synchronisation inter-flux initiale entre le flux vidéo et le flux audio 1. De plus, le flux audio 1 est récupéré par le module de mixage audio afin de le mixer avec le flux audio 2. Ce traitement induit également un temps θ. On constate alors que les deux flux qui seront transmis vers les téléspectateurs ne sont dés lors plus synchrones. On se trouve ici face à un problème de désynchronisation inter-flux causé par les temps de traitements. Cette désynchronisation se traduit par un décalage Δt entre le flux audio et le flux vidéo (cf. Figure 6). En effet, ces temps agissent sur les relations temporelles des flux de données synchrones car chacun d'eux induit des retards différents. On imagine volontiers que, dans le cas d'une AMD plus complexe, les retards accumulés affecteront de manière importante les relations de synchronisation. Ce genre de situation n'est évidemment pas acceptable si l'on veut que les AMD fournissent des informations correctes.

Cet exemple nous permet d'identifier un premier problème faisant obstacle aux relations de synchronisation inter-flux : le fait que l'on doive séparer des flux synchrones pour les envoyer sur des modules de traitement différents. Une solution à ce problème consisterait à multiplexer les flux et à rendre obligatoire l'utilisation d'un module de traitement unique capable de travailler sur un flux multiplexé. Habituellement, ces modules doivent inclure une phase de démultiplexage en amont du traitement et de multiplexage en aval. Cependant, nous avons vu que ce type de solution est

un frein à l'intégration des médias puisqu'elle ne permet pas de considérer chaque média de manière indépendante.

2.4.2 Le Transport Distribué

Le deuxième type de transport que l'on rencontre dans une AMD consiste à diffuser les médias entre les différents sites qui la composent à l'aide du réseau Internet[18]. Le transport d'informations sur ce réseau d'une machine émettrice vers une machine réceptrice est réalisé par la couche transport de la pile de protocole Internet [KUR03]. Cette couche est chargée de la transmission des données issues de la couche application en ayant recours à des protocoles de transport comme TCP (Transmission Control Protocol) ou UDP (User Datagram Protocol) pour les plus courants. Elle propose également d'autres protocoles plus spécifiques dont l'objectif est de répondre à des besoins particuliers, c'est le cas par exemple de RTP (Real-Time Transport Protocol). Au travers de la présentation spécifique de leurs caractéristiques, nous allons essayer d'identifier les problèmes induits par ce type de transport à travers l'Internet dans des cas de transmission de médias.

2.4.2.1 Le Réseau Internet

Depuis maintenant une quinzaine d'années, les évolutions de l'informatique et des télécommunications ont conduit à une modification radicale du paysage de la communication informatique et, en conséquence, de l'Internet et de ses services. De nos jours, l'utilisation de l'Internet a évolué dans le sens où le nombre d'applications a fortement augmenté et les services rendus se sont diversifiés. Outre les données traditionnelles, on transmet maintenant des données multimédias telles que nous les avons présentées auparavant.

Le réseau Internet procure un service « best-effort » (« au-mieux ») pour les données qu'il véhicule. Ce service est fourni par la couche réseau de la pile de protocoles de l'Internet et plus précisément par le protocole IP (Internet Protocol) [KUR03] qui est chargé du routage des paquets de données sans aucune garantie d'ordre de réception des paquets ni de fiabilité (possibilité de pertes de données). Ce service est de fait inadapté pour les AMD et certaines applications émergeantes. La raison principale est que les AMD manipulent des médias sensibles au temps dont la sémantique dépend du respect des relations de synchronisation multimédia mais aussi

[18] Notre étude se limite aux AMD sur le réseau Internet largement utilisé comme infrastructure de support à ce type d'applications.

de la restitution des données dans l'ordre d'acquisition/création. Le service « best-effort » n'étant pas flexible, il est du ressort des AMD de s'adapter à ce dernier.

2.4.2.2 La Couche Transport

La couche transport se situe entre la couche réseau et la couche application et c'est à elle qu'incombe le rôle essentiel de fournir des services de transport exploitables par les AMD.

2.4.2.2.1 Définition

Le rôle de cette couche est de créer un canal de communication de bout en bout [OWE00]. Il est dédié aux échanges de données entre les programmes applicatifs sans se préoccuper des caractéristiques de l'infrastructure physique utilisée pour les véhiculer qui sont gérées par les couches sous-jacentes. Les données provenant de la couche application sont divisées en unités plus petites appelées paquets et transmises à la couche réseau par l'intermédiaire de la couche transport. Ce canal doit pouvoir être bidirectionnel afin que les deux extrémités puissent s'échanger des données. Cette couche dispose des services fournis par les couches sous-jacentes et introduit en outre des services de transport. Ce principe de transport est décrit sur la Figure 7.

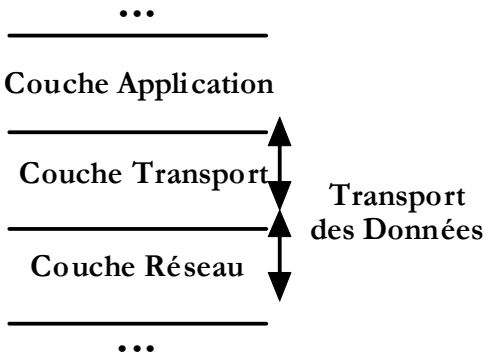


Figure 7 Principes de Transport des Données

2.4.2.2.2 Les Protocoles de Transport

Les services de transport fournis sont regroupés au sein de protocoles qui spécifient les règles d'utilisation. Les protocoles de transport les plus utilisés sur l'Internet sont TCP et UDP qui sont implémentés sur toutes les machines connectées à l'Internet. Nous présentons également le protocole RTP qui est, quant à lui, utilisé dans certaines AMD.

Le Protocole TCP

TCP, documenté dans la RFC 793 [POS81], est un protocole orienté connexion[19] conçu pour fournir un service de transport de données de bout en bout du réseau. Il fournit des services de fiabilité et d'ordre qui constituent ses atouts majeurs. Lorsqu'une erreur de transmission se produit, elle est détectée par non-acquittement du destinataire et corrigée par retransmission des données. De plus, TCP garantit la réception des données dans l'ordre d'émission. Hormis ces services, TCP met en œuvre un mécanisme important pour assurer le bon fonctionnement du réseau : le contrôle de congestion. La congestion se présente lorsque trop de sources tentent de transmettre trop de données, ce qui a pour effet d'augmenter les pertes de données et donc les délais de transmission. Ce protocole propose alors de s'adapter à la bande passante disponible en diminuant son débit lorsqu'une perte est détectée.

Malgré sa robustesse et ses performances élevées, TCP présente quelques inconvénients en particulier pour le transport des médias sensibles aux délais. En effet, si les mécanismes implémentés par TCP sont très intéressants lorsque l'on se place du point de vue de la fiabilité des transmissions, ils se révèlent peu efficients du point de vue des délais de transmission. De par les possibilités de réémission en cas d'erreur de transmission, TCP introduit de la latence et de la gigue entre les paquets reçus, c'est à dire des délais variables pouvant être largement supérieurs aux contraintes temporelles de type intra-flux des médias continus. De plus, la retransmission des paquets perdus augmente la quantité de données envoyées à travers le réseau. Ceci peut donc le rendre incompatible avec les contraintes des médias continus. Certains travaux (voir à ce sujet [KRA01]) préconisent malgré tout son utilisation pour certains types d'AMD, en particulier celles qui mettent en œuvre des techniques de streaming. Les services de contrôle de congestion et de retransmission sont considérés comme importants dans ce type d'AMD. Ces travaux s'orientent vers la proposition de solutions permettant de bénéficier de ces avantages. En effet dans [KRA01], les auteurs proposent la gestion de tampons mémoire et l'adaptation de la qualité des médias transmis avec pour objectif la réduction de la gigue introduite par TCP. Le plus souvent ces recherches visent à proposer de nouvelles versions de TCP plus rapides et offrant une gigue réduite (voir à ce sujet les recherches concernant Fast-TCP [JIN03] et les TCP-like [CES97], [PAD99]) c'est à dire plus adaptées au transport de médias continus.

---

19 Une connexion doit être établie avant toute transmission.

Le Protocole UDP

UDP, documenté dans la RFC 768 [POS80], est un protocole sans connexion[20] conçu pour fournir un service de transport de données de bout en bout du réseau sans garantie d'ordre ni de fiabilité. Ce protocole de transport est plus simple et plus léger que TCP car il assure un service minimum qui est l'acheminement des données.

Grâce à son service de transmission non fiable, UDP n'introduit quasiment pas de gigue et de latence. Il est préféré à TCP par les applications qui n'ont pas besoin d'un service de transport fiable. En effet dans ces applications, il est souvent préférable de gérer les pertes de données plutôt que d'essayer de les éviter. Certains médias supportent des pertes de données dans des limites raisonnables dés lors qu'elles ne sont pas détectables lors de leur restitution. Par exemple, il n'est pas dramatique de perdre trois ou quatre images de temps en temps dans un flux vidéo dont le rythme est de 25 images par seconde. Voir à ce sujet, pour une étude quantifiée, les études sur la perception humaine de la synchronisation des médias [GHI98], [STE96], [WEI98].

Le Protocole RTP

RTP, documenté dans la RFC 1889 [SCH96], est un protocole proposé pour transmettre des médias continus comme l'audio et la vidéo. Il est considéré comme un protocole de transport, mais il constitue en réalité une surcouche du protocole UDP. Ceci signifie que les paquets UDP sont encapsulés dans des paquets RTP qui contiennent, entre autre, une entête permettant l'identification du contenu du flux transporté, un numéro de séquence et une référence au temps. Il fournit des services tels que la détection de pertes et la sécurité. La référence au temps peut être utilisée pour déterminer quand doit être diffusé un paquet par rapport à un autre mais aussi pour synchroniser différents flux. Cependant cette tâche doit être mise en œuvre au niveau de la couche application à l'aide des informations fournies dans les paquets RTP. Le numéro de séquence permet de réordonner les paquets et sert également pour la détection des pertes de données. Grâce à ces informations l'entité qui reçoit les données peut savoir comment les diffuser (notions d'ordre et de relations temporelles).

RTP offre une couche supérieure de fonctionnalités permettant de mettre en place des mécanismes de canal de retour[21]. Cette possibilité peut être utilisée afin de gérer les transmissions par exemple en informant l'émetteur sur les propriétés du canal de transmission, sur l'état du tampon du récepteur ou pour demander des changements de format ou de débit des données. Ces mécanismes peuvent être réalisés grâce

[20] Un protocole sans connexion ne nécessite pas l'établissement d'une connexion avant chaque transmission.

[21] Appelés feedback en langue anglaise.

aux informations fournies par le protocole RTCP (Real-Time Transport Control Protocol) [SCH96] qui va de pair avec RTP. RTP et RTCP sont utilisés par un grand nombre d'AMD à travers l'Internet. Les travaux suivants en sont des exemples : [BOU03], [CHA00], [CHE95], [CHE03], [EID02], [GAU98], [GRI03], [HAG02], [HAM04], [LAY04].

Bien que RTP fournisse une approche intéressante pour le transport des médias continus, son utilisation est inappropriée dans les AMD telles que nous les définissons. RTP utilise des formats spécifiques pour transporter les données[22], ce qui signifie que certains types de données non prévus par le protocole ne peuvent être diffusés à moins de l'étendre et donc de proposer une solution peu flexible (comme dans le § 2.3.2, il est impossible de donner une liste exhaustive de tous les types de données manipulés par une AMD a priori). Un point intéressant est que RTP fournit des informations qui permettent de restituer deux propriétés essentielles des médias continus qui sont les relations de synchronisation et la séquence des données.

2.4.2.2.3 Synthèse

A travers ce rapide aperçu, nous avons pu aborder les principaux avantages et inconvénients des protocoles de transport les plus communs (TCP et UDP) ainsi que RTP qui est plus spécifique mais s'intéresse aux transferts de médias continus. Dans notre contexte de recherche, nous pensons qu'il est difficile de choisir l'un de ces protocoles. Dans un contexte de gestion de la qualité de service, il paraît intéressant de pouvoir proposer différentes possibilités de transport. Il paraît également intéressant que les AMD soient adaptatives afin d'offrir un service optimal pour tous ses utilisateurs. Selon les qualités requises, selon les possibilités de l'environnement d'exécution et selon les médias à transporter on peut choisir un protocole de transport fiable ou non. Nous ne faisons donc pas de choix particulier quant à ces protocoles et nous laissons la possibilité de pouvoir tous les utiliser indifféremment.

Nous avons également abordé les principales caractéristiques du protocole RTP qui n'est pas aussi répandu que TCP du fait de sa spécificité. L'approche proposée est intéressante car elle permet la transmission de médias continus synchrones de bout en bout du réseau ce qui a permis de populariser son utilisation. Néanmoins, dans les AMD cette possibilité est nécessaire mais pas suffisante en raison du fait que les traitements de certains médias introduisent des retards par rapport à ceux qui ne sont pas traités et sont donc susceptibles en amont et en aval des transmissions de détruire les relations temporelles de synchronisation de type inter-médias. Lorsque l'on veut

[22] Voir à ce titre l'implémentation de RTP fournie par l'API JMF du langage de programmation Java [SUN99].

concevoir et développer des AMD, il est nécessaire de considérer cet aspect peu abordé dans la littérature. En conséquence, nous pensons que les relations de synchronisation doivent être conservées de bout en bout dans une AMD et ce malgré les traitements et les transferts réseau. Ainsi, nous n'utiliserons pas le protocole RTP qui est insuffisant dans notre approche. En revanche, RTP utilise dans ses paquets des informations intéressantes comme une étiquette temporelle et un numéro de séquence qui permettent de reconstituer les principales caractéristiques des médias. Nous avons choisi d'utiliser de telles informations durant tout le transit des médias à l'intérieur d'une AMD.

3 La Qualité de Service dans les Applications Multimédias Distribuées

La démocratisation de l'utilisation des réseaux de communication a permis d'introduire la notion de Qualité de Service (QdS) qui se propose alors d'étudier les services rendus par tout élément participant à une interaction réseau (par exemple un protocole) en terme de niveau de qualité proposé. Par exemple, un service de transport fiable comme celui proposé par TCP (cf. section précédente) possède un niveau de qualité élevé. Un autre exemple, plus lié à un type de réseau, concerne le service « best-effort » fourni par le réseau Internet. Ce service ne fournit aucune garantie quant à la délivrance du service en termes de débits et de temps de transmission qui dépendent de la charge du réseau au moment où on l'utilise.

En raison de l'abondance des applications distribuées ces dernières années, la QdS a tout naturellement fait son apparition dans les phases de conception de ces dernières. En effet, les applications déployées sur un réseau sont fortement dépendantes de son fonctionnement. Les AMD déployées sur le réseau Internet doivent donc pouvoir s'adapter à son mode de fonctionnement et aux environnements hétérogènes (logiciels et matériels) qu'il supporte.

De nombreux travaux ont tenté de répondre à ces préoccupations. Ces recherches s'inscrivent dans le domaine de la Qualité de Service (QdS) dont l'objectif est de proposer des niveaux de qualité différents aux utilisateurs des AMD et donc indirectement des réseaux. La QdS ne fait pas l'objet actuellement d'une approche consensuelle en raison des multiples domaines d'application. Elle est donc abordée selon différents points de vue. En ce qui nous concerne, les travaux précédents [LAP06] ont consisté à la définir selon deux points de vue qui semblent intéressants pour décrire la QdS d'une AMD, à savoir ceux de l'utilisateur et de l'environnement d'exécution.

La prise en charge de la QdS dans les AMD nécessite de revoir les méthodes de conception habituelles et donc de définir et de mettre en place des mécanismes dédiés à la gestion de la QdS. Plusieurs solutions sont alors proposées dans la littérature. Le choix d'une approche par rapport à une autre dépend des caractéristiques dont on veut doter le système visé et des propriétés qu'il impose (par exemple par l'utilisation du réseau Internet). La dernière partie de ce chapitre présentera quelques approches significatives du domaine de la gestion de la QdS.

3.1 Définition de la Qualité de Service

La QdS est un terme vaste qui englobe de nombreux concepts. Toutefois, cette notion est intimement liée à la finalité de l'application elle-même. Ainsi dans une application de calcul scientifique, elle pourra se mesurer en terme de précision et de fiabilité des résultats alors que dans une application interactive, elle devra tenir compte de la réactivité et de la forme des informations présentées aux utilisateurs.

La QdS est également liée aux entités logicielles et matérielles utilisées pour la réalisation d'une application. En effet, elle peut traduire la capacité de ces entités à fournir un service avec des niveaux différents de qualité donc de différentes manières. Dans le cas des AMD, on peut par exemple transmettre un flux vidéo avec différentes qualités en fonction de la bande passante du réseau utilisé (flux compressé, taille de l'image, résolution, couleur ou noir et blanc).

L'organisme international de normalisation ISO (International Standard Organization) définit la QdS comme un ensemble de caractéristiques qui se rapporte au comportement collectif d'un ou plusieurs objets [ISO95]. L'ISO situe la QdS par rapport à la performance d'un ensemble d'objets qui elle-même est définie par les infrastructures qui les supportent. On évalue de la sorte le comportement localisé ou global d'une entité informatique.

Ainsi, la QdS peut être abordée selon plusieurs points de vue car elle dépend de différents paramètres. Les méthodes de spécification et de développement d'applications introduites par le génie logiciel ont pour objectif de répondre à des besoins exprimés par des clients. Les utilisateurs se trouvent alors au centre de ces approches puisqu'ils sont à la base de ces démarches de conception. Il paraît donc évident de placer de même les utilisateurs au centre de l'approche de QdS.

La conception d'une application intégrant la QdS aura pour principal objectif de fournir aux utilisateurs des services correspondant à leurs souhaits. Il s'agit dans un premier temps de pouvoir atteindre la qualité demandée par chacun d'eux. Toutefois, l'obstacle majeur à l'obtention et au maintien de cette QdS est l'environnement de dé-

ploiement des applications. En effet, si les diverses phases de conception peuvent tenir compte des environnements de déploiement, la notion de distribution complique ce travail puisqu'elle introduit le fait qu'une application est déployée sur plusieurs sites. De plus, lorsqu'elles le sont sur un réseau comme Internet, elles sont soumises à une forte hétérogénéité qu'il est nécessaire de considérer afin de permettre leur fonctionnement quelles que soient les caractéristiques des sites concernés. Enfin, les spécificités du réseau Internet dépendent de sa charge à un instant donné, ce qui implique qu'elles sont amenées à évoluer pendant l'exécution d'une application. Il paraît donc nécessaire de considérer également l'environnement d'exécution et ses évolutions dans une approche de QdS dynamique [LAP06].

3.1.1 Le Point de Vue de l'Utilisateur

La QdS du point de vue de l'utilisateur se traduit par la perception qu'il a du service. Le CCITT (Comité Consultatif International Téléphonique et Télégraphique) donne une définition qui rejoint cette idée[23]. Nous appelons ce point de vue la QdS requise car il permet d'identifier les exigences de l'utilisateur vis à vis d'une application. Ces exigences sont propres à chaque utilisateur car chacun a une perception différente des services fournis par une application.

En raison de sa subjectivité, ce point de vue est critique car il est difficile à évaluer. De fait, beaucoup d'applications et qui plus est d'AMD ne le considèrent pas et préfèrent orienter leur gestion de la QdS sur le point de vue de l'environnement d'exécution. Hafid *et al.* [HAF98] le définissent comme des critères de QdS subjectifs car ils ne sont pas directement mesurables. Malgré cette difficulté, nous pensons qu'il est important de considérer ce point dans les AMD car il en dépend de l'intérêt que le public porte sur ces dernières. En effet, il nous est à tous arrivé de nous détourner d'une application qui ne répondait pas directement à nos exigences. Cette constatation nous conforte dans le fait qu'il est impératif de tenir compte de ce point de vue dans les AMD. Certains travaux décident de le considérer dans une démarche de QdS afin par exemple de définir des critères qualitatifs ainsi que des exigences propres aux utilisateurs [GUX02].

---

23 « Quality of service is the collective effect of service performance which determines the degree of satisfaction of a user of the service » [CCI89].

3.1.2 Le Point de Vue de l'Environnement d'Exécution

Le second point de vue est celui de l'environnement d'exécution d'une application. Il traduit la capacité du contexte d'exécution à supporter l'application qui y sera déployée. Lorsque l'on parle de contexte d'exécution, on considère l'environnement informatique[24] logiciel et matériel (réseau, périphériques, etc.). Ce point de vue est également important car il va permettre de déterminer si le service proposé par une application peut être fourni. De plus si on le couple au point de vue précédent, il permet également de vérifier que le service requis par un utilisateur peut être rendu ou pas. Vogel *et al.* donnent une définition de la QdS qui se base sur ce point de vue[25]. Nous appelons ce point de vue la QdS fournie car il traduit les capacités des environnements d'exécution à rendre un ou plusieurs services.

Ces paramètres de QdS sont plus faciles à évaluer que les précédents puisqu'ils sont directement observables et mesurables. Hafid *et al.* [HAF98] parlent à ce sujet de critères de QdS objectifs.

3.1.3 Synthèse

Ces observations permettent de classer les critères de QdS selon deux axes que nous pensons nécessaires à prendre en considération dans le développement des applications actuelles afin de susciter un large intérêt auprès des utilisateurs. En effet, si l'on couple ces deux points de vue alors on peut traduire la capacité des applications déployées sur des environnements d'exécution à fournir les services requis par les utilisateurs. De plus, pour des applications capables de fournir plusieurs niveaux de qualité, on peut choisir l'un de ces niveaux en trouvant un compromis entre la QdS requise et la QdS fournie. La QdS globale d'une application est donc définie par ces deux critères, nous proposons la définition suivante[26].

---

24 La plupart des logiciels commercialisés actuellement imposent une configuration logicielle et matérielle minimale pour pouvoir être installés et correctement exécutés. Ces contraintes font parties du point de vue de l'environnement d'exécution et sont imposées par les constructeurs. L'inconvénient majeur de ce type de démarche est qu'il n'existe pas plusieurs niveaux de qualité et donc d'exigence dans les logiciels actuels, ce qui veut dire que si un utilisateur ne possède pas la configuration requise il ne pourra pas accéder au service. Ainsi, ce type d'applications se révèle peu flexible.

25 « By quality of service we mean the set of those technical and other parameters of a distributed multimedia system, which influence the presentation of multimedia data to the user » [VOG94].

26 « Evaluer la qualité de service consiste à mesurer l'adéquation entre le service désiré par l'utilisateur et le service qui lui est fourni » [LAP06].

3.2 Gestion des contraintes de Qualité de Service

Gérer la QdS consiste à mettre en place des mécanismes qui vont permettre d'établir et de maintenir une QdS globale dans les AMD en fonction d'un ou des deux points de vue définis précédemment. La gestion de la QdS se base donc sur ces paramètres.

Une telle gestion nécessite dans un premier temps de collecter les paramètres de QdS sur lesquels on va pouvoir se baser, c'est une phase d'évaluation. Ces informations doivent par la suite pouvoir être traduites en termes de répercussion sur l'application dont on veut gérer la QdS (QoS-Mapping). Cette tâche permet de déduire si le service peut être rendu et avec quel niveau de qualité.

La gestion de la QdS peut être mise en place à différents moments de l'exécution d'une application. Lorsqu'elle se situe avant l'exécution d'une application, on parle de gestion statique. Lorsqu'elle s'effectue pendant l'exécution d'une application, on parle de gestion dynamique. Dans un contexte statique, le système de gestion doit essayer de prévoir si le service pourra être rendu selon les paramètres collectés avant l'exécution. Dans un contexte dynamique, l'application doit se doter d'un système de surveillance chargé de récupérer à tout moment les paramètres pertinents, de détecter les variations de QdS et de proposer des plans d'actions afin d'adapter la qualité fournie en considérant les paramètres collectés.

Les mécanismes de gestion de la QdS sont liés à l'implémentation non-fonctionnelle d'une application car ils concernent des aspects de l'application externes à sa fonctionnalité première (celle pour laquelle elle a été développée). Ces notions sont empruntées à la programmation orientée aspect [BOU01] qui prône, entre autre, une séparation claire et précise des aspects fonctionnels et non-fonctionnels d'une application. Diverses solutions sont possibles pour l'implémentation de la gestion de la QdS. On peut par exemple l'intégrer au code de l'application mais aussi définir une entité supplémentaire chargée de mettre en œuvre ces mécanismes. Dans ce cas, on parle d'intergiciel, de middleware ou de plate-forme d'exécution.

Les mécanismes de gestion de la QdS peuvent être abordés de différentes façons. En effet, on peut considérer que c'est l'environnement d'exécution d'une application qui doit s'adapter aux besoins de cette dernière. Une autre façon de voir les choses est de dire que ce sont les applications qui doivent adapter leur fonctionnement à ce que permet l'environnement d'exécution. La suite donne un exemple de la gestion de la QdS selon ces deux approches.

3.2.1 La Réservation de Ressources

Cette approche repose sur l'allocation et l'ordonnancement des ressources de l'environnement d'exécution (principalement les ressources réseau et système) afin de permettre l'exécution d'une application selon des critères de QdS définis. Ces critères sont ensuite exprimés en termes de besoins en ressources (processeur, mémoire, bande passante, délai, gigue, etc.).

Le contrôle d'admission permet d'évaluer si les ressources requises par une application sont disponibles. Si c'est le cas les ressources sont réservées pour l'application. Sinon, une phase de négociation est établie entre l'application et l'environnement d'exécution. L'issue de cette phase doit permettre de trouver un compromis entre les ressources disponibles et celles requises par l'application afin de fournir une QdS négociée. Ces spécifications sont définies à l'aide de contrats de QdS. Cette tâche aboutit à un accord entre l'application et l'environnement d'exécution. La finalité est de réserver les ressources nécessaires définies par l'accord passé entre les deux parties. Comme l'application peut s'exécuter en sachant totalement ce dont elle dispose, la garantie de service permet d'assurer à l'utilisateur une QdS fixe. Par contre, son utilisation est restreinte aux environnements d'exécution qui offrent cette possibilité.

Différents travaux préconisent ces solutions, en voici quelques exemples : [AND93], [BAI96], [GAA01], [RAY87], [TOK92].

3.2.2 L'adaptation de l'application

La seconde approche consiste à adapter le comportement des applications avec pour objectif d'approcher leur fonctionnement de la QdS requise. Ces applications sont dites élastiques ou opportunistes.

L'objectif ici est de collecter les paramètres de QdS que l'on veut gérer (QdS requise et/ou QdS fournie). Comme pour les systèmes à réservation de ressources, on va essayer d'évaluer à travers une phase de négociation la QdS que pourra fournir l'application en fonction des paramètres collectés. Suite à quoi, le comportement de l'application peut alors être adapté à l'aide d'un plan d'action qui permet de traduire le compromis trouvé en des phases d'adaptation à mener sur l'application. La plupart du temps, ces adaptations reposent sur des phases de reconfiguration de la structure de l'application. Cette approche nécessite de posséder des architectures logicielles malléables afin de pouvoir réaliser ces adaptations.

Diverses solutions sont utilisées pour mettre en œuvre cette politique de gestion de la QdS. Les applications auto-adaptables modifient leur comportement afin de

fournir la QdS négociée. Des mécanismes à boucle de retour (feedback) permettent de surveiller un ou plusieurs événements puis d'adapter les comportements qui sont liés à l'évolution de ces derniers. Ces mécanismes sont basés sur les principes des systèmes asservis[27] [CEN97]. Certains travaux déploient des intergiciels, middlewares ou plates-formes d'exécution chargés de procéder à la gestion de la QdS par adaptation de l'application.

Ces solutions se retrouvent dans de nombreux travaux dont voici quelques exemples : [DIO95], [KON00], [LAY04], [SIN99].

3.3 Synthèse

Cette partie donne une vision sommaire du domaine de la QdS et plus particulièrement de la gestion de la QdS dans les AMD. Le lecteur intéressé par une étude plus complète pourra se référer à [LAP06] qui fournit un état de l'art plus conséquent de ce domaine puis détaille la plate-forme chargée de la gestion de la QdS.

Deux mécanismes possibles pour la gestion de la QdS sont abordés. Le premier propose une gestion par réservation des ressources. Cette solution est très utilisée mais n'est pas envisageable dans tous les cas. En effet, il est nécessaire que l'environnement support de ces applications fournisse cette possibilité. Dés lors que l'on s'intéresse à des applications distribuées à travers des réseaux de type Internet, une telle solution ne peut être choisie car l'Internet ne permet pas la réservation des ressources. Cette approche n'a donc pas été retenue et donc nous nous sommes orientés vers une autre solution. La seconde approche propose une adaptation des applications aux paramètres de QdS que l'on désire considérer. Cette solution consiste à rendre les applications malléables [BOU00] afin qu'elles puissent s'adapter à leurs environnements et/ou à leurs utilisateurs. Notre intérêt pour les AMD à travers l'Internet nous a conduit à choisir ce type de solution pour la gestion de la QdS [LAP06]. Dans la solution proposée, les points de vue de l'utilisateur et de l'environnement d'exécution sont tout deux considérés. Ainsi, les AMD sont capables de s'adapter aux exigences des utilisateurs suivant les capacités des environnements d'exécution sur lesquels elles sont distribuées. Les adaptations sont menées à l'aide d'une plate-forme d'exécution chargée de la gestion de la QdS. Ces travaux sont détaillés dans la deuxième partie du présent mémoire.

[27] Ce genre de mécanisme est basé sur le principe des asservissements en automatique qui ont pour principe d'adapter le fonctionnement d'un système en fonction de ce qu'il fournit en sortie. Le but étant de se rapprocher le plus possible de la consigne donnée en entrée du système.

Nous avons à travers la première partie de ce chapitre un aperçu du domaine du multimédia et de ses principales notions. Cette présentation est focalisée sur les médias des AMD car nous pensons que c'est un concept central lorsque l'on s'intéresse à ce type de développement. L'objectif de la partie suivante est d'illustrer l'ensemble de ces notions à travers la présentation de quelques travaux issus de la littérature. Puis, en conclusion de ce chapitre nous nous attacherons à comparer ces travaux avec notre approche afin de mettre en évidence l'apport de cette thèse.

4 Quelques Travaux de Recherche sur les Applications Multimédias Distribuées

Avant de refermer ce chapitre sur le domaine des AMD, nous allons évoquer à titre d'exemple quelques travaux qui portent sur tout ou partie des points présentés. Ainsi, nous allons mettre en évidence les solutions introduites par ces réalisations afin de répondre aux problématiques de conception et de développement des AMD. Nous pourrons ensuite terminer ce chapitre en positionnant notre approche par rapport à ces travaux puis en citant les points sur lesquels nous nous attarderons.

Les travaux abordés sont regroupés en trois parties :

- la première s'intéresse à donner quelques exemples d'AMD ;
- la deuxième présente des travaux ayant pour objectif d'assurer la synchronisation multimédia dans les AMD ;
- enfin la troisième et dernière partie présentera quelques travaux qui mettent en place des systèmes de gestion de la QdS dans les AMD.

4.1 Des Exemples d'Applications Multimédias Distribuées

Nous commençons par présenter deux exemples d'AMD. Le premier traite d'un navigateur Internet qui permet la diffusion de pages webs intégrant différents types de médias. Le second concerne une AMD de type vidéo à la demande.

4.1.1 Le Navigateur Internet Vosaic

La navigation sur le réseau Internet est rendue possible à l'aide de logiciels appelés navigateurs. Le navigateur Vosaic [ANN93], [CHE95] est l'un des précurseurs, il propose la diffusion de pages web intégrant des médias continus et discrets.

Son architecture, décrite sur la Figure 8, se compose de trois couches distinctes :

- une couche de transport est chargée de récupérer les médias à diffuser en provenance du réseau à l'aide de différents protocoles de transport ;
- une couche de décodage permet la décompression des médias reçus par la couche supérieure ;
- une couche de diffusion permet la restitution des médias en fonction des données HTML des pages à afficher.

Vosaic fonctionne à l'aide d'un serveur HTTP (HyperText Transfer Protocol) [FIE99] étendu qui utilise les protocoles de transport supportés par Vosaic. Un dispatcheur reçoit les requêtes HTTP. Elles sont triées suivant le type des données requises et traitées séparément. Un contrôleur d'admission est utilisé afin de déterminer et d'estimer les exigences en terme de ressources clientes nécessaires (bande passante, charge processeur, etc.) des requêtes à servir. Le serveur prend alors des décisions sur la faisabilité des requêtes basées sur les connaissances des environnements clients à servir.

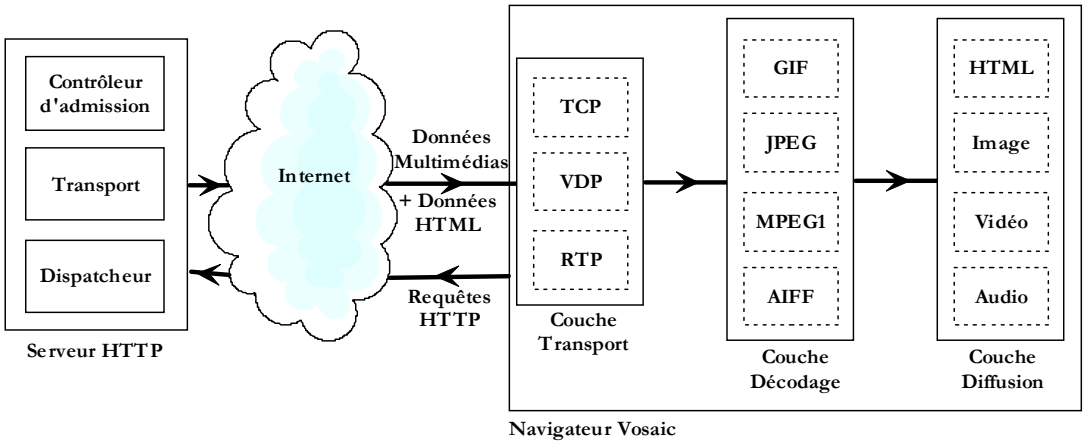


Figure 8 Architecture du Navigateur Vosaic

Le serveur complète les données à transmettre à l'aide de méta-informations qui vont permettre de connaître leur type. Ainsi, les données sont transmises en choisissant un protocole approprié suivant leur type : TCP pour le texte et les images, VDP (Video Datagram Protocol) pour les médias continus et RTP pour des formats particuliers. Ces méta-informations sont également utilisées côté client. En effet, lors de la réception des données une composition dynamique des modules à utiliser dans chaque couche est effectuée (ils ne sont pas tous représenté sur la Figure 8).

Le protocole VDP a été spécifié et développé comme une surcouche de RTP. Il permet de réduire significativement la gigue introduite par le réseau Internet et propose d'adapter dynamiquement la transmission en fonction de la charge processeur de

la machine cliente et du taux de congestion du réseau. Il se base sur un mécanisme de type boucle de retour (feedback) afin de disposer des informations nécessaires à de telles adaptations par augmentation ou dégradation de la qualité des médias transmis (cf. [CEN97]).

Ce navigateur permet la diffusion de médias continus et discrets en temps-réel à travers l'Internet. Vosaic, quoiqu'un peu ancien, utilise une idée fort intéressante qui se base sur la composition dynamique de modules en fonction des données transmises. Par exemple, au niveau de la couche transport on définit des points communs entre les caractéristiques des médias et celles des protocoles de transport. Vosaic utilise des protocoles de transport non fiables pour des médias comme la vidéo qui supportent les pertes de données. Cette composition est guidée par les données puisqu'elles sont prépondérantes dans le choix des modules à utiliser et ce dans chaque couche du navigateur.

Un autre point fort que l'on retrouve dans de nombreux travaux est l'idée d'adapter les médias côté serveur pour la transmission soit par augmentation, soit par dégradation de leur qualité en fonction des caractéristiques de l'environnement d'exécution (réseau + machines clientes) à travers lequel les médias vont transiter. De fait, on met en place un système de gestion de la QdS des médias. La qualité des médias est adaptée aux capacités de l'environnement d'exécution.

Nous considérons cependant comme une faiblesse le fait d'utiliser des formats prédéfinis pour la manipulation des médias. Ce genre de solution est peu flexible puisqu'elle impose de disposer des codecs[28] nécessaires pour décompresser chacun des médias reçus avant leur diffusion. Vosaic supporte donc un nombre fini de formats. Qu'en est-il si un format n'est pas supporté par le navigateur ? Nous pensons qu'il est préférable de s'affranchir des formats, sans exclure leur utilisation, afin d'offrir plus de flexibilité et de souplesse dans la manipulation et le transfert des médias.

4.1.2 Vidéo à la Demande adaptable par un code mobile

L'application de vidéo à la demande présentée dans [HAG02] se base sur le modèle client/serveur. Un serveur permet de distribuer des vidéos vers des machines clientes via le réseau Internet. Les flux délivrés par ce serveur passent à travers un cer-

28 Un codec (COmpression DECompression) est un mécanisme qui permet la compression et la décompression de médias comme l'audio et la vidéo. Ce mécanisme est fourni soit de façon logicielle, soit de façon matérielle. L'objectif d'un codec est de réduire de manière significative la taille des données. Ils sont souvent utilisés pour la transmission de données.

tain nombre de sites intermédiaires appelés mandataires[29]. Ce sont des couches d'adaptation entre les clients et les serveurs. Leur fonction est d'adapter les caractéristiques des flux vidéo à destination du client aux capacités de son point d'accès et aux conditions courantes du réseau. Pour ce faire, un mandataire utilise ses propres ressources de stockage et de traitement.

Les flux vidéo qui transitent à travers les mandataires sont adaptés à l'aide d'un code mobile. Une plate-forme à agents mobiles est utilisée pour réaliser les adaptations. Les mandataires constituent des supports aux adaptations, ils sont configurés à distance par des agents. Les mandataires sont capables de répartir entre eux la tâche d'exécution des adaptations afin de soulager certains de la charge induite par le traitement mais aussi de limiter les interactions entre eux dans le cas où l'adaptation doit utiliser les ressources de différents mandataires.

L'objectif est d'améliorer les performances de l'application par adaptation des flux vidéo aux environnements d'exécution des clients. De plus, ces adaptations contribuent à d'autres tâches comme par exemple la personnalisation du service fourni suivant les souhaits des utilisateurs ou le traitement de pannes temporaires.

Le client se connecte à un mandataire grâce à l'URL (Uniform Resource Locator) de la vidéo qu'il souhaite visionner. Le mandataire accède à la vidéo requise à l'aide du protocole HTTP [FIE99]. La vidéo est transmise du mandataire vers le client à l'aide du protocole RTP [SCH96]. L'application est développée en Java à l'aide de la bibliothèque JMF[30] (Java Media Framework) [SUN99].

Cette recherche introduit l'adaptation de médias à l'aide de mandataires afin de répondre à plusieurs objectifs : l'amélioration des performances, le traitement de pannes temporaires et la personnalisation du service. Elle a retenu notre attention car elle permet l'adaptation des AMD par celle des médias qu'elles supportent. Les adaptations mises en place permettent de considérer les deux points de vue de la QdS abordés précédemment (cf. § 3.1). Nous pensons que l'adaptation des médias est une solution intéressante et nécessaire dans le contexte des AMD à travers un réseau aussi hétérogène que l'Internet. De plus, l'utilisation du langage Java dans un tel environnement semble justifiée par sa portabilité.

---

29 En anglais, le terme mandataire est traduit par le mot proxy.

30 JMF est une bibliothèque qui permet la création, la manipulation, le traitement, le transfert et la restitution de médias au sein d'applications ou d'applets Java.

4.2 Gestion de la Synchronisation dans les Applications Multimédias Distribuées

Nous présentons maintenant des travaux ayant trait à la gestion de la synchronisation dans les AMD. La première approche définit une politique de synchronisation dont le but est d'assurer la restitution synchrone de plusieurs médias. Les autres recherches modélisent des scénarios de synchronisation destinés à être utilisés dans les AMD avec les mêmes finalités.

4.2.1 Le Modèle MultiSync

MultiSync [CHE96] est un modèle de synchronisation destiné à des environnements multi-processus. L'idée est de partager la tâche de synchronisation entre plusieurs processus chargés de diffuser des médias. Ce modèle permet l'implémentation de lecteurs multimédias qui sont des dispositifs très utilisés dans les AMD. L'objectif est d'assurer la diffusion synchrone de médias continus et discrets malgré les surcharges de l'environnement d'exécution sous-jacent susceptibles d'altérer une telle diffusion.

Deux types de processus sont distingués. Les processus médias sont chargés de la diffusion des médias et du respect de la synchronisation. Les processus de contrôle sont responsables de la gestion des interactions avec l'utilisateur (lecture, pause, avance rapide, retour rapide, arrêt) ainsi que de la gestion du cycle de vie des processus médias (création et destruction). Le modèle introduit la notion de priorités entre les médias. Les médias considérés comme les plus importants se voient attribuer les priorités les plus hautes. Ces priorités sont donc définies lors de la phase de conception et dépendent des médias manipulés et du type d'application visé. Elles sont définies au sein de chaque processus média.

Les synchronisations intra- et inter-médias sont assurées par la coopération des processus médias concernés. Deux politiques sont proposées pour la réalisation de cette tâche :

- une synchronisation relative permet de synchroniser les processus médias entre eux en utilisant des techniques de communication interprocessus ;
- une synchronisation absolue propose d'utiliser pour chaque processus média le même référentiel temporel, à savoir l'horloge physique du système.

Avant la phase de diffusion des médias, seul un processus de contrôle est actif. Lorsqu'un utilisateur requiert la diffusion de médias en appuyant sur le bouton de lecture, un événement est levé pour indiquer la requête au processus de contrôle. Les in-

formations temporelles nécessaires sont récupérées par ce dernier à l'aide desquelles il tient à jour une table d'informations temporelles qu'il va utiliser comme points de référence pour assurer la synchronisation. Chaque enregistrement de cette table indique la durée de diffusion de chaque unité appartenant à un média (date de début et de fin). Ensuite, le processus de contrôle génère un processus pour chaque média à diffuser. Les informations temporelles de la table sont communiquées aux processus médias correspondants. La diffusion synchrone débute alors. Le modèle fournit des mécanismes de gestion de la diffusion afin d'éviter les désynchronisations ou autres ruptures (par exemple attendre ou rejeter les images d'une vidéo, cf. [CHE96]). A la fin de la diffusion ou lorsque l'utilisateur appuie sur le bouton stop, le processus de contrôle met fin aux actions de diffusion par destruction des processus médias.

L'objectif de ce modèle est de permettre la diffusion synchrone de médias finis dans le temps puisque les informations temporelles de chaque média doivent être connues a priori afin d'établir la table nécessaire à leur synchronisation. De plus, les relations de type inter-médias doivent être connues à l'avance afin de pouvoir synchroniser entre eux les processus médias. Lorsque ces contraintes ne peuvent être respectées, des mécanismes de gestion de la diffusion sont utilisés afin de résoudre les problèmes empêchant le respect de ces contraintes (surcharge du système). Cette approche se limite à la diffusion de médias dont on connaît à l'avance les caractéristiques. Il se révèle donc inapproprié à la diffusion de médias capturés en temps-réel.

De plus, la diffusion de médias à l'aide de ce modèle ne permet pas la mise en place, en l'état, de traitement ou de manipulation sur ces derniers. Par exemple, les solutions d'adaptation proposées par [HAG02] (cf. § 4.1.2) ne sont pas compatibles avec ce modèle. L'intégration des médias reste donc limitée car la mise en place de certains traitements constitue une source de désynchronisation qui n'est pas prise en compte.

En revanche, l'intérêt de ce modèle réside dans sa simplicité de mise en œuvre puisqu'il est basé sur une architecture multi-processus. Il tire donc tous les avantages de ce type d'architecture : cycle de vie des processus, priorité des processus, flexibilité, etc. Le contrôle des processus est facile à mettre en œuvre par exemple pour des finalités de gestion de la QdS.

4.2.2 Modélisation des Contraintes de Synchronisation à l'aide des Réseaux de Pétri à Flux Temporels

Les travaux présentés dans [OWE03] permettent de modéliser les contraintes de synchronisation multimédia à l'aide de Réseaux de Pétri à Flux Temporels (RdPFT) [DIA93]. Cette modélisation peut être utilisée dans les AMD afin de respecter ces contraintes lors de la restitution des médias.

Les RdPFT utilisent des conditions temporelles sur les arcs sortant des places. Dans le cas des AMD, cette particularité permet d'introduire la durée de présentation des données issues des médias. Ces conditions sont exprimées sous la forme de triplets $[x^s, n^s, y^s]$ appelés intervalles de validité temporelle. Les éléments de ces triplets représentent respectivement le temps de présentation minimal, nominal et maximal.

A l'aide de cette modélisation, on peut contrôler de façon précise les dérives temporelles[31] entre les médias grâce à la définition de règles de synchronisation. Ces règles introduisent des politiques de synchronisation que l'on peut appliquer aux médias des AMD (média le plus en avance, média le plus en retard, média maître, etc.) [OWE03].

Afin de montrer les grands principes de cette modélisation, nous utilisons un scénario de synchronisation simple. L'objectif est d'assurer la synchronisation entre un flux vidéo et un flux audio dans le cadre d'une AMD de vidéoconférence. L'une des premières tâches consiste à lister les contraintes temporelles du scénario :

- un débit vidéo de 10 images par seconde ;
- une gigue maximale entre les unités issues des différents flux de 10 millisecondes ;
- le son doit jouer un rôle prépondérant sur la vidéo (priorité) ;
- la dérive temporelle entre les deux flux ne doit pas excéder 100 millisecondes.

Ces contraintes permettent de déduire les paramètres du RdPFT :

- une granularité identique sur chaque flux donc une durée de présentation nominale de 100 millisecondes ;
- la gigue de 10 millisecondes permet de définir les conditions temporelles sous la forme d'un intervalle de validité temporelle [90, 100, 110] ;
- la prépondérance du flux audio se traduit par l'utilisation d'une politique de synchronisation « et-maître » qui consiste à garantir le respect des contraintes temporelles du flux audio en essayant autant que possible de respecter celles du flux vidéo ;

31 Les dérives temporelles peuvent être calculées facilement grâce aux conditions temporelles et en particulier aux durées maximales et minimales de présentation des unités issues des médias.

- la dérive temporelle maximale autorisée permet de définir une période de synchronisation inter-flux égale à la diffusion de cinq unités sur chaque flux.

Le RdPFT de ce scénario est représenté sur la Figure 9. Il permet de montrer comment les données issues de chaque flux sont liées temporellement et traduit une vision de la synchronisation côté utilisateur.

Ce travail introduit également la spécification de scénarios de synchronisation du côté de l'environnement d'exécution à l'aide de RdPFT. Cette vision permet de prendre en compte l'environnement d'exécution des AMD. Par exemple, certaines cartes audio et vidéo introduisent des temps de latence que l'on peut prendre en considération en utilisant la notion de rendez-vous décalés [OWE96] qui permet de démarrer la diffusion des flux en avance ou en retard suivant la spécification des cartes. Une présentation plus complète de ces deux niveaux de synchronisation est disponible dans [OWI96].

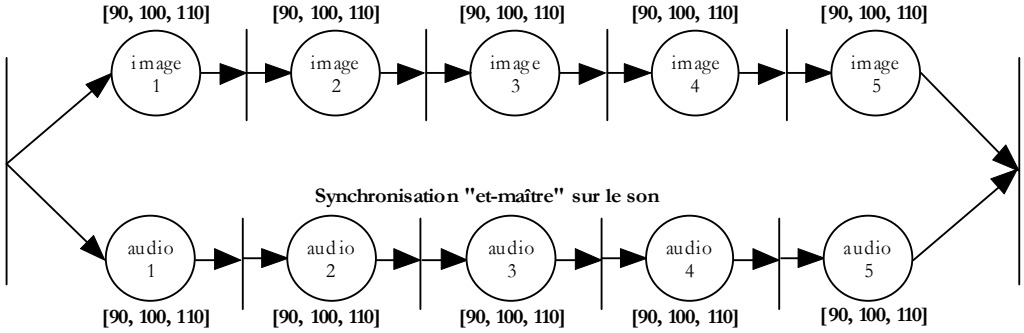


Figure 9 Scénario de Synchronisation entre un Flux Audio et un Flux Vidéo [OWE03]

Ce modèle permet la spécification de contraintes temporelles pour assurer la diffusion synchrone des médias dans les AMD. Lors des phases de développement, il est utilisé pour déterminer l'ordonnancement temporel des processus chargés de la diffusion des médias. Il permet donc de garantir les relations de synchronisation intra et inter-médias. Cette approche est complète car elle considère les points de vue de l'utilisateur et de l'environnement d'exécution. Toutefois, cette considération nécessite de connaître à l'avance l'ensemble des paramètres relatifs à ces points de vue et ayant un impact direct sur la synchronisation multimédia. Cette solution semble donc difficile à mettre en œuvre dans les AMD qui évoluent dans des environnements hétérogènes et mouvants.

Comme la précédente (cf. § 4.2.1), ce type de modélisation nécessite de définir a priori des scénarios de synchronisation entre des médias et donc de connaître leurs contraintes temporelles. Nous pensons que ce type d'approche soulève d'énormes

problèmes lorsque l'on veut considérer des flux de médias discrets dont on ne connaît rien a priori sauf sans doute le type des données. La définition d'intervalles de validité temporelle sur les unités de ces médias semble donc difficile. De plus, que se passe-t-il si certains médias subissent des traitements en amont de leur diffusion ? Les intervalles sont-ils toujours les mêmes ?

4.2.3 Synchronisation Multimédia basé sur des Relations de Causalité

Le mécanisme de synchronisation proposé par Courtiat *et al.* [COU96] est basé sur des relations de causalité avec pour objectif d'assurer les relations de synchronisation intra et inter-flux de bout en bout du réseau. Il est destiné à être implémenté comme une surcouche d'un service de transport orienté connexion avec une bande passante garantie, un taux de pertes limité et des mécanismes de compensation de la gigue. La Figure 10 décrit l'architecture d'une application utilisant un tel mécanisme.

La couche applicative associe des étiquettes temporelles aux unités d'information issues des médias. Les médias devant être transmis de manière synchrone sont groupés en paquets avant d'être délivrés à la couche synchronisation.

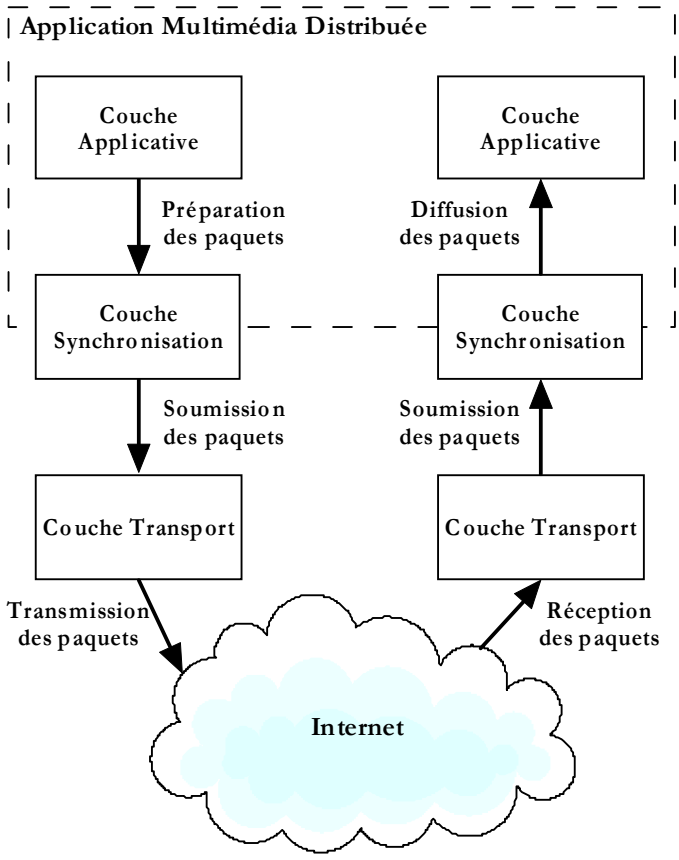


Figure 10 Architecture de l'application

Des expressions de dépendances conditionnelles sont également associées aux unités d'information issues des médias. Ces dépendances expriment des relations de causalité sous la forme d'expressions booléennes. Elles ont pour but de traduire les contraintes de diffusion d'un flux (relations de synchronisation intra-flux) ou de plusieurs flux (relations de synchronisation inter-flux). Ainsi, $Dexpr^m$ représente l'expression de la dépendance conditionnelle associée à l'unité m. Prenons l'exemple d'un flux S qui possède les contraintes intra-flux suivantes : $S = t_1^{n1}, t_2^{n2}, {}^{n2}t_3^{n3}, \ldots$ où les t_i^{ni} représentent les instants de diffusion des unités ni du flux S. Alors, $Dexpr^{n1}=Dexpr^{n2}=$vrai et $Dexpr^{n3}=n2$ sont les expressions de dépendance conditionnelle associées aux unités n1, n2 et n3. Elles permettent d'exprimer que n1 et n2 ne sont soumises à aucune contrainte de diffusion et que la diffusion de n3 dépend de celle de n2. Cette dernière dépendance implique que si n2 n'est pas délivrée alors n3 ne le sera pas non plus.

Les contraintes de type inter-flux sont exprimées selon le même principe. Prenons maintenant l'exemple d'un élément B composé de deux flux et affichant les relations suivantes :

$$B = \begin{vmatrix} t_1^{n1}, t_3^{n2}, {}^{m2}t_5^{n3}, \ldots \\ t_2^{m1}, {}^{n3}t_4^{m2}, \ldots \end{vmatrix}.$$

Alors, $Dexpr^{n3}=m2$ et $Dexpr^{m2}=n3$ sont les expressions de dépendance associées aux unités n3 et m2. Elles expriment que les unités n3 et m2 doivent être diffusées en même temps. Ces dépendances expriment des points de synchronisation entre les unités issues des flux. Plusieurs types de points de synchronisation sont prévus par le modèle, nous ne les détaillons pas mais le lecteur intéressé pourra les trouver dans [COU96].

Côté récepteur, ces dépendances sont utilisées par la couche applicative pour la diffusion des unités (cf. Figure 10) en synchronisme. Cette diffusion dépend également de contraintes temporelles que l'on peut définir suivant la nature des flux concernés. Ainsi, on appelle intervalle de diffusion, un intervalle associé à certaines unités défini de la manière suivante :

$$\Delta m = \left[td_{\min}^m, td_{\max}^m \right].$$

Il permet de définir le temps maximal $\delta d = td_{\max}^m - td_{\min}^m$ pendant lequel une unité m peut être stockée côté récepteur avant qu'elle ne soit délivrée à la couche applicative. Suivant le type du flux et l'importance de ses contraintes temporelles, le temps δd sera défini différemment. Si aucun mécanisme de compensation de la gigue n'est appliqué au flux, alors les contraintes temporelles du flux ne sont pas critiques et δd est défini comme un paramètre de la couche synchronisation. Si ce n'est pas le cas, δd est déduit

de la gigue résiduelle autorisée par le mécanisme. Ceci implique que si une unité arrive trop tard, c'est à dire après $td^{m}_{\max}$, elle ne sera pas délivrée à la couche applicative de même que toutes les unités qui lui sont liées par des dépendances conditionnelles. Un algorithme est utilisé afin d'évaluer, pour chaque unité, son expression de dépendance conditionnelle et le respect de l'intervalle de diffusion temporelle. Selon les résultats, l'algorithme prend la décision de rejeter ou non l'unité concernée ainsi que celles qui lui sont liées.

Cette approche de synchronisation base les aspects temporels des flux sur des relations de causalité entre les unités qui les composent. L'objectif est de délivrer des flux synchrones à la couche applicative. Grâce aux points de synchronisation, il est possible d'étendre les relations de synchronisation inter-flux. On peut par exemple différer la diffusion de plusieurs flux tout en assurant pour chacun leur relation de synchronisation intra-flux. Ce modèle permet donc de spécifier des relations de synchronisation multimédia que l'on peut qualifier de synthétiques (cf. § 2.2). Par exemple, on peut créer des relations temporelles comme celles définies par [ALL83] et [HAM72].

Ces travaux mettent en œuvre des politiques de livraison des données à la couche applicative. Si une unité est évincée par ces politiques ou perdue, aucune des unités qui lui sont liées ne sera délivrée. Nous trouvons cette méthode problématique si beaucoup d'unités sont liées entre elles. En cas de problème, on risque de perdre un trop grand nombre d'unités. Dans les AMD où la durée des médias est longue, comme par exemple dans des vidéoconférences ou des présentations à distance, ces politiques s'avèrent dangereuses. A la lecture des exemples d'applications donnés dans [COU96], cette approche semble appropriée pour des AMD hautement interactives où la durée des médias transmis est relativement courte.

Comme dans les travaux précédents (cf. § 4.2.1 et 4.2.2), il est nécessaire de définir des scénarios de synchronisation au préalable, ce qui nécessite de connaître les contraintes temporelles des médias utilisés. Ainsi, l'approche présentée permettra d'assurer ces scénarios.

Le mécanisme proposé assure de transmettre les médias concernés en synchronisme de bout en bout du réseau. Il est destiné à être implémenté en tant que surcouche d'un service de transport orienté connexion avec des garanties de bande passante, un taux de pertes limité et en fournissant des mécanismes de compensation de la gigue. Cette approche n'est donc pas adaptée aux réseaux comme l'Internet car ils ne fournissent pas ces services. De plus, nous pensons que dans les AMD, en raison des traitements que certains flux peuvent subir, le transport synchrone doit être étendu à l'ensemble de l'application.

Nous retenons de ces travaux une idée intéressante qui consiste à rassembler les unités des flux possédant des relations de synchronisation inter-flux en paquets. De cette manière, on définit explicitement ce type de relation de synchronisation.

4.3 Gestion de la Qualité de Service dans les AMD

Avant de refermer ce chapitre, nous présentons des travaux relatifs à la gestion de la QdS dans les AMD. Les recherches retenues utilisent des techniques de gestion par adaptation de l'application.

4.3.1 La plate-forme Polka

La plate-forme POLKA (PrOcessus Légers et KAlité de service) a été développée par l'ENST (Ecole Nationale Supérieure des Télécommunications) Paris [SIN99]. C'est une plate-forme qui supporte l'exécution d'AMD qui manipulent des flux continus contraints temporellement c'est à dire intégrant des relations de synchronisation intra- et inter-flux. Ces travaux fournissent un modèle de spécification des AMD à l'aide de graphes de flots de données permettant de décrire une AMD et son comportement temporel ainsi que les composants matériels importants du système sous-jacent. La plate-forme utilise ces spécifications pour exécuter les AMD dans la limite des ressources disponibles.

Les AMD sont implémentées à l'aide d'objets coopérants (composants). Les interactions entre ces objets sont réalisées par appels de méthodes. Un flux continu correspond à une suite d'invocations de méthodes qui se décomposent en événements. Des équations et inéquations de QdS sont utilisées pour spécifier les contraintes temporelles des flux. Ces contraintes étant amenées à évoluer pendant l'exécution, les équations et inéquations sont modifiables dynamiquement par l'utilisateur. Elles sont exprimées à l'aide d'une logique temporelle nommée QL (QoS Language) [STE93]. Par exemple, l'inéquation suivante permet de spécifier des relations de synchronisation de type inter-flux : $\forall$ n : $\varepsilon_1 \leq \tau(e, n+k)-\tau(e', n) \leq \varepsilon_2$ où e et e' sont des événements et $\tau(x, n)$ un opérateur qui fournit la date de l'occurrence n de l'événement x. Ainsi, cette inéquation permet de stipuler que la $n^{ème}$ occurrence de e' doit être séparée d'au moins ε_1 et d'au plus ε_2 unité de temps de l'occurrence n+k de e. De la même façon, ces inéquations peuvent servir à spécifier des relations de synchronisation intra-flux.

Les nœuds des graphes de flots de données représentent les composants matériels et logiciels tandis que les arcs représentent les flux continus. Un flux d'horloge permet de modéliser le temps physique. La QdS est spécifiée selon le point de vue de l'utilisateur et celui de l'AMD. En effet, l'utilisateur spécifie les contraintes de QdS

qu'il veut que l'AMD respecte comme par exemple la synchronisation multimédia. Des équations et inéquations de QdS en entrée du graphe permettent de spécifier les contraintes requises par l'AMD afin de satisfaire celles de l'utilisateur.

Les composants sont les briques de base de l'AMD, ils sont assemblés en parallèle ou en séquence. Le comportement temporel de chacun est défini par un contrat de QdS. Un tel contrat est composé de deux jeux d'équations : une pour la QdS offerte et l'autre pour la QdS requise. La plate-forme utilise ces contrats ainsi que la composition de composants et la QdS utilisateur pour déduire la QdS requise. Cette dernière constitue une condition suffisante pour que l'AMD respecte la QdS utilisateur dans la limite des ressources disponibles. Si des ressources viennent à manquer, il est du ressort de l'AMD d'adapter son comportement de façon à restreindre la QdS requise. Ceci peut s'opérer de façon dynamique en modifiant les équations et inéquations. Cependant, ce modèle n'exclut pas la possibilité d'utiliser des services de réservation de ressources.

Ces travaux proposent une méthode intéressante pour la spécification des AMD à l'aide d'un modèle basé sur des graphes de flots de données qui permettent de décrire les composants logiciels et matériels ainsi que les flux continus qu'ils échangent. La gestion de la QdS s'opère par adaptation de l'application en fonction de contraintes exprimées par les utilisateurs, des ressources disponibles et des contrats de QdS fournis par chaque composant d'une AMD. Les contraintes temporelles des flux continus sont également prises en compte et participent à cette gestion. Ces dernières sont considérées durant tout le transit des flux continus à l'intérieur d'une AMD.

4.3.2 Mécanismes de Gestion de la Qualité de Service

L'approche présentée dans ce paragraphe introduit un langage de spécification des AMD et les politiques de reconfiguration [LAY04]. Les AMD visées sont celles qui utilisent des techniques de streaming de données multimédias.

Cette approche se divise en trois niveaux distincts. Un niveau de spécification définit la structure d'une AMD et ses politiques de reconfiguration. Une telle AMD se compose d'un assemblage de composants logiciels. Les reconfigurations sont de deux niveaux, soit structurelles, soit par paramétrage des composants et par contrôle de leur cycle de vie. Dans ce but, un langage de spécification basé sur XML (eXtended Markup Language) est utilisé, il se nomme APSL (AdaPtive media Streaming Language). Un niveau de contrôle permet de traduire les spécifications APSL au niveau exécution. Il se comporte comme une interface entre les spécifications des AMD et leur déploiement. Enfin, le niveau exécution s'intéresse aux aspects d'implémentation de l'AMD.

Le langage APSL est basé sur un modèle de graphe des tâches. Chaque tâche T_i correspond à une fonction atomique d'une AMD. Ces tâches sont reliées entre elles par un graphe acyclique. Une tâche T_i consomme des flux en entrée et produit des flux en sortie. APSL permet également la spécification de politiques de reconfiguration exprimées sous la forme de « probes » (sondes), conditions et actions. Un « probe » est composé d'un ou plusieurs événements qui définissent les paramètres observés et les valeurs de ces deniers susceptibles de déclencher des reconfigurations. Une condition permet d'associer des actions de reconfiguration (actuators) à ces événements. Ces actions correspondent à des opérations à mener sur la structure de l'AMD. Elles peuvent agir à trois niveaux :

- sur les paramètres fonctionnels des composants : qualité du codage, fréquence, résolution, etc. ;
- sur la structure du graphe en modifiant les liens entre les composants ;
- sur les politiques de reconfiguration en les validant, les invalidant ou en changeant leur comportement.

Les spécifications définies par le langage APSL sont compilées en deux phases distinctes :

- la première phase consiste à vérifier la syntaxe et la sémantique du graphe des tâches ;
- la seconde phase permet de transformer le graphe des tâches afin de le rendre implémentable par instanciation, configuration et connexion des composants (travail du gestionnaire de configuration).

Un gestionnaire de reconfiguration est chargé de la création des composants nécessaires aux opérations de reconfiguration afin, par exemple de pouvoir remplacer un composant par un autre fonctionnellement identique. Ce gestionnaire est également responsable de la création de composants nommés probes et actuators. Un probe est responsable de la détection des événements déclencheurs des reconfigurations. Les probes ressources sont chargés de surveiller les ressources de l'environnement d'exécution. Les probes QdS interagissent avec les composants et retournent des événements relatifs à la QdS. Les événements détectés par les probes sont récupérés par les actuators qui sont chargés d'exécuter les actions de reconfiguration adéquates.

Un point fort de ces travaux est qu'ils fournissent des processus de transformation qui permettent d'obtenir à partir des spécifications une implémentation possible pour une AMD.

Le langage APSL permet de décrire l'architecture d'une AMD à l'aide de graphes de tâches acycliques. Les travaux présentés dans le paragraphe précédent (cf. § 4.3.1) utilisent le même type de modélisation qui semble être une approche intéressante pour la description de l'architecture des AMD. En effet, les structures représentées par des graphes sont facilement manipulables ce qui constitue une qualité certaine pour mener des phases de configuration/reconfiguration sur la structure d'une application.

Cette approche utilise les composants logiciels pour implémenter les AMD et en introduit différents types. Les composants que l'on peut qualifier de fonctionnels sont destinés à implémenter les fonctionnalités qui composent une AMD. D'autres composants sont utilisés pour la gestion de la QdS et ne sont donc pas directement en rapport avec les fonctionnalités des AMD, on peut les qualifier de non-fonctionnels. Cette façon de faire est une volonté des auteurs de séparer les préoccupations. L'avantage de ce type de démarche est de promouvoir la réutilisation des composants car les différents aspects des AMD sont gérés indépendamment. Nous verrons plus tard que cette séparation des préoccupations possède des avantages dans l'implémentation des applications actuelles.

Ces travaux diffèrent de précédemment (cf. § 4.3.1) dans le sens où la gestion de la QdS est laissée à des composants spécialisés qui permettent de surveiller l'application mais aussi de provoquer ses reconfigurations conformément à des politiques pouvant évoluer durant l'exécution. Le principe de reconfiguration est d'adapter la structure des AMD aux besoins émergeants. Ce type de reconfiguration s'avère très puissant.

5 Synthèse

Ce chapitre donne un aperçu du monde des AMD avec pour ambition de dégager les points critiques et essentiels qu'il est nécessaire d'adresser lorsque l'on s'intéresse au développement de ce type d'applications. Bien entendu, ces considérations sont importantes pour les AMD que nous visons (cf. définition dans le § 1.1).

Nous avons pris conscience dans ce chapitre, et ce dés l'introduction (cf. § 1), de l'importance des données manipulées. En effet, le média est un concept central, toutes les définitions et classifications que nous avons donné s'accordent sur ce point. La particularité de ces données est qu'elles peuvent exister sous différentes formes (médias continus et discrets), ce qui complexifie leur manipulation dans les applications. Les médias continus sont des données particulières qui affichent des propriétés et des contraintes d'ordre temporel qui n'existent pas explicitement dans les médias

discrets ou dans les données plus « traditionnelles ». Ces propriétés doivent être prises en compte car ces médias sont basés sur des propriétés psychosensorielles humaines. La compréhension des médias par les utilisateurs dépend donc de ces caractéristiques. Les études présentées sur la perception humaine [GHI98], [STE96], [WEI98] viennent appuyer cette argumentation. Les contraintes temporelles se traduisent en relations de synchronisation entre les données appartenant à un même média mais aussi entre celles appartenant à différents médias continus ou discrets.

La synchronisation multimédia est un paramètre important qu'il est nécessaire de conserver. Malgré les différents types de médias, une AMD doit être capable de traiter, transmettre et diffuser ces médias ensemble (notion d'intégration des médias dans la classification de [BLA96]). Nous avons identifié deux sources de désynchronisation des médias dans les AMD. Non seulement ces sources sont nuisibles à la synchronisation mais aussi, dans certains cas, aux autres propriétés des médias. La première peut survenir lorsque l'on traite des médias liés par des relations de synchronisation à d'autres non traités (cf. Figure 6). En effet, si le traitement ne détruit pas le débit du média (dans le cas d'un média continu), il introduit généralement un retard de sorte qu'un groupe de médias initialement synchrones se trouvera désynchronisé dés lors que l'un des médias aura fait l'objet d'un traitement. Toute manipulation, à moins qu'elle ne soit capable de porter sur l'ensemble du groupe de médias, détruit donc les relations temporelles qui lient ces médias. La seconde source de désynchronisation identifiée provient de l'aspect distribué de ces applications. En effet, les transferts de médias à l'aide de réseaux de communication détruisent les relations de synchronisation et sont même susceptibles d'altérer la séquence propre aux médias continus. Ces désagréments sont causés par les services rendus par les protocoles de transport utilisés. De nombreuses recherches apportent des solutions en proposant des protocoles de transport adaptés aux médias de bout en bout du réseau (cf. par exemple RTP [SCH96] et les travaux de Courtiat *et al.* [COU96]). Ce problème peut également être résolu en utilisant des formats de codage et de compression particuliers basés sur des techniques de multiplexage des médias (cf. par exemple [HEM99] et [RAN96]). Ces approches permettent de figer les relations de synchronisation pendant le transport des médias. Néanmoins, ces solutions contraignent à l'utilisation de formats particuliers. De plus, elles ne permettent pas une intégration aisée de tous les types de média. Ces solutions constituent donc un frein à l'intégration des médias. Elles rendent également les traitements des médias plus complexes à gérer au sein d'une AMD. Nous trouvons l'idée intéressante mais nous pensons qu'il faut opter pour une solution moins contraignante (cf. [TEN90]). Nous pensons que le transport synchrone des données doit être étendu de bout en bout d'une AMD afin d'éviter les sources de synchronisation dont nous avons parlé. Nous retenons donc les idées et les mécanis-

mes introduits par les protocoles de transport synchrone : étiquette temporelle, numéro de séquence, etc.

En raison de leurs spécificités et caractéristiques et d'une utilisation toujours croissante du réseau Internet [JOU05], [OBS06], [RIV03], le développement des AMD a considérablement augmenté. Ce réseau fournit un environnement hétérogène tant au niveau logiciel qu'au niveau matériel, ce qui rend son fonctionnement non prévisible. Dans ce sens, les travaux basés sur la QdS tentent de donner une réponse en essayant de garantir l'offre d'un service minimum. Des travaux antérieurs abordent ce domaine [LAP06].

Afin d'illustrer ces différents thèmes, nous avons proposé dans la dernière partie de ce chapitre une étude de travaux issus du domaine des AMD. Cette étude s'est articulée autour de trois types de travaux : ceux qui ont trait à la définition d'AMD particulières, ceux qui se sont fixé comme problématique d'assurer la synchronisation multimédia et enfin ceux qui mettent en place des systèmes de gestion de la QdS. Nous proposons en guise de synthèse une comparaison de ces travaux par rapport aux quatre principaux points abordés, à savoir les médias, la synchronisation, l'aspect réseau et la gestion de la QdS. Afin de mieux situer nos travaux nous les avons comparés aux autres selon ces quatre axes. Ainsi, nous espérons montrer l'apport de notre recherche. Le résultat de cette synthèse est donné par la Table 1.

Table 1 Comparaison des Différents Travaux

Travaux	Type d'AMD	Médias		Synchronisation			Réseau		QdS	
		Type	Intégration	Type	Couche	Spécification	Type	Protocole	Point de Vue	Type de Gestion
Vosaic	*Navigateur Internet*	Continus/ Discrets	Forte	Pas de Synchronisation			Internet	TCP/VDP/RTP	Environnement	Adaptation des médias
Serveur Vidéo	*Vidéo à la Demande*	Continus	Faible	Pas de Synchronisation			Internet	RTP	Environnement/ Utilisateur	Adaptation des médias
MultiSync	*Lecteur Multimédia*	Continus/ Discrets	Forte	intra et inter	application	a priori	Non distribué		Environnement	Adaptation des médias
RdPFT	*Vidéoconférence*	Continus	Faible	intra et inter	application/ matériel	a priori	Internet	POC	Environnement/ Utilisateur	Pas de Gestion
Relations de Causalité	*Applications Interactives*	Continus/ Discrets	Faible	intra et inter	réseau	a priori	Services Garantis	non précisé	Environnement	Réservation de Ressources
Polka	*Audio/Vidéo*	Continus/ Discrets	Forte	intra et inter	application	a priori	Services Garantis	non précisé	Environnement/ Utilisateur	Adaptation de l'application
Gestion de la QdS dans les AMD	*Streaming*	Continus	Forte	Pas de Synchronisation			Internet	RTP	Environnement	Adaptation de l'application
Notre Approche	*Systèmes Numériques*	Continus/ Discrets	Forte	intra et inter	application/réseau	pendant l'exécution	Internet	TCP/UDP	Environnement/ Utilisateur	Adaptation de l'application

La Table 1 montre que la gestion de la QdS est un point important dans les AMD. Même si elle n'est pas explicitement nommée dans ces divers travaux, nous pouvons constater qu'elle est présente dans la majorité d'entre eux. Cette constatation

conforte donc nos choix. La synchronisation n'est pas prise en compte par tous les travaux. Lorsqu'elle l'est, on remarque que les contraintes temporelles des médias doivent être connues a priori. Cette connaissance nous semble difficiles à obtenir dans les cas où les médias sont créés en temps réel par l'AMD. Il est en effet difficile de prévoir toutes les contraintes surtout si l'on considère que les traitements subis par certains médias peuvent modifier ces contraintes temporelles. Dans ce sens, nous croyons qu'une synchronisation assurée pendant l'exécution en considérant les contraintes telles qu'elles existent lors de la création des médias est préférable. De même que leur modification doit être possible en cours d'exécution. Nous pouvons également remarquer que l'Internet est un réseau très prisé pour le déploiement des AMD. Certains travaux utilisent des protocoles de transport classiques. D'autres s'axent davantage sur les aspects réseau en définissant des protocoles spécifiques. Nous n'avons pas retenu cette approche. Enfin, du fait que nous nous intéressons aux systèmes numériques [BLA96], nous proposons une intégration forte des différents types de média comme dans certains travaux présentés.

Maintenant que nous avons décrit les principes et les exigences des AMD, nous allons nous intéresser aux moyens nécessaires pour leur mise en œuvre. En d'autres termes, nous devons définir une architecture logicielle adaptée à ce type d'application du moins telles que nous les percevons. Nos travaux s'orientent autour de la gestion de la QdS dans ces applications [LAP06]. Ces dernières étant distribuées sur le réseau Internet, nous avons choisi une gestion par adaptation de l'application, ce qui nous impose de disposer d'une architecture flexible. Pour ce faire, nous pensons qu'il est nécessaire d'utiliser une approche modulaire afin de définir une AMD à l'aide d'entités de granularité plus faible et donc plus facilement manipulables. Ainsi, les développeurs ne créent plus d'applications monolithiques mais créent et emploient des briques logicielles réutilisables.

Nous choisissons, comme approche modulaire, le paradigme des composants logiciels. L'intérêt est de définir des architectures plus facilement manipulables à tous les niveaux de leur cycle de vie. Ainsi, dans un souci de gestion de la QdS une AMD peut facilement être mise à jour par configuration/reconfiguration de sa structure en fonction des paramètres collectés. Ceci pouvant être réalisé par ajout, suppression, remplacement ou déplacement de composants. Une autre facette intéressante de ce paradigme est qu'il permet de séparer les différents aspects d'une application. On peut donc implémenter sur des éléments architecturaux séparés les différentes activités d'une AMD. Ce type de démarche permet de promouvoir la réutilisation car ces derniers peuvent être définis de manière indépendante et pourtant fonctionner ensemble.

Nous avons pu remarquer dans les travaux présentés que ce paradigme ou du moins les idées qu'il propose sont utilisées dans beaucoup de travaux de recherches. En conséquence, nous pensons qu'il est nécessaire d'offrir dans cette approche des mécanismes autonomes permettant l'utilisation de composants de traitement des médias qui tiennent compte des caractéristiques particulières de ces données. De plus, nous proposons que leur transport soit réalisé à l'aide d'un dispositif unique non seulement entre les composants d'une AMD mais aussi au travers d'un réseau comme l'Internet.

Afin d'avoir un aperçu des possibilités offertes par le paradigme des composants logiciels, le prochain chapitre présente les principes de ce domaine.

Chapitre 2 – Les Composants Logiciels

« The most important characteristic of a software components industry is that it will offer families of routines for any given job. […] In other words, the purchaser of a component from a family will choose one tailored to his exact needs. […] He will expect the routine to be intelligible, doubtless expressed in a higher level language appropriate to the purpose of the component, thought not necessarily instantly compilable in any processor he has for his machine. He will expect families of routines to be constructed on rational principles so that families fit together as building blocks. In short, he should be able safety to regard components as black boxes. »

Mass-Produced Software Components, Douglas Mc Ilroy, octobre 1968

L'un des enjeux actuels du développement de logiciels est la recherche d'abstractions proposant des entités pour manipuler plus facilement et plus sûrement les programmes et les données. Le génie logiciel est une discipline de l'informatique qui traite de ces aspects en fournissant des langages et des méthodes de conception et de développement d'applications. Les méthodes sont influencées par les structures que sont susceptibles de supporter les langages de programmation au niveau organisationnel. Ces méthodes sont sensées couvrir tous les aspects du cycle de vie des logiciels depuis l'analyse des besoins jusqu'à la maintenance du produit. Le paradigme des composants logiciels est l'une de ces méthodes, elle a pour enjeu la conception et le développement d'applications par assemblage d'entités logicielles appelées composants. Comme on peut le remarquer dans l'article de Mc Ilroy [ILR68], l'assemblage de composants est une question relativement ancienne. Le principe est de décomposer une application en fonctionnalités, de concevoir et de développer ces fonctionnalités puis de « rassembler » le tout afin de disposer du produit désiré. La difficulté de ces méthodes ne réside pas dans l'implémentation des fonctionnalités d'une application mais plutôt dans leur assemblage pour proposer des architectures flexibles dans lesquelles des parties ou sous-parties peuvent être réutilisables dans la même ou dans d'autres applications. Cette approche puise ses concepts dans les systèmes matériels qui sont réalisés par un assemblage de composants électriques ou électroniques où chacun est chargé de la réalisation d'une tâche particulière considérée comme atomique par rapport à la fonctionnalité d'ensemble du système.

Un composant est une entité logicielle spécifiée et développée pour réaliser une fonction particulière d'une application. Le défit de ce paradigme réside dans la capacité de définir les informations et moyens nécessaires pour qu'un composant puisse être

assemblé avec un ou plusieurs autres mais aussi dans la réalisation de mécanismes permettant un tel assemblage. Ces moyens dépendent fortement des impératifs définis par le domaine d'application, aussi appelé métier. Nous voulons dire par là qu'une application appartient à un domaine donné (cf. par exemple les AMD présentées dans le chapitre 1) et que, généralement, les caractéristiques et propriétés de ce domaine imposent le respect de certaines contraintes et règles. Il est clair qu'un composant qui effectue des transactions sur une base de données ne se dotera pas des mêmes moyens qu'un composant chargé de traiter les données issues d'un capteur. Ainsi un modèle de composants logiciels est spécifié et développé pour un domaine particulier et se doit de tenir compte de ses particularités.

Un composant est donc vu comme une entité logicielle indépendante, déployable et pouvant être assemblée avec d'autres [SZY02]. Cette approche suscite un intérêt toujours croissant chez les acteurs de l'industrie du logiciel. Ainsi, de nombreux modèles sont issus des milieux académiques et industriels. Toutefois, le paradigme des composants logiciels ne fait pas, pour l'instant, l'objet d'un consensus. Nous montrons dans ce chapitre la vision que nous avons de ce domaine et en quoi cette approche peut répondre à nos besoins dans la définition d'une architecture logicielle.

1 Introduction

Depuis plusieurs années, l'industrie du logiciel se trouve confrontée à de nombreuses mutations. La diversité des systèmes d'exploitation et des périphériques disponibles sur le marché ont rendu le développement des applications de plus en plus long et complexe. L'émergence des réseaux de communication implique fréquemment que les applications soient basées sur des systèmes distribués. En effet, l'utilisation toujours croissante de réseaux à grande échelle comme l'Internet (voir à ce sujet [JOU05], [OBS06], [RIV03]) a favorisé l'émergence de nouvelles architectures logicielles appelées multi-niveaux ou n-tiers[32]. Ce type d'architecture induit une hétérogénéité logicielle et matérielle qu'il est difficile mais important de considérer.

Les méthodes et techniques de conception et de développement des applications sont confrontées à une complexité croissante des architectures logicielles, ce qui a pour effet négatif d'augmenter les coûts de conception et de développement. De plus, ces changements du paysage informatique offrent des possibilités telles que les besoins

[32] Les architectures de type multi-niveaux ou n-tiers sont des extensions du modèle client/serveur qui consistent à déporter les logiques métier d'une application vers des niveaux intermédiaires afin de répondre à des besoins de facilité de mise à jour et de maintenance, d'augmentation des performances et de la sécurité [LAI03].

et les attentes des clients évoluent rapidement et souvent plus que les temps de conception et de développement. Ceci pose évidemment un problème de réactivité auquel doit faire face l'industrie du logiciel. Naturellement, elle s'intéresse de plus en plus aux notions de flexibilité et de réutilisabilité. Il y a quelques années, la programmation orientée objet a été introduite pour tenter de répondre à ces attentes. Plusieurs concepts ont été définis mais semblent insuffisants pour les développements actuels [FIN01]. En effet, la programmation orientée objet a été introduite afin de promouvoir la réutilisabilité par la conception d'objets suffisamment générique pour remplir cette tâche. Il semble que ce paradigme est passé à côté de cet objectif. Un autre exemple est celui de la prise en compte des propriétés non-fonctionnelles[33] à laquelle se heurte cette approche. Ces propriétés sont directement liées aux caractéristiques d'implémentation logicielle et matérielle. En fait, ce paradigme ne tient pas suffisamment compte de ces aspects concrets, ce qui constitue un frein vis à vis des capacités d'intégration et de réutilisation dans les environnements actuels.

Les acteurs du génie logiciel se sont orientés vers d'autres solutions. Les nombreux travaux sur les composants ont pour objectif de trouver des réponses aux questions laissées en suspend par la programmation orientée objet. Ainsi, l'une des volontés de l'industrie est de capitaliser du code afin de fournir du logiciel sur étagère (cf. l'approche des COTS[34] [JAM04], [LIU03], [VIG96]) avec pour objectif l'accroissement de la productivité et la réduction des temps de conception et de développement. Ce paradigme se base sur des techniques de spécification et de définition d'applications qui intègrent de façon homogène des entités logicielles – appelées composants – ainsi que leurs interactions – appelées connecteurs. L'objectif étant de fournir un ensemble de modèles permettant de supporter et d'intégrer ces différents aspects. Ces modèles décrivent de façon exhaustive les entités, règles et mécanismes qui vont permettre la construction d'applications par assemblage de composants. Tout ceci devant être défini et supporté tout au long du cycle de vie des applications. Nous abordons dans ce chapitre les concepts clés de l'approche des composants logiciels.

2 Les Composants Logiciels

Un composant est une entité logicielle de déploiement indépendante qui permet la modélisation et l'implémentation d'applications par assemblage. Un composant doit

[33] Les propriétés non-fonctionnelles concernent les aspects du développement d'une application qui ne relèvent pas de sa logique métier.

[34] Commercial Off The Shelf.

donc, dans un premier temps, faire l'objet d'une définition complète au niveau de la modélisation afin de spécifier les caractéristiques, propriétés et mécanismes qu'il devra proposer pour avoir, lors de l'implémentation, le comportement souhaité. Le premier point de cette section s'attache à définir le concept de composants logiciels à travers ses grands principes.

Un paramètre important et prometteur de ce type de paradigme est celui de l'assemblage de composants. En effet, si la programmation orientée objet enfouissait ce code dans les objets constituant l'application, la programmation orientée composant tente, quant à elle, de le définir au même niveau que les composants en proposant l'utilisation d'entités ou de mécanismes spécialisés. L'objectif étant bien entendu de proposer des architectures logicielles plus flexibles basées sur une réelle réutilisation des entités qui les composent. Le second point de cette partie introduit la notion de connecteur en présentant quelques moyens d'interaction classiques en guise d'illustration.

Un autre aspect ignoré par la programmation orientée objet est celui de la séparation des préoccupations de conception et d'implémentation. Certains paradigmes ont vu le jour en se proposant d'étudier ces notions qui s'avèrent importantes lorsque l'on veut promouvoir la réutilisation, c'est le cas par exemple de la programmation orientée aspects[35] [BOU01], [DUC02]. On distingue habituellement les aspects fonctionnels et non-fonctionnels. L'approche par composants logiciels fait un effort considérable dans ce sens. Ainsi, des services non-fonctionnels sont proposés aux développeurs de façon à ce qu'ils puissent concentrer leurs efforts sur les aspects fonctionnels (également appelés métier) des applications. La mise à disposition de tels services ainsi que leur possible extension est un point important des modèles de composants. Le dernier point de ce paragraphe donne un aperçu de ces propriétés en décrivant quelques solutions utilisées pour leur gestion.

2.1 Qu'est-ce qu'un Composant Logiciel ?

Le terme de composant logiciel couvre un large domaine, il en est pour preuve le nombre de définitions et d'approches que l'on peut trouver à ce sujet dans la littérature. En effet, il n'existe pas à l'heure actuelle de consensus sur sa définition et sur sa vision à long terme.

[35] Aspect Oriented Programming (AOP) en anglais.

2.1.1 Définitions

Plusieurs définitions sont intéressantes et exposent des points de vue différents que nous étudions ici. La définition du terme « composant » que l'on peut trouver dans le dictionnaire [LAR04] souligne l'aspect de composition de composants, et ce quelle que soit leur nature :

> *« Qui entre dans la composition de quelque chose (adjectif). Constituant élémentaire d'une machine, d'un appareil ou d'un circuit électrique ou électronique (nom masculin). »*

Un composant est donc une entité élémentaire, atomique qui entre dans la composition d'entités de plus haut niveau : machine, appareil, circuit, application informatique, etc. Un autre point intéressant de cette définition est qu'elle permet de définir le terme composant dans le domaine du logiciel comme un constituant élémentaire, une unité de composition utilisée pour la conception et le développement d'applications. Certaines définitions du domaine confirment cette vision des choses, c'est le cas par exemple de celle donnée par Sametinger [SAM97] :

> *« Software components are defined as prefabricated, pretested, self-contained, reusable software modules […] that perform specific function. »*

En plus de définir cette notion d'unité de composition, cette définition insiste sur la propriété de réutilisabilité des composants puisqu'elle précise leur vocation à être utilisés dans plusieurs applications différentes (*« prefabricated, pretested, reusable »*). Cette définition reste toutefois très générale car elle peut s'appliquer à tous les types de composants (par exemple les composants électroniques).

Il est nécessaire de préciser les moyens qui vont permettre à un composant de rentrer dans une composition (application, circuit électronique, et pourquoi pas un autre composant) en assurant, bien entendu, un fonctionnement correct. Pour ce faire, un composant doit pouvoir interagir avec son environnement. On retient à ce titre deux définitions :

> *« A component is a special kind of object that communicates with other components in a structured way. » [ALD02]*
>
> *« A software component is a static abstraction with plugs. » [NIE95]*

Ces interactions (*« with other components »*) doivent se réaliser au travers de mécanismes clairement définis (*« in a structured way »*) afin de fixer les règles de communication entre les composants qui définissent un composé. La seconde définition introduit le terme *« plugs »* qui se traduit par le mot « prises » en français, ce qui sous-entend la nécessité de disposer d'un moyen de connexion pour faire interagir plusieurs composants.

Le paradigme des composants logiciels suppose la spécification et la définition de façon claire et précise des propriétés, caractéristiques, mécanismes et règles que devront posséder les composants pour la conception et l'implémentation d'applications. Ces abstractions doivent donc faire l'objet d'un modèle dont le rôle est d'expliciter les différentes façons d'utiliser les composants pour cette tâche. Cette notion de modèle est très présente en informatique, et plus particulièrement dans le domaine du génie logiciel, car elle permet de donner un cadre à la conception. La définition de Heineman *et al.* [HEI01] argumente notre point de vue :

> *« A software component is a software element that conforms to a component model and can be independently deployed and composed without modification according to a composition standard. […] A component model defines specific interaction and composition standard. A component model implementation is the dedicated set of executable software elements required to support the execution of components that conform to the model. »*

Cette définition est très intéressante car elle éclaire cette notion de modèle, toujours un peu obscure dans ce paradigme. Elle la sépare en deux niveaux hiérarchiques distincts. Le premier fait référence au modèle de conception qui permet de définir les modes d'interaction et de composition standard des composants. Tandis que le second niveau fait référence au modèle d'implémentation qui représente les implémentations possibles des composants tels qu'ils sont définis au niveau conceptuel. Le concept d'interactions standards spécifie les moyens dont doivent se doter les composants pour pouvoir être assemblés. La notion de composition définit les règles d'assemblage des composants en se basant directement sur les moyens d'interactions standards fournis par les composants. Ces deux aspects sont donc fortement liés. La spécification et la définition de l'ensemble de ces aspects doivent donc être rassemblées au sein de modèles [SZY02].

Cette étude nous conduit à proposer la définition suivante du concept de composant logiciel :

> *« Un composant logiciel est une unité de composition d'applications conçue et développée pou réaliser une fonctionnalité particulière capable de communiquer et d'interagir avec son environnement. Les moyens et les règles utilisés pour arriver à cet objectif sont définis par un modèle de composants. »*

La suite du paragraphe précise les caractéristiques principales des composants.

2.1.2 Propriétés des Composants Logiciels

Quand on parle de composant, il faut garder à l'esprit qu'il se décline en tant que composant au niveau modèle mais aussi en tant qu'instance au niveau implémen-

tation. Sa première déclinaison décrit le composant en tant qu'entité abstraite de modélisation. La seconde traduit le composant comme une implémentation chargée de fournir une fonctionnalité particulière au même titre qu'une instance d'objet, c'est-à-dire une entité concrète qui s'exécute au sein d'un système. Une application est spécifiée et conçue à l'aide de composants en tant qu'entité abstraite puis implémentée à l'aide de composants en tant qu'entité d'implémentation. Cette notion de composant est donc utilisée à travers différents niveaux du cycle de vie des logiciels depuis la phase de conception, jusqu'aux phases d'implémentation puis d'exécution. L'étude et le développement de modèles consistent à définir les composants logiciels à ces différents niveaux.

Chacune des deux approches précédentes possède ses propriétés et caractéristiques propres. Ainsi, celles définies au niveau abstrait précisent les règles d'utilisation et les mécanismes d'un composant de façon générique. Ceci fait abstraction totale des plates-formes cibles utilisées pour l'implémentation. Les propriétés et caractéristiques définies au niveau de l'implémentation permettent de disposer de plusieurs comportements pour un même composant par paramétrage de ce dernier. On peut de la sorte adapter dynamiquement l'exécution d'un composant en fonction des contextes applicatifs.

Ces propriétés et caractéristiques se répartissent en deux parties distinctes : les propriétés fonctionnelles et les propriétés non-fonctionnelles. Les propriétés fonctionnelles d'un composant correspondent à sa mise en œuvre d'un point de vue métier c'est-à-dire qu'elles sont directement liées à sa fonctionnalité. La plupart du temps, elles sont définies au niveau de l'implémentation. Les propriétés non-fonctionnelles permettent en général la définition de mécanismes pour la mise en œuvre du composant comme une unité de composition capable d'interagir avec son environnement. Elles représentent les aspects qui permettent à un composant d'avoir le fonctionnement voulu. Les propriétés non-fonctionnelles sont également présentes au niveau implémentation. Elles permettent dans ce cas l'adaptation des propriétés fonctionnelles aux conditions d'exécution et aux implémentations induites par la modélisation [DUC02]. Par exemple, la synchronisation, la récupération des erreurs et la persistance sont des propriétés non-fonctionnelles. Cette distinction des propriétés d'un composant est empruntée à la séparation des préoccupations [LOP95] qui se fixe comme objectif de réduire les complexités de conception, de réalisation et de maintenance en abordant les applications par parties puis en séparant les préoccupations fonctionnelles des préoccupations non-fonctionnelles. Ces idées sont également reprises par la programmation par aspects [BOU01], [DUC02].

2.2 Composition des Composants Logiciels

Les composants logiciels sont des unités de composition. Cette caractéristique est habituellement abordée selon deux points de vue que l'on nomme respectivement la composition verticale et la composition horizontale. La composition verticale, également appelée composition hiérarchique [MAG95], permet de définir des composants de granularité supérieure par composition de composants de granularité inférieure. La composition horizontale [ROM06] permet de définir des assemblages de composants par interactions de ces derniers afin de déployer des applications. Nous présentons dans ce paragraphe ces deux types de composition en insistant sur la composition horizontale à cause de son caractère indispensable car elle permet de définir des assemblages donc des applications.

2.2.1 La Composition Verticale

Cet aspect consiste à voir l'implémentation des composants selon différents niveaux hiérarchiques. Le but est de définir des composants de granularité supérieure en utilisant des composants de granularité inférieure pour les réaliser. Ainsi, on distingue ces niveaux en définissant les composants primitifs et les composants composites. Un composant est dit primitif lorsqu'il implémente une fonctionnalité de granularité faible, c'est le cas par exemple des composants définis par les modèles EJB (Enterprise Java Beans) [SUN03] et .NET [BOB01]. Ces modèles sont appelés modèles plats. Par opposition, un composant est dit composite s'il est lui-même écrit à l'aide de composants primitifs et/ou composites, c'est le cas par exemple des modèles Darwin [MAG95] et Fractal [BRU03]. Cette approche permet donc de définir des composants de granularité différente pouvant réaliser des fonctionnalités à différents niveaux hiérarchiques d'une application. Cependant, un composite doit posséder les mêmes propriétés et caractéristiques qu'un primitif si l'on veut pouvoir les considérer comme des composants et donc pouvoir les faire interagir avec d'autres.

Cet aspect de la composition n'est pas indispensable dans les modèles de composants. Cependant, il est pratique pour la configuration et la reconfiguration de systèmes à base de composants puisqu'il permet de manipuler les composites de la même manière que les primitifs.

2.2.2 La Composition Horizontale

La composition horizontale, quant à elle, est capitale pour les composants logiciels dés lors que l'on veut fournir la possibilité de développer des applications à l'aide

d'assemblages de composants. Elle fournit une vue compositionnelle des applications derrière laquelle se cache la notion de connexion.

Cette fonction est assurée par différents moyens, nous nous efforçons d'en présenter quelques uns. Nous abordons dans un premier temps la définition d'interactions simples entre composants à l'aide d'outils introduits par la programmation orientée objet : les interfaces et les événements. Puis, nous introduisons dans un second temps une entité utilisée dans certains modèles de composants ainsi que dans les langages de description d'architectures[36] [ACC02], [MED00], [SHA95]. Cette entité est communément appelée connecteur. Elle permet de définir des interactions d'ordre plus complexes entre des composants logiciels.

2.2.2.1 Des interactions simples…

Les interactions définies dans cette section sont réalisées à l'aide d'outils empruntés à la programmation orientée objets. Ces outils sont connus sous le nom d'interfaces et d'événements [VIL03].

2.2.2.1.1 Interfaces et Evènements

Deux types de connexions simples sont utilisés dans le paradigme composant :

- les interfaces fournies/requises ;
- les sources/puits d'événements.

Le concept d'interface [MER00] décrit un ensemble de signatures d'opérations groupées sous un même type mais ne contenant aucune implémentation de celles-ci[37]. Le paradigme des composants logiciels exploite ce principe. L'interface d'un composant est définie comme une spécification de ses points d'accès [SZY02]. Ainsi, un client accède aux services fournis par un composant en utilisant ces points d'accès [CRN02]. Le paradigme des composants logiciels ajoute des notions supplémentaires qui permettent de spécifier les connexions. Ainsi, on distingue les interfaces fournies et les interfaces requises. Lorsqu'elle est fournie, une interface décrit l'ensemble des services fournis par un composant ou par une partie de celui-ci. Lorsqu'elle est requise, une interface décrit l'ensemble des services auxquels un composant ou une partie de celui-ci doit accéder pour fonctionner correctement. L'un des avantages est de détacher les services fournis et requis de leurs implémentations, ce qui laisse la possi-

---

[36] Architecture Description Language (ADL) en anglais.

[37] En analyse et conception orientée objet, une méthode correspond à l'implémentation d'une opération.

bilité de remplacer les implémentations sans changer les interfaces et donc de favoriser les performances du système sans pour autant le reconstruire. Il est également possible d'ajouter de nouvelles interfaces et par là même de nouvelles implémentations sans changer les implémentations existantes. Cette possibilité permet de favoriser l'adaptabilité des composants mais aussi leur réutilisabilité.

Le concept d'événement est également emprunté à la programmation orientée objet [MER00]. En principe, un événement est associé à une implémentation exécutée lorsqu'un événement particulier se produit sur un système. Ce principe est également exploité par les composants au travers des concepts de source et de puits. Une source représente un événement pouvant être déclenché par un composant. Un puits est un événement requis par un composant pour assurer son fonctionnement, c'est à dire qu'il attend l'arrivée d'un événement particulier afin d'exécuter l'opération associée. Le mécanisme d'événements est intéressant car il permet d'associer des implémentations à des occurrences significatives localisées dans le temps et dans l'espace.

Chacun de ces moyens assure un mode de communication particulier aux composants. Ainsi, les interfaces proposent un mode de communication synchrone qui n'est autre que de l'invocation de méthodes tandis que les sources et les puits d'événements proposent un mode asynchrone par émission et réception d'événements. L'avantage de ces approches est de faciliter la substitution d'un composant par un autre. La Table 2 résume ces interactions.

Table 2 Différents Moyens d'Interactions

Interactions entre Composants		
Type	**Principe**	**Mode de Communication**
Interface fournie/requise	Invocation de méthodes	Mode Synchrone
Source/Puits	Emission et Réception d'événements	Mode Asynchrone

Ces moyens permettent de réaliser les connexions entre les composants et leur environnement.

2.2.2.1.2 Les Connexions

Les connexions entre les composants se réalisent à l'aide des moyens décrits précédemment entre dispositifs de même nature. Certaines recherches émergeantes s'intéressent cependant à faire interagir des composants qui utilisent des modes de communication différents [LOU07]. Ainsi, deux composants qui communiquent par

interfaces sont connectés en liant l'interface requise de l'un à l'interface fournie de l'autre. De la même manière, deux composants peuvent être assemblés en connectant la source de l'un au puits de l'autre. Des cardinalités peuvent être associées à ces connexions afin par exemple de connecter une source à plusieurs puits. Nous illustrons ces connexions sur la Figure 11 en présentant un exemple de chaque type de connexion modélisé à l'aide du langage UML (Unified Modeling Language) [OMG03], [PIL06]. Un CompteBancaire fournit une interface Identite qui apporte les services nécessaires pour connaître l'identité d'un compte. Une OperationBancaire nécessite de connaître l'identité d'un compte afin de procéder à une telle opération sur le compte d'un client. La Figure 11 donne un exemple de connexion à l'aide de sources et de puits. La notation utilisée pour modéliser cette connexion n'est pas à notre connaissance une notation UML, elle est empruntée au modèle de composants CORBA [MAR99]. Une FenetreGraphique lève un événement ClicSouris lorsqu'un clic est effectué sur cette dernière. Le composant GestionnaireFenetre est à l'écoute de cet événement. Lorsqu'un tel événement se produit, ce composant exécute l'implémentation associée à ce puits qui peut par exemple consister à modifier l'aspect graphique de la fenêtre.

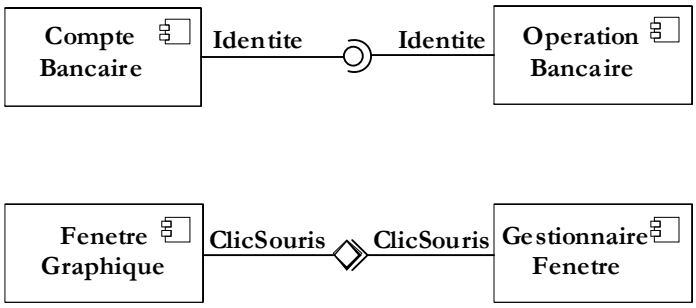


Figure 11 Exemples de Connexions entre Composants

Lorsqu'une connexion est établie par interface, les méthodes fournies par un composant peuvent être invoquées par un autre qui les nécessite pour son exécution. La connexion entre les composants est ainsi établie. Les connexions établies par des sources et des puits reposent sur le même principe. Certains travaux proposent l'étude de mécanismes de vérification syntaxique des connexions, c'est le cas, par exemple, de l'environnement ArchJava qui propose un mécanisme de vérification de type exécuté à chaque fois que de telles connexions entre des composants sont établies [ALD02].

2.2.2.1.3 Les Contrats

La sémantique de ce type de connexions peut être précisée à l'aide de contrats [CRN02]. Ils permettent de lister les contraintes auxquelles doivent répondre les implémentations des connexions afin de pouvoir être assurées. En principe, ces contrain-

tes consistent en la spécification des opérations associées à l'aide de préconditions et de postconditions. De la sorte, on spécifie les conditions d'implémentation, les règles de connexion et les changements subis par les objets du modèle.

Les préconditions sont des conditions et des propriétés sur les objets et les entités du modèle. Plus précisément, elles permettent de décrire l'état dans lequel devra se trouver le système pour que la connexion puisse être établie. Les postconditions décrivent, quant à elles, les modifications de l'état des objets et des entités du modèle ou les effets de l'exécution d'une opération lorsque la connexion vient d'être établie. Ces modifications peuvent par exemple concerner des créations d'instances, des formations et des ruptures d'association mais aussi des modifications d'attributs.

D'autres niveaux peuvent également être pris en charge par les contrats [BEU99] :

- le niveau syntaxique décrit les propriétés de compatibilité en vérifiant la conformité des participants à une interaction ;
- le niveau synchronisation permet de spécifier le comportement global d'une connexion en terme de synchronisation entre les appels de méthode ;
- le niveau de QdS donne des informations sur les propriétés mesurables souhaitées et/ou offertes par une opération.

Le contrat est un moyen intéressant d'analyse des besoins qui expose en détail les changements d'état d'un ensemble d'objets ou d'entités sans qu'il soit nécessaire de décrire comment ces changements doivent être obtenus. En d'autres termes, les contrats permettent de différer l'implémentation, et on peut se concentrer sur l'analyse de ce qui doit se produire sans se soucier de la manière dont cela se produit. De plus, les contrats sont applicables aux opérations à tous les niveaux de granularité : les opérations d'un composant mais aussi les opérations des objets qui rentrent dans la constitution d'un composant.

2.2.2.2 … vers des interactions plus complexes

Les types d'interaction abordés précédemment consistent à faire coopérer simplement des composants par invocation de méthodes ou sur déclenchement d'événements. Des liens plus complexes peuvent être mis en œuvre et quelquefois faire appel à une entité supplémentaire nommée connecteur [GAR94]. Les connecteurs sont des abstractions de communication de même niveau que l'abstraction composant, ce sont des éléments supplémentaires de l'architecture logicielle qui servent d'intermédiaires entre les composants [GAR01]. De façon générale, un connecteur définit l'ensemble des messages que des composants échangent, la façon dont ils se synchronisent (cf.

par exemple le modèle producteur/consommateur) et les contraintes qu'impose l'interaction. Cette entité encapsule donc un protocole d'interaction entre composants. Au niveau implémentation, il est vu comme une entité exécutable à part entière. Le concept de connecteur est donc similaire à celui de composant sauf qu'il concerne des préoccupations différentes. La distinction entre les rôles représentés par composants et connecteurs permet de promouvoir la réutilisabilité. Les protocoles sont réifiés à travers des entités réutilisables indépendantes des composants. La définition d'une telle entité insiste sur la mise en avant d'une vue compositionnelle des applications dans laquelle les interactions entre les composants sont réalisées par une entité concrète.

Un connecteur peut, par exemple, permettre l'accès à une variable partagée, un échange de données sous la forme de flux[38], la mise en place d'une communication bâtie selon le modèle client-serveur, un lien avec une base de données, etc. Afin de décrire ce genre d'interactions, le connecteur doit se doter des mécanismes d'interaction utilisés par les composants (cf. paragraphe précédent) mais aussi permettre d'assurer l'interopérabilité de l'ensemble composants-connecteurs [MED00].

Un connecteur est associé à une implémentation dont le rôle est de mettre en œuvre le protocole utilisé. Il ne correspond pas obligatoirement à une seule unité d'implémentation. En effet, dans le cas d'une communication par réseau, le connecteur est composé de la partie cliente, de la partie serveur et du réseau utilisé. Certains auteurs ont travaillé sur la classification des connecteurs selon les catégories de services rendus. Par exemple, [MEH00] distingue quatre catégories :

- les services de communication qui supportent la transmission de données entre des composants ;
- les services de coordination qui permettent de transférer le contrôle de l'exécution des composants à travers une application c'est à dire de contrôler le déclenchement de l'exécution de chaque composant de l'architecture ;
- les services de conversion qui permettent de convertir une interaction fournie par un composant en une interaction requise par un autre. Ce type de service permet de faire interagir des composants hétérogènes (la conversion de données est un exemple de cette classe de service) ;

---

38 Ce mode de communication des données est rencontré quelquefois dans les modèles de composants logiciels, c'est un mode de diffusion en continu d'informations entre les composants.

- les services de médiation[39] qui permettent de faciliter et d'optimiser l'interaction entre plusieurs composants, les mécanismes de répartition de charge, d'ordonnancement ou de contrôle de la concurrence en sont des exemples.

Les connecteurs fournissent des services pouvant appartenir à une ou plusieurs de ces catégories afin de fournir des interactions plus riches. Il est, par exemple, tout à fait possible de définir un connecteur qui fournit un service de communication et un service de coordination [MEH00].

Dans les modèles qui définissent des connecteurs, une application est composée par interconnexion de composants et de connecteurs.

2.3 Les Types de Composants Logiciels

Dans les modèles de composants, il est possible de définir plusieurs types de composants en tant qu'abstraction. Ainsi, un type de composant représente une définition abstraite d'une entité logicielle. Il se caractérise par des modalités de connexion et par des propriétés communes voulues pour un composant. Un composant étant une unité indépendante, un type de composant est défini sans connaissance des types de composants auxquels il pourra être connecté. Cette information sera explicitée lors de la définition de la composition des composants. Un composant au niveau implémentation n'est autre qu'une instance d'un type de composant.

Le modèle EJB définit trois types de composants, chacun étant spécifié pour répondre à des besoins particuliers de conception et d'implémentation. Ainsi, les sessions s'exécutent pour le compte d'un client, les entités sont liées à la notion de persistance et les message-driven permettent le traitement de messages asynchrones [SUN03].

2.4 Les Conteneurs de Composants

Les conteneurs de composants sont des entités introduites afin de prendre en charge les propriétés non-fonctionnelles des composants. Ils doivent faciliter la gestion et la réutilisation de ces dernières. Le rôle du conteneur est de proposer des services d'ordre non-fonctionnels aux composants, il peut être vu comme un environne-

39 Dans l'article [MEH00], ce terme est désigné par le mot anglais facilitation et défini de la manière suivante : *« Facilitation connectors mediate and streamline component interaction ».*

ment d'exécution pour un ou plusieurs composants autorisant l'accès à ces services. Dans les architectures à base de composants, ces services sont réutilisés par réutilisation des conteneurs. Cette méthode fournit une séparation claire entre les aspects des composants. Elle représente une amélioration intéressante de la programmation orientée objet.

Ce concept n'est pas présent dans tous les modèles de composants. Les modèles EJB [SUN03] et CCM (Corba Component Model) [MAR99] proposent des conteneurs. Le modèle Fractal [BRU03] introduit le concept de contrôleur qui est une partie intégrante du composant chargée de gérer des propriétés non-fonctionnelles. Ainsi, ces entités proposent, par exemple, des services d'administration, de persistance, de transaction, de sécurité, etc. L'avantage de cette approche est que ces services, bien souvent complexes à mettre en œuvre, peuvent être gérés de façon relativement transparente pour les concepteurs. Cependant, l'inconvénient de certains modèles dans l'utilisation des conteneurs réside dans le fait qu'ils définissent un nombre figé de services qu'il n'est pas possible d'étendre, c'est le cas par exemple des modèles EJB et CCM [DUC02]. Cette dernière remarque est importante à considérer dés lors que les modèles évoluent et se dirigent vers de nouveaux domaines. Les conteneurs doivent donc pouvoir évoluer en termes de services proposés afin de rendre la programmation des composants indépendante de celles des propriétés non-fonctionnelles. Les composants doivent pouvoir utiliser les services augmentés ou les nouveaux services sans modification de leur comportement.

3 Modèle de Composants Logiciels

Un aspect important dans la conception d'une application est son architecture. Le développement d'applications revient alors à définir les architectures en termes de composants et d'interactions entre eux. Un modèle de composants permet de rassembler et de définir les composants, les règles d'utilisation, les comportements et les moyens de connexion ainsi que tous les dispositifs nécessaires à la conception et au déploiement d'applications. Cet ensemble va permettre de constituer une application comme un assemblage de composants. Les modèles doivent également permettre de décrire les cycles de vie des applications.

La notion de modèle est un concept central du génie logiciel qui permet, quel que soit le paradigme considéré et le domaine visé, l'étude des propriétés abstraites indépendamment des objets réels qui pourraient satisfaire ces propriétés. L'objectif étant d'aboutir à une solution « informatique » des problèmes rencontrés dans le monde réel. Un modèle se veut réutilisable, il doit donc définir les règles et les

connaissances nécessaires pour remplir cet objectif de modélisation et de passage du monde réel vers le monde informatique. Dans cette définition, deux idées sont capitales, d'un côté la notion de propriétés abstraites et de l'autre celle d'objets réels. Nous retrouvons les mêmes idées dans les modèles de composants. Dans ce paradigme, un modèle[40] va permettre la définition à tout point de vue des entités logicielles (composants, connecteurs, conteneurs, etc.) qui le composeront selon les besoins identifiés et les applications visées. Un modèle de composants décrit les concepts à partir desquels sont définis les entités ainsi que les différents mécanismes basés sur ces derniers.

L'objectif premier d'un modèle de composants est de se placer à un niveau d'abstraction élevé afin d'introduire des entités ayant le comportement voulu sans se préoccuper des détails techniques qui sont plutôt liés à des préoccupations d'implémentation considérées comme secondaires. De la sorte, on va pouvoir définir de façon précise le rôle que va jouer chaque élément dans la conception et le développement d'un système[41]. De plus, on va rendre explicite le type d'entités utilisées et les interactions entre elles puisque ces développements sont basés sur la construction d'applications par assemblage de composants. D'un point de vue structurel, ces entités représentent une vision abstraite d'une interaction d'objets (au sens large du terme) utilisés pour les définir. Toutes les caractéristiques, propriétés et entités (interactions, modes de communication, propriétés configurables, propriétés fonctionnelles et non-fonctionnelles, etc.) que nous avons présentées jusqu'à maintenant doivent être définies à ce niveau, cet ensemble fait donc partie intégrante de la définition d'un modèle de composants.

Une fois défini et complet, un modèle permet alors la spécification et la conception d'applications à base de composants. Lorsque l'on veut passer à une phase de développement de ces applications, il est nécessaire de redescendre vers des préoccupations de bas niveau afin de pouvoir proposer une ou plusieurs implémentations d'un modèle utilisables pour le déploiement des applications. On parle alors de modèles d'exécution ou de frameworks [VIL03], ce concept correspond à l'idée d'objets réels retenue dans la tentative de définition générale de la notion de modèle. Ces modèles rassemblent les outils et langages nécessaires à l'implémentation d'entités concrètes (par opposition à leurs définitions abstraites à un niveau hiérarchique plus élevé). Les

---

40 La notion de modèle peut se définir selon une approche ascendante (bottom-up) ou descendante (top-down). Nous décrivons dans ce paragraphe le concept de modèle selon une approche descendante car nous avons retenu cette méthodologie dans notre approche puisque les applications auxquelles nous nous intéressons sont centrées utilisateur.

41 Mary Shaw et David Garlan définissent ce principe d'abstraction capital dans toutes définitions de modèle : *« It Should be possible to describe the components and their interactions of a software architecture in a way that clearly and explicitly prescribes their abstract roles in a system »* [SHA94].

modèles d'exécution ou frameworks décrivent les comportements « réels » des entités abstraites définies précédemment. Ainsi, les comportements « réels » sont décrits à l'aide des langages de programmation qui permettent de définir comment les propriétés et entités abstraites vont être mises en œuvre. Cette tâche n'est possible que par la transformation du modèle abstrait de composants vers un ensemble d'éléments de bas niveau ou d'un langage de programmation représentant le modèle au niveau de l'implémentation. Cette phase de transformation va permettre de mettre en évidence des détails techniques et autres mécanismes que la phase d'abstraction est sensée cacher ou du moins reléguer à un second plan. Ces détails et mécanismes sont des propriétés d'ordre non-fonctionnel car elles sont liées à l'implémentation du modèle abstrait. Il est nécessaire, lors de la transformation, de les prendre en considération si l'on veut aboutir à une implémentation conforme aux spécifications avancées par le modèle abstrait. Un framework ou un modèle d'exécution est constitué des entités et propriétés implémentées du modèle abstrait ainsi que d'entités non-fonctionnelles et autres outils. Ces implémentations dépendent bien évidemment des caractéristiques des langages de programmation utilisés pour ces réalisations. La Figure 12 résume ce point de vue.

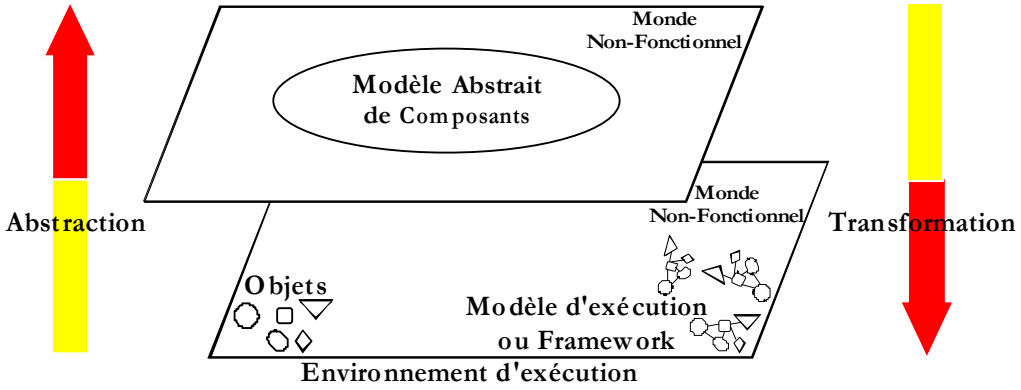


Figure 12 Concept de Modèle de Composants

4 Exemple : un Modèle de Composants Multimédias

De nombreux modèles de composants sont actuellement définis aussi bien dans les milieux académiques qu'industriels. Afin d'illustrer le concept de modèle, nous proposons de décrire un modèle de composants logiciels lié au domaine du multimédia et destiné au développement d'applications de même type. Cet exemple va permettre de faire le lien entre les deux domaines de recherche présentés dans ces deux chapitres.

Bien qu'un peu ancien, le travail présenté dans cette section [GIB94] est tout de même intéressant car il décrit les spécifications d'un modèle de composants spécialisé dans le développement d'AMD. Ce modèle s'oriente autour de quatre classes différentes :

- la classe média définit un type de média, une instance de cette classe correspond à un type de média particulier comme par exemple l'audio ou la vidéo ;
- la classe traitement représente une opération particulière applicable sur un média ;
- la classe format encapsule des informations relatives à la représentation d'un type de média c'est-à-dire son format ;
- la classe composant définit une entité logicielle ou matérielle capable de produire, consommer et traiter des médias.

Tous les types de média sont considérés par ce modèle, ils existent sous la forme de flux de données mais aussi sous la forme de données discrètes. Le modèle est développé pour une plate-forme cible. Les composants sont implémentés à l'aide des services fournis par cette plate-forme. Une application multimédia est développée par instanciation des composants et connexion de ces instances.

Trois types d'interface sont introduits afin de permettre la communication d'un composant avec son environnement[42] :

- l'interface synchrone décrit les opérations pouvant être invoquées sur un composant, elle permet le contrôle externe du composant ;
- l'interface asynchrone permet l'émission d'événements à destination de l'environnement d'un composant, elles sont utilisées pour coordonner et surveiller le comportement du composant ;
- l'interface isochrone est dédiée au transfert des médias dans un composant de façon isochrone c'est-à-dire au même rythme pour éviter tout effet de congestion.

Le modèle utilise des connecteurs afin de transporter les données entre les composants (mémoire partagée, réseau, etc.). Des dispositifs nommés ports sont associés aux composants et aux connecteurs et vont permettre aux médias de transiter entre

[42] Lorsque l'on parle d'environnement d'un composant, on fait référence aux autres composants et aux objets utilisés pour l'implémentation d'une AMD.

ces différentes entités. Les ports sont de deux types : les ports d'entrée et les ports de sortie. Les connexions entre les entités sont réalisées en connectant les ports de sortie aux ports d'entrée à condition que ces ports soient compatibles. En effet, à chaque port est associé un ensemble de formats qui sont des instances de la classe format. Ainsi, on définit les formats supportés par un port. La règle utilisée pour la compatibilité des ports énonce que pour que deux ports soient compatibles, il faut que leurs ensembles de formats aient au moins un élément commun.

Une application conçue avec un tel modèle est constituée d'un ou plusieurs groupes de composants et de connecteurs les reliant. Chaque groupe est appelé réseau de composants. L'application est responsable de la gestion statique et dynamique de ces réseaux. Elle est donc responsable de leur cycle de vie. L'application est aussi chargée de l'instanciation des composants par allocation des ressources matérielles qui leur sont nécessaires (mémoire, bande passante, processeur, périphériques). Les composants sont initialisés à l'aide de leur interface synchrone. Elle se charge de surveiller la synchronisation des médias devant l'être à l'aide des interfaces isochrones. Lorsque des désynchronisations se produisent, l'application doit rétablir la situation en s'auto-adaptant. Enfin, c'est elle qui gère le trafic des événements entre les composants et donc assure la coordination de l'ensemble.

Ce modèle est entièrement dédié au développement des AMD. Il est naturel d'y retrouver des mécanismes pour répondre aux exigences métier de ce domaine. En effet, les interfaces et les classes définies vont répondre à ces exigences et permettre le transport des données multimédias à travers ces applications. Cependant, certains services sont définis sous une forme minimale, ils doivent de fait être étendus par les développeurs afin de répondre aux besoins, c'est le cas par exemple du service de synchronisation [GIB94].

Le modèle introduit le concept de connecteur et propose différents types d'interactions entre les composants par utilisation de différents types de connecteurs. Les connexions entre les connecteurs et les composants sont rendues possibles grâce à des dispositifs nommés ports d'entrée et ports de sortie. Ils se connectent l'un avec l'autre s'ils sont compatibles, ce sont les points d'entrée des données sur les entités. Aucun mécanisme de vérification automatique de la compatibilité n'est décrit par le modèle, ce qui peut avoir pour effet de multiplier les risques d'erreur si beaucoup de données sont manipulées. Les reconfigurations de ces applications doivent donc définir des règles précises afin de pallier ce type d'inconvénient.

Un autre point intéressant réside dans le fait que les entités du modèle sont synchronisées à l'implémentation à l'aide d'événements. Ceci permet une exécution cor-

recte de l'application en faisant transiter les données de composant en composant tout en étant sûr que chacun a bien appliqué son traitement.

L'utilisation de ce modèle se restreint aux environnements où la réservation de ressources est possible (cf. chapitre 1). Cette particularité limite son champ d'action et le destine donc à des plates-formes particulières autorisant ces mécanismes.

Les données sont transmises dans les composants de façon isochrone c'est-à-dire au même rythme afin de conserver les relations de synchronisation entre elles. Ceci implique de ne pas disposer de relations temporelles trop disparates car certaines données multimédias très sensibles au temps (cf. chapitre 1) pourraient en pâtir. De plus, les données sont manipulées de façons différentes selon leur type. Les mécanismes définis dans le modèle doivent prendre en compte cette propriété, ce qui complexifie leur définition et leur implémentation.

Ce modèle permet de définir des applications auto-adaptables c'est-à-dire capables de gérer le cycle de vie des entités exécutées. Dans ce type de solution, les aspects d'adaptation et les aspects métier sont placés au même niveau bien qu'ils concernent des préoccupations différentes.

5 Synthèse

Nous avons présenté à travers ce chapitre les concepts qui nous semblent importants dans le paradigme des composants logiciels. De par les améliorations qu'il est sensé apporté par rapport à la programmation orientée objet, il devrait constituer un réel progrès pour le développement d'applications par réutilisation d'entités logicielles. Les résultats attendus sont la réduction du temps de développement des applications et une plus grande réactivité de l'industrie du logiciel face à une complexité croissante.

La modularité apportée par cette approche permet de définir les applications comme des connexions de modules indépendants. Elle s'apparente aux méthodologies d'analyse descendante qui consiste à identifier les fonctionnalités principales et secondaires des systèmes complexes. Cette approche est intéressante puisqu'elle permet d'obtenir des architectures malléables (concept défini dans [BOU00]). Elle permet de concevoir et de proposer des architectures dotées de modes de gestion de l'exécution autorisant par exemple de s'adapter aux nouvelles infrastructures informatiques et à leurs évolutions.

La séparation des aspects permet de favoriser la réutilisation en définissant des entités par type de fonctions réalisées. Ainsi, les composants implémentent les fonctionnalités des applications tandis que les connecteurs (même sous leur forme la plus

simple : événements, interfaces) implémentent les liaisons entre les composants. Cette distinction des fonctionnalités est intéressante car elle sépare les diverses implémentations tout en les mutualisant au sein d'entités spécialisées dans certaines tâches. L'objectif, lorsque l'on développe une application, est de concentrer ses efforts sur la partie fonctionnelle en utilisant les services non-fonctionnels déjà implémentés. Ainsi, les tâches de développement deviennent moins complexes.

La composition telle que nous l'avons présentée est une notion plus forte que dans le paradigme objet. En effet, couplée à la séparation des aspects, elle permet de disposer d'architectures totalement contrôlables.

Afin d'illustrer ces concepts, nous avons présenté en fin de chapitre un modèle de composants multimédias qui se destine à l'implémentation d'AMD. Il s'adapte aux particularités de ce domaine en proposant par exemple de modéliser les données soit à l'aide de flux de données, soit sous la forme de données discrètes. Pour leur transport, il définit une entité dédiée qui n'est autre qu'un connecteur. Les connexions entre les composants et les connecteurs sont réalisées par des ports d'entrée et de sortie qui sont les points d'accès des données sur les entités. Une contrainte de compatibilité, liée aux types de données manipulées est associée à ces dispositifs afin de valider la pertinence des connexions. Des interfaces et des classes spécifiques sont définies pour permettre l'implémentation des fonctionnalités, des traitements et des communications de données dans l'application. Son inconvénient majeur réside dans le fait qu'il mélange les différents aspects d'une application ce qui contraint fortement le développement des composants et des connecteurs ainsi que leur degré de réutilisation. Les propriétés non-fonctionnelles y sont succinctement abordées.

Notre étude se cantonne aux AMD avec pour objectif la gestion de la QdS de ces dernières. Nous voulons cette gestion dynamique car les exigences de QdS sont amenées à évoluer pendant l'exécution d'une application. Cette constatation nous a amenés à penser que la mise en œuvre d'une architecture logicielle flexible est nécessaire. Dans ce sens, nous avons choisi l'approche par composants logiciels car nous pensons qu'elle se prête bien à ces objectifs [DAL02]. Il en est pour preuve les travaux présentés dans [BRU02] et [BRU03].

Nous proposons un modèle qui se destine comme celui de [GIB94] à la conception et au développement des AMD. Nous allons voir dans la partie 2 de ce mémoire que la méthode de conception choisie permet de modéliser ces applications selon une approche dont la finalité est d'obtenir une application comme interconnexion de fonctionnalités atomiques. En ce sens, nous ne proposons pas un modèle hiérarchique. Seule une composition horizontale forte est définie à l'aide de connecteurs. Les interactions sont définies à l'aide d'interfaces et d'événements. En raison de l'importance

que nous attachons à la réutilisabilité et à la reconfiguration dynamique des applications, nous axons notre approche sur une séparation claire des aspects. Nous utilisons donc la notion de conteneur comme dans les modèles EJB [SUN03] et Fractal [BRU02], [BRU03]. Pour les mêmes raisons, nous définissons des composants dédiés à l'implémentation fonctionnelle et d'autres dédiés à l'implémentation non-fonctionnelle. Enfin, toutes les entités définies sont supervisables par la plate-forme d'exécution que nous utilisons comme middleware pour gérer la QdS dans ces applications. Nous reviendrons sur ce point dans le chapitre suivant.

Le paradigme des composants logiciels est donc l'outil de notre démarche de spécification et de développement d'AMD. Dans une problématique de gestion de la QdS, une approche qui apporte de la flexibilité et permet la réutilisation s'avère extrêmement intéressante. Les caractéristiques du multimédia et les problèmes inhérents (cf. chapitre 1) doivent être pris en considération lors de la définition de notre modèle afin de proposer une solution viable, efficiente et complète pour le développement des AMD.

La mise en œuvre d'un modèle de composants multimédias dans un tel contexte va nous permettre de définir des architectures logicielles malléables. Ces architectures nécessitent d'être supervisées par une plate-forme d'exécution dont le seul but est la gestion de la QdS. L'objectif de la plate-forme est d'approcher le mieux possible la QdS requise par les utilisateurs. Pour réaliser cette tâche, la plate-forme est capable d'appréhender son environnement tant du point de vue des utilisateurs que des traitements effectués sur les données multimédias et des réseaux (point de vue de l'environnement d'exécution cf. chapitre 1). Dans ce but, nous proposons une infrastructure permettant aux concepteurs de mettre en place les AMD en faisant abstraction des aspects non-fonctionnels. Nous proposons donc un modèle unique de données multimédias et des dispositifs de transport de ces données dont l'objectif est double : établir le lien entre la plate-forme et la partie opérative et distribuer les données multimédias entre les composants mais aussi à travers le réseau Internet sans perdre les relations de synchronisation qui les lient. Avant de rentrer dans le détail de ces différents modèles, nous allons décrire la plate-forme d'exécution et ses principales fonctionnalités afin que le lecteur puisse bien s'imprégner de notre méthode de gestion de la QdS dans les AMD. De plus, cette présentation va nous permettre de mettre en évidence les spécifications des différents modèles que nous introduisons dans cette thèse. En conséquence, le chapitre suivant décrit l'architecture des AMD ainsi que les principes de fonctionnement de la plate-forme en prenant comme postulat que la partie opérative est supervisable par cette dernière.

Partie 2 – Architecture et Représentation des Applications Multimédias Distribuées

Cette partie détaille l'architecture logicielle globale retenue pour le développement des AMD. Elle s'articule autour de deux chapitres. Le premier décrit l'architecture générale des AMD en donnant des détails sur les deux parties principales qui la composent. Nous décrivons une vue de l'application à laquelle les modèles proposés dans cette thèse devront correspondre. Puis, nous donnons quelques éléments sur la plate-forme d'exécution qui est une entité définie et dédiée à la gestion de la QdS. Cette gestion s'effectue par la supervision de l'application. A travers un exemple, nous allons étudier comment elle se déroule et donc définir ce qu'impose la plate-forme d'exécution sur la définition d'un modèle de composants logiciels « manipulable » par cette dernière.

Le second chapitre de cette partie introduit la méthode de conception des AMD décrite dans [LAP06]. Nous allons aborder la représentation que nous avons choisie pour décrire précisément ces applications. Cette représentation est basée sur différents graphes dont chacun décrit des niveaux d'abstraction différents. Ils permettent de définir la structure de l'application. La plate-forme d'exécution est à même de manipuler ces graphes afin de pouvoir reconfigurer la partie applicative si besoin est. Nous allons voir que ces graphes vont nous permettre de spécifier les modèles que nous présentons dans cette thèse. Ces deux chapitres vont donc nous permettre d'introduire l'approche présentée dans ce mémoire et de justifier des choix que nous avons faits pour définir une architecture logicielle pour la modélisation et l'implémentation de la partie applicative des AMD.

Chapitre 3 – Architecture des Applications Multimédias Distribuées

« As the size and complexity of software systems increases, the design problem goes beyond the algorithms and data structures of the computation: designing and specifying the overall system structure emerges as a new kind of problem. Structural issues include gross organization and global control structure; protocols for communication, synchronization, and data access; assignment of functionality to design elements; composition of design elements; scaling and performance; and selection among design alternatives. This is the software architecture level of design. »

Experience with a Course on Architecture for Software Systems, David Garlan et al., 1992.

Nos travaux s'intéressent à la façon de concevoir, de déployer et d'exploiter des AMD à travers le réseau Internet. Ce réseau apporte une grande hétérogénéité aussi bien logicielle que matérielle et fournit un service de type « best-effort ». Le déploiement et l'utilisation d'AMD sur un tel réseau est compromis dans certains cas en raison de son caractère non-prédictitble. En effet, nous avons vu que ces applications manipulent des données sensibles à la QdS fournie par leur environnement d'exécution. De plus, l'une des particularités des AMD est la perception du service rendu qu'ont les utilisateurs. Dans ce sens, nous proposons d'étudier et de définir une solution afin de rendre leur exécution possible malgré le contexte et en respectant les qualités requises par les utilisateurs. Notre démarche s'oriente autour de deux axes principaux qui consistent à superviser des applications avec une finalité de gestion de la QdS et à définir une architecture logicielle permettant l'implémentation des AMD reconfigurables dynamiquement afin d'adapter leur fonctionnement aux critères de QdS retenus. Cette thèse s'attache à définir ce dernier point.

Nous pensons dans un premier temps qu'il est utile de rappeler les principes définis par le premier axe de nos travaux. En conséquence, nous commençons dans ce chapitre par présenter l'architecture que nous avons définie précédemment pour les AMD [LAP06].

1 Introduction

L'utilisation et la manipulation des données multimédias appelées plus communément médias sur l'Internet sont de plus en plus fréquentes. Il en est pour preuve le nombre toujours croissant d'applications ou services qui manipulent des médias. Les AMD auxquelles nous nous intéressons manipulent ce type de données capturées en temps-réel ou stockées afin, par exemple, de mettre en œuvre des situations de communications distantes ou des systèmes de surveillance. Dans ce genre de systèmes, les utilisateurs ont une place centrale puisque les applicatifs mis en œuvre proposent un rendu audiovisuel qui leur est directement destiné. Ils sont donc seuls juges de la qualité de ces rendus. La perception qu'ils ont du service est alors très importante et il est indispensable de la prendre en compte si l'on souhaite susciter un réel intérêt de leur part dans l'utilisation de ces applications.

Le service « best-effort » fourni par l'Internet n'est pas adapté à l'utilisation des médias, et plus particulièrement des médias continus. Ses caractéristiques font de l'Internet un réseau fortement hétérogène tant au niveau logiciel qu'au niveau matériel, ce qui constitue un contexte d'exécution imprévisible pour les AMD. Dans ce sens, nous pensons qu'il est indispensable de prendre en compte l'environnement d'exécution des applications afin de définir des modalités d'exécution en fonction de celui-ci.

Ces deux points de vue de QdS constituent le leitmotiv de notre approche. L'objectif est de permettre aux AMD un fonctionnement satisfaisant en les faisant s'adapter à ces divers paramètres afin de fournir, en toutes circonstances, le service pour lequel elles ont été conçues. Ainsi, nous voulons gérer la QdS des AMD en considérant les exigences des utilisateurs et les possibilités de l'environnement d'exécution. L'objectif final est de s'approcher le plus possible des souhaits des utilisateurs. Cependant, ces paramètres sont amenés à évoluer pendant le fonctionnement des AMD. En effet tant les besoins des utilisateurs que le contexte de l'environnement d'exécution constituent des critères fortement mouvants dans le temps. Les AMD doivent être capables d'évaluer ces évolutions et par conséquent de déduire des politiques d'adaptation pour forcer leur exécution à en tenir compte. Nous pensons donc que la gestion de la QdS dans les AMD doit être menée de façon dynamique. C'est pourquoi elles doivent être reconfigurables pendant leur exécution afin d'intégrer les nouvelles exigences.

L'architecture logicielle que nous présentons dans ce chapitre est conçue en tenant compte de ces spécificités. Elle est détaillée plus longuement dans les documents [DAL02] et [LAP06]. Un exemple vient clore ce chapitre afin de montrer le genre d'adaptations que l'on peut mener dans des cas critiques.

2 Architecture des Applications Multimédias Distribuées

L'architecture proposée est capable d'adapter le fonctionnement des AMD aux exigences des utilisateurs en considérant les capacités de l'environnement d'exécution. Cette adaptation est dynamique afin de considérer les évolutions possibles de ces paramètres. Cet objectif ne peut être atteint que si l'on dispose d'une architecture logicielle malléable que l'on puisse modeler aux exigences de QdS.

L'architecture globale d'une AMD se divise en deux parties décrites sur la Figure 13 [DAL02], [LAP06] :

- une partie applicative implémente les fonctionnalités des AMD, l'architecture logicielle que nous définissons dans cette thèse répond à ces préoccupations ;
- une plate-forme d'exécution dont le but est de superviser et de gérer l'exécution de la partie applicative.

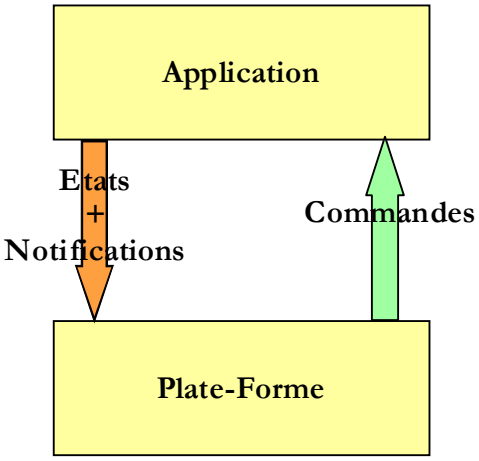


Figure 13 Architecture Globale des Applications Multimédias Distribuées

Les entités qui composent la partie applicative sont supervisées et manipulées par la plate-forme. Elles fournissent des états et des notifications sur leur fonctionnement. Ainsi, la plate-forme centralise ces informations et, en fonction des informations de QdS sur les utilisateurs et sur l'environnement d'exécution qu'elle connaît, elle est capable d'exécuter des commandes sur les entités de la partie applicative afin de la reconfigurer dynamiquement si besoin est. Cette architecture est donc réflexive et modifiable. La réflexivité est incluse dans la plate-forme c'est-à-dire que c'est elle, et elle seule, qui a la connaissance de l'architecture de la partie applicative et qui peut la modifier. Le fait que nous nous intéressions à des applications distribuées implique que la plate-forme et la partie applicative soient distribuées sur les sites où l'AMD est déployée.

Notre tâche consiste à définir des briques logicielles pour la partie applicative qui soient dynamiquement manipulables, modifiables, éventuellement paramétrables et sensibles à leur environnement. Afin de définir ces briques, nous choisissons comme solution l'approche des composants logiciels qui semble être intéressante pour ce mode de fonctionnement. Elle constitue un moyen simple d'évolution de la structure d'une application par création, suppression, connexion et déconnexion de composants.

2.1 La Partie Applicative

Nous commençons par donner la structure générale de la partie applicative. Elle sera, bien entendu, abordée plus en détail dans la partie 3 de ce mémoire.

La question qui se pose, lorsque l'on veut disposer de systèmes adaptables et flexibles, est de savoir où l'on va placer les points de modification dans l'architecture logicielle. Ces points définissent les différents niveaux sur lesquels on peut intervenir pour modifier la structure d'une application. Nous structurons donc la partie applicative en différents niveaux hiérarchiques représentés sur la Figure 14 sous la forme d'un diagramme UML (Unified Modeling Language). Chaque niveau de cette architecture représente un point de modification sur lequel on peut intervenir pour adapter la partie applicative. Chacun de ces points correspond à une sémantique précise. Ce découpage permet de modifier la structure de la partie applicative au niveau service (groupe), au niveau fonctionnalité d'un service (sous-groupe) ainsi qu'au niveau rôle atomique d'une fonctionnalité (composant et flux de données). Selon le type d'adaptation nécessaire à un moment donné, la plate-forme pourra agir sur chacun de ces niveaux que nous détaillons par la suite.

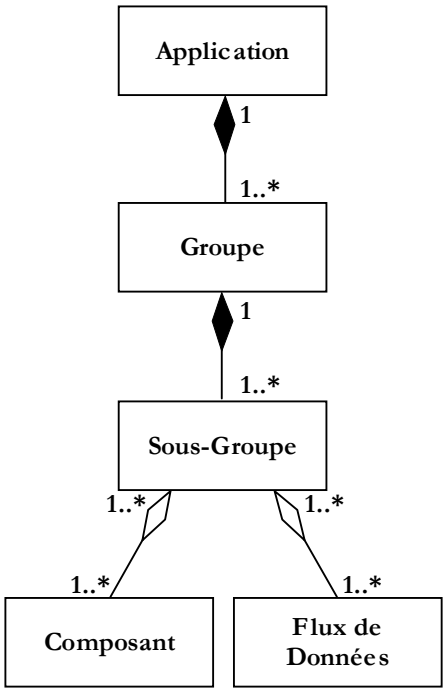


Figure 14 Modèle Structurel de la Partie Applicative [LAP06]

2.1.1 Les Groupes

Un groupe est défini comme un service fourni par l'application. Il représente un service proposé à un utilisateur. Chacun possède donc un groupe qui lui est attaché et qui représente le service que l'application lui fournit. Ainsi, dans une AMD donnée, chaque utilisateur bénéficiera d'une instance du groupe d'utilisateur auquel il appartient. Par exemple, un utilisateur qui effectue une présentation à distance se verra attribuer un groupe locuteur disposant des composants logiciels et matériels nécessaires pour capturer et transmettre son discours. Un utilisateur peut également faire partie de plusieurs groupes à la fois. Il peut être locuteur et téléspectateur dans une réunion de travail à distance.

En introduisant cette abstraction, nous rendons possible la gestion des utilisateurs dans les AMD et plus généralement la gestion des services fournis. Ainsi, un utilisateur peut joindre ou quitter une AMD quand bon lui semble, ce qui se traduit par l'instanciation ou la suppression d'un groupe avec pour conséquence l'ajout ou la suppression d'un service. De plus, aussi bien les vœux qu'expriment les utilisateurs que les évaluations des services qu'ils font, se situent à ce niveau hiérarchique.

2.1.2 Les Sous-Groupes

Un sous-groupe représente une fonctionnalité observable attachée à un service, c'est-à-dire à un groupe. Un groupe est donc composé de sous-groupes. Selon les fonctionnalités requises par les utilisateurs (paramètres de QdS), les compositions des groupes sont amenées à évoluer pendant l'exécution des AMD. De la sorte, on peut exprimer différents niveaux de qualité pour un service donné.

On considère que toutes les fonctionnalités d'un service sont observables par les utilisateurs des AMD afin qu'ils puissent les évaluer. Par exemple, la fonctionnalité son d'une vidéo peut être obtenue soit par une bande son, soit par des sous-titres. Ainsi, l'utilisateur est à même de déterminer ce qu'il préfère puisqu'il peut directement observer le résultat du choix de l'une ou l'autre de ces deux fonctionnalités. Leur existence permet une gestion de la QdS car, même si l'utilisateur préfère une bande son, le choix de sous-titres peut offrir une amélioration sensible lorsque le son est de mauvaise qualité (environnement de prise de son perturbé ou transfert réseau difficile) ou lorsqu'il est inadapté (langue inconnue de l'utilisateur).

Ce niveau hiérarchique facilite la gestion des niveaux de QdS par la gestion des fonctionnalités des services. Ainsi, la qualité d'un service peut être augmentée ou dégradée par ajout, retrait ou remplacement de fonctionnalités.

2.1.3 Les Composants et les Flux de Données

Un composant est défini comme une entité matérielle ou logicielle chargée de fournir un rôle atomique pour un sous-groupe. Un rôle atomique représente la fonctionnalité d'un sous-groupe avec la granularité la plus faible que l'on puisse obtenir lorsque l'on applique une modélisation des AMD selon une approche descendante. Les composants sont définis par leur rôle et par la QdS qu'ils fournissent. La QdS d'un sous-groupe sera donc déterminée par le choix des composants qui le constituent à un instant donné. Ainsi, on peut agir sur la composition des sous-groupes en choisissant les composants adéquats afin d'adapter le niveau de QdS à fournir. De sorte que, par exemple, l'adjonction d'un composant de transformation d'une vidéo couleur en noir et blanc peut correspondre à une perte intrinsèque de qualité mais constituer une bonne réponse à un problème de bande passante insuffisante. Ainsi son ajout apporte au final une amélioration de la QdS perçue par l'utilisateur pour peu que celui-ci ait exprimé qu'il préférait sacrifier la qualité de l'image à sa fluidité.

Les composants constituent la partie opératoire et concrète, de ce fait ils sont susceptibles d'apporter à la plate-forme des informations sur les conditions

d'exécution des AMD à partir desquelles la plate-forme pourra décider de la nécessité d'éventuelles adaptations de l'architecture et choisir la façon de les réaliser.

Les flux de données sont des structures qui permettent de transporter uniformément les médias dans les applications. Nous aborderons dans la prochaine partie du mémoire les raisons d'un tel choix. Dans une AMD, les rôles atomiques[43] échangent des médias afin de procéder à leur traitement. Les flux de données permettent donc d'établir la connexion entre les composants des AMD afin de réaliser une fonctionnalité particulière (sous-groupe).

Les flux de données participent également à la QdS des AMD. Nous verrons que le modèle que nous proposons tient compte des relations de synchronisation des flux. Ils constituent un outil prépondérant de mesure de l'environnement d'exécution puisqu'ils sont à même de détecter les aléas de fonctionnement du réseau (engorgements, bande passante, gigue, etc.) et ceux des composants (composant sous ou sur dimensionné, machine surchargée, etc.). La plate-forme les utilisera, par conséquent, comme sondes de la partie applicative.

Les sous-groupes et les groupes seront constitués de composants connectés entre eux par des flux de données.

2.2 La Plate-Forme d'exécution

L'objectif de nos travaux est de fournir une QdS acceptable pour les utilisateurs des AMD et ce, quelles que soient les capacités de l'environnement d'exécution. Un tel objectif nécessite d'être capable de contrôler l'exécution des AMD pour les adapter aux contraintes de QdS. On retrouve plusieurs solutions dans la littérature qui définissent des applications adaptables [CAM92], [DIO95], [HAG02], [KON00], [LAY04], [MAO01], [MUN99], [OCC03], [SEG02], [SIN99]. L'utilisation de plates-formes d'exécution ou d'intergiciels (middlewares) constitue une approche intéressante pour résoudre ce type de problème. Nous utilisons une solution de ce type, la définition de telles entités correspond à une volonté de séparation des aspects fonctionnels et non-fonctionnels.

2.2.1 Principe retenu

Nous proposons d'utiliser une plate-forme qui sert de support à l'exécution des AMD et dont le but est d'optimiser la QdS. Le modèle et la définition complète de la

[43] Les rôles atomiques représentent les fonctionnalités de base d'une AMD. Ils sont définis dans le chapitre suivant.

plate-forme sont donnés dans [LAP03], [LAP06]. La plate-forme est en fait une entité de supervision distribuée sur les différents sites où une AMD est déployée. Elle reçoit les états des entités qui composent la partie applicative et est capable d'exécuter des commandes sur ces dernières (cf. Figure 13). Les commandes permettent de modifier dynamiquement la structure applicative. La plate-forme est capable d'intervenir à plusieurs niveaux, elle sait :

- modifier le comportement des composants logiciels qui proposent différents niveaux de QdS ;
- modifier la composition de l'application en intervenant sur l'un des points de modification présentés précédemment, à savoir au niveau des groupes, sous-groupes, des composants et des flux de données[44].

Les adaptations de la partie applicative sont donc plus ou moins profondes selon l'origine des problèmes identifiés afin de coller au mieux aux paramètres de QdS. La Figure 15 donne une vue plus détaillée de l'architecture distribuée que nous proposons.

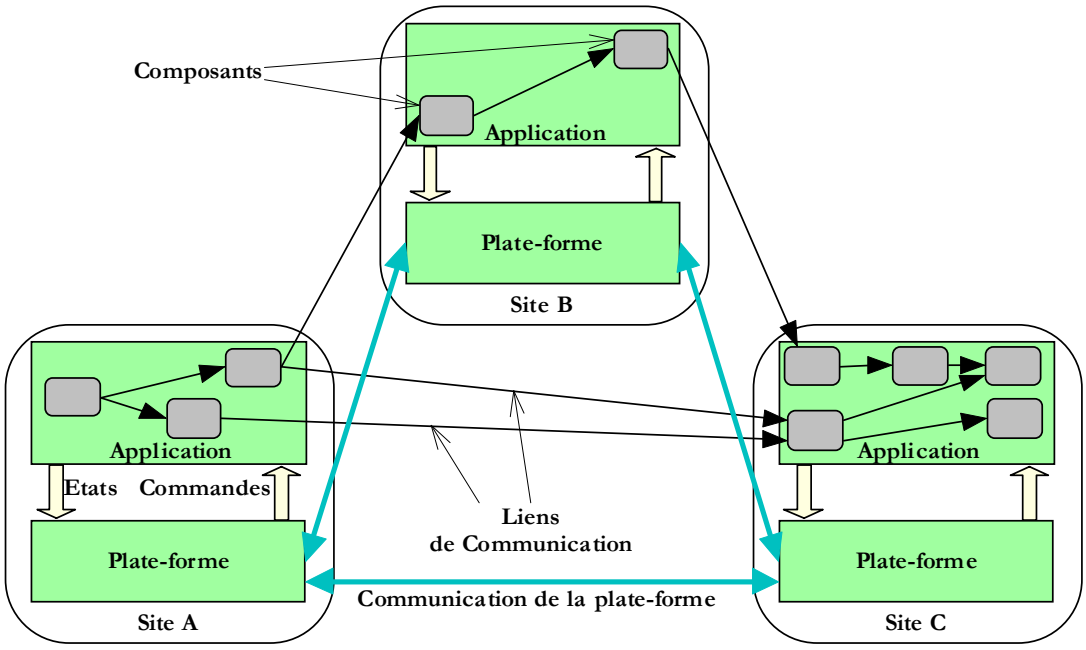


Figure 15 Applications Multimédias Distribuées

Une partie de la plate-forme est localisée sur chaque site de l'AMD et récupère les informations contextuelles de la partie applicative mais aussi de l'environnement

[44] Ces modifications peuvent être exécutées par ajout, retrait, remplacement ou déplacement de groupes, sous-groupes, composants et flux de données.

d'exécution et des utilisateurs. Lorsque la QdS fournie à un moment donné n'est pas satisfaisante, la plate-forme évalue les différents niveaux de qualité qu'elle est capable de fournir. Suivant les informations contextuelles qu'elle possède, elle va pouvoir choisir un niveau de qualité qu'elle va mettre en œuvre en adaptant la structure de la partie applicative. Ainsi, la plate-forme pourra augmenter ou diminuer le niveau de qualité selon les cas de figure.

2.2.2 Modèle Structurel

La plate-forme d'exécution se structure autour de cinq gestionnaires, chacun chargé de réaliser une tâche spécifique. La Figure 16 décrit le modèle structurel de la plate-forme en décrivant ces cinq gestionnaires :

- un gestionnaire d'événements est associé à chaque groupe et donc à chaque service d'une application ;
- un gestionnaire d'évaluation est associé à chaque site de l'application, il est chargé de réaliser une évaluation locale des composants et des flux de données ;
- un gestionnaire de communication[45] est associé à chaque site de l'application, il assure la communication entre tous les sites où l'application et par conséquent la plate-forme sont déployées ;
- un gestionnaire d'utilisateurs est associé à chaque utilisateur et donc à chaque groupe qui lui correspond, il est chargé de recueillir les vœux de ceux-ci ;
- un gestionnaire de supervision est associé à chaque site de l'application, il gère les reconfigurations locales des composants et des flux de données.

---

45 Il fait en sorte que, bien que distribuée, la plate-forme se comporte comme une unité de supervision unique de la totalité de l'AMD.

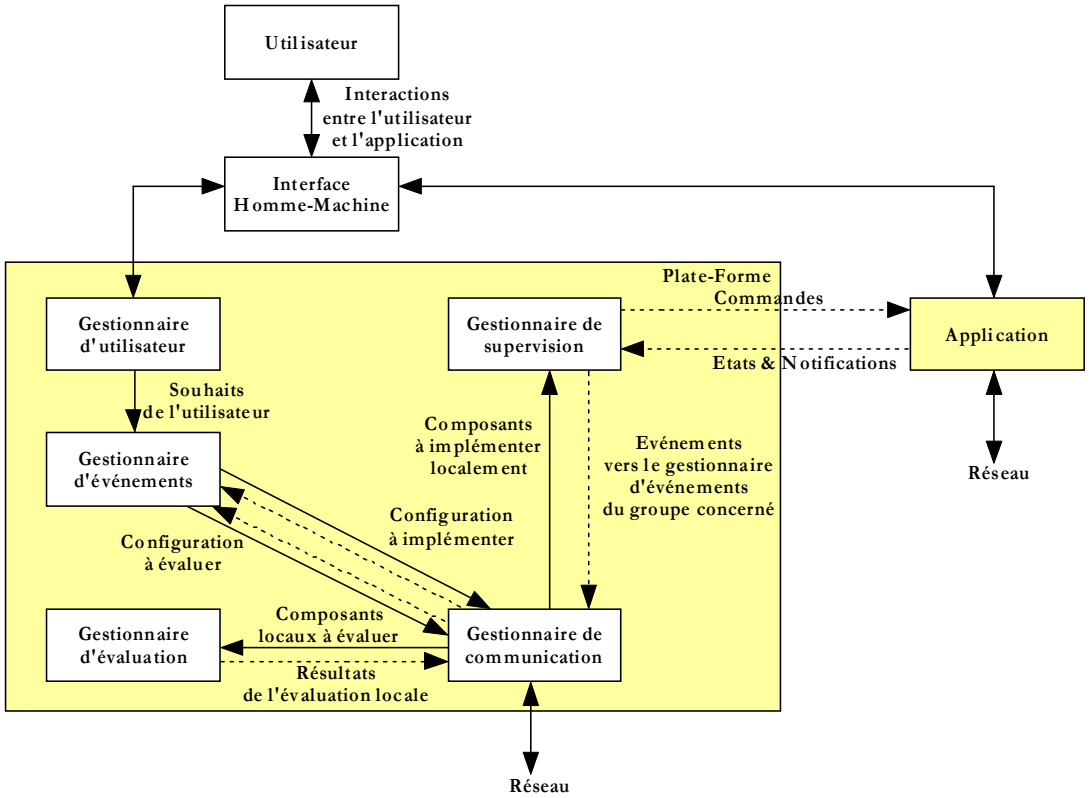


Figure 16 Modèle Structurel de la Plate-forme [LAP06]

Dans la mesure où nous définissons une architecture d'applications distribuées, le modèle structurel complet de la plate-forme se trouve réparti sur chaque site supportant l'exécution de l'AMD. Les cinq gestionnaires seront donc présents sur tous les sites afin de superviser chaque partie locale de l'AMD (cf. Figure 16).

2.2.3 Principes de Fonctionnement

Chaque partie de la plate-forme reçoit des événements en provenance des entités qui composent la partie applicative de chaque site de l'AMD. Ces événements traduisent le comportement local de celle-ci et sont récupérés par le gestionnaire de supervision correspondant. Ils sont ensuite transmis au gestionnaire d'événements via le gestionnaire de communication.

En se basant sur les exigences de l'utilisateur, le gestionnaire d'événements détermine l'événement à traiter en priorité puis la configuration à évaluer. Puis, les gestionnaires d'évaluation de la configuration à évaluer déterminent les caractéristiques des services. Ainsi, la configuration est étudiée en évaluant chaque partie locale de l'AMD. Enfin, le gestionnaire d'événements à l'origine de l'évaluation reçoit les caractéristiques de service de la configuration à évaluer à laquelle il associe une note de QdS (cf. [LAP06] pour obtenir plus de détails sur ces notes de QdS).

En fonction de la note obtenue, le gestionnaire d'événements décide s'il procède ou non à une nouvelle évaluation, s'il reconfigure la partie applicative ou tout simplement s'il traite un nouvel événement. S'il décide de procéder à une reconfiguration, la configuration à déployer est transmise à tous les gestionnaires de supervision de l'AMD via le gestionnaire de communication. Ainsi, la nouvelle configuration est dynamiquement et simultanément déployée sur les sites de l'AMD concernés.

3 Exemple de Déploiement

Dans cette section, le fonctionnement de l'architecture présentée est illustré à travers un exemple d'AMD. Nous voulons, par ce biais, mettre en évidence les possibilités de reconfiguration selon les différents niveaux hiérarchiques du modèle structurel de la partie applicative (cf. Figure 14). L'AMD sur laquelle nous nous basons est une application de formation à distance dont l'architecture est donnée sur la Figure 17. Les utilisateurs de cette application appartiennent à deux groupes distincts : les locuteurs et les téléspectateurs. Nous allons donner des exemples de situations susceptibles de déclencher des phases de reconfiguration.

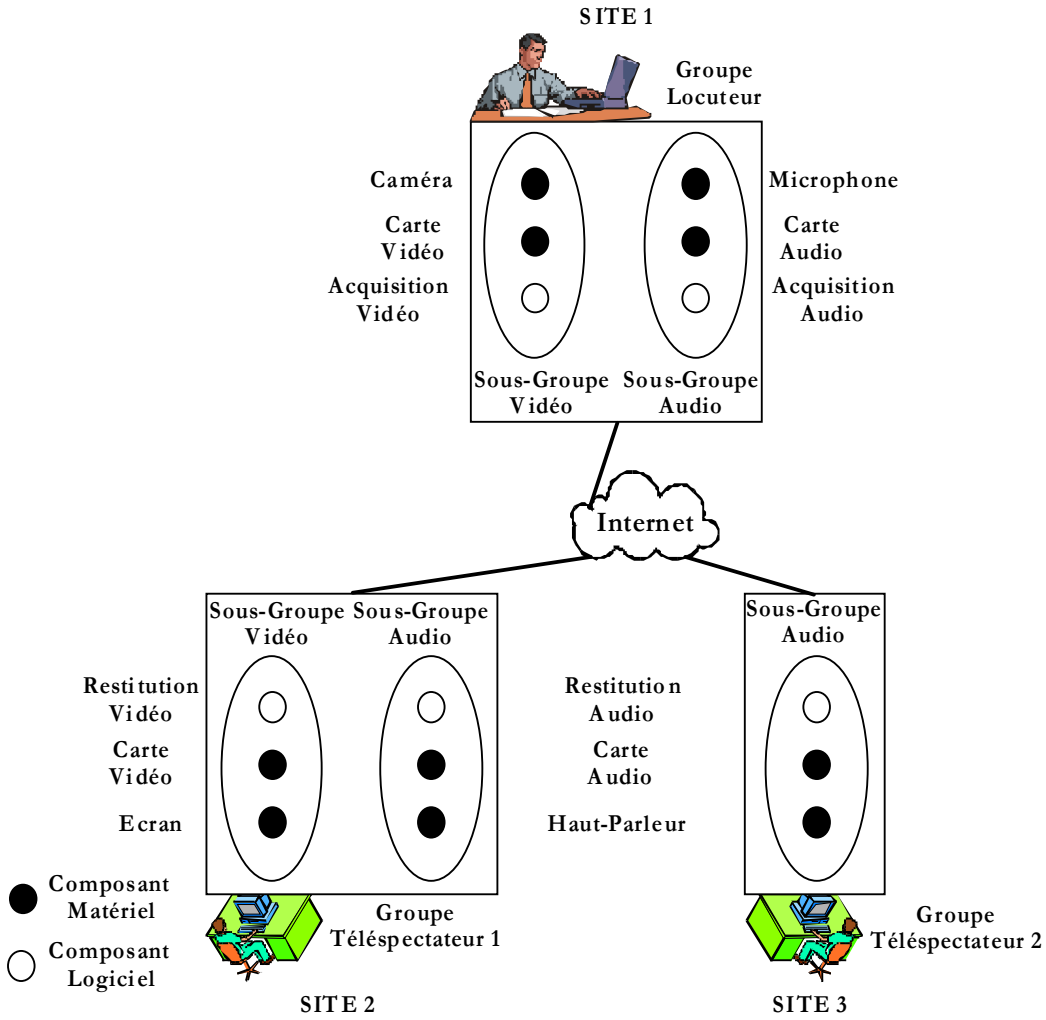


Figure 17 Application de Formation à Distance

L'application (cf. Figure 17) est distribuée sur trois sites. Elle est composée de trois groupes attachés aux trois utilisateurs, à savoir un locuteur et deux téléspectateurs. Pour participer à cette application, chaque utilisateur dispose de périphériques informatiques différents. Le groupe locuteur et le groupe téléspectateur 1 sont tous les deux composés de deux sous-groupes, un pour la vidéo et l'autre pour l'audio. Le groupe téléspectateur 2 est, quant à lui, composé d'un seul sous-groupe pour l'audio car le périphérique qu'il utilise possède des capacités de traitement restreintes.

La Figure 17 représente la configuration initiale de l'AMD. Si au cours de l'exécution l'un des deux téléspectateurs quitte l'application, le groupe correspondant est supprimé par la plate-forme. Si, à nouveau, il se joint au cours dispensé, la plate-forme instancie un nouveau groupe en tenant compte des choix de l'utilisateur et des capacités de son environnement d'exécution (cf. groupe téléspectateur 2).

Avant de débuter le cours dispensé grâce à cette AMD, chaque téléspectateur a indiqué ces exigences ainsi que les caractéristiques de l'environnement qu'il utilise. Ces informations sont récupérées par le gestionnaire d'utilisateurs de la plate-forme locale à ce site (cf. Figure 16). Par exemple les téléspectateurs peuvent indiquer les langues qu'ils comprennent et, de la même façon, le locuteur peut préciser la langue qu'il va utiliser pour le cours. Ainsi, si la plate-forme détecte que la langue du locuteur n'est pas comprise par le téléspectateur 1, elle va lui proposer une traduction à l'aide de sous-titres. Elle instancie, pour ce faire, deux sous-groupes représentant cette fonctionnalité de sous-titrage. Elle en ajoutera un au groupe locuteur et un au groupe téléspectateur 1. Ainsi, ce dernier pourra suivre la formation à distance dans sa langue maternelle. Ce genre d'adaptation au contexte des utilisateurs est important et souvent peu abordé dans les AMD. Il s'agit bien là d'adaptation dynamique. Dans une application comportant plusieurs locuteurs, il faudra que la plate-forme soit capable d'adapter l'architecture pour chaque téléspectateur en fonction du locuteur qui prend la parole.

Nous étudions maintenant des changements de contextes liés à l'environnement d'exécution. Imaginons que le débit de la liaison réseau du téléspectateur 1 baisse. La plate-forme est informée de ce changement par la partie applicative, et plus particulièrement grâce aux flux de données, qui de manière locale ont détecté un ralentissement des transmissions sur le réseau. La plate-forme va alors intervenir en modifiant la composition des groupes et des sous-groupes concernés afin de pallier cet inconvénient. Une solution extrême pourrait, par exemple, consister à supprimer le sous-groupe vidéo. Cette solution n'est satisfaisante que si le téléspectateur 1 a indiqué dans ses préférences qu'il acceptait éventuellement de se passer de la vidéo pour suivre le cours. Une autre solution moins drastique consiste à ajouter au sous-groupe vidéo du locuteur un ou plusieurs composants de traitement permettant de réduire le volume des données à transmettre. Ces composants peuvent être, par exemple, un composant de réduction de la taille de la vidéo ou un composant permettant de mettre le flux vidéo en noir et blanc. La plate-forme peut également choisir de supprimer le sous-groupe audio de ce téléspectateur puisqu'il ne comprend pas la langue parlée par le locuteur et que donc la vidéo et les sous-titres lui sont amplement suffisants. D'autres possibilités de reconfiguration sont possibles comme par exemple la délocalisation de certains composants. La plate-forme choisira la solution à mettre en œuvre en fonction des mesures du contexte dont elle dispose et des vœux exprimés par les utilisateurs. Ainsi, si le téléspectateur 1 a indiqué qu'il préférait ne plus recevoir la bande son plutôt que d'avoir une vidéo en noir et blanc, c'est ce choix qui sera fait à moins, bien entendu, que cette solution ne suffise pas à résoudre le problème. En toute circonstance, la plate-forme appliquera la restructuration la plus acceptable par l'utilisateur

permettant de résoudre le problème. Dans ce but le gestionnaire d'évaluation utilise une heuristique basée sur la notion de proximité de service [LAP06].

4 Synthèse

Nous proposons d'utiliser une plate-forme d'exécution afin de gérer la QdS dans les AMD. Cette plate-forme est chargée de superviser l'exécution de la partie applicative. De plus, elle est capable d'intervenir sur la structure de cette dernière afin de respecter les exigences de QdS définies par les utilisateurs et l'environnement d'exécution.

Afin de réaliser de telles adaptations de manière efficace, nous proposons de structurer les AMD en trois niveaux différents qui peuvent être vus comme trois moyens de modification possibles de l'architecture. On peut ainsi intervenir à chacun de ces trois niveaux pour réaliser une telle gestion. Le premier niveau permet de définir une AMD en terme de services fournis. Cette vision est celle des utilisateurs d'une application. Le second niveau permet de découper chaque service en fonctionnalités requises pour assurer ce service. Le dernier niveau permet de décomposer chacune de ces fonctionnalités en rôles atomiques et en échange d'informations entre eux. Ces rôles et ces échanges sont implémentés respectivement à l'aide de composants et de flux de données.

Cette structuration se justifie par le fait que nous plaçons l'utilisateur au centre de notre approche de QdS. En effet, nous définissons la QdS comme une adéquation entre le service requis par l'utilisateur et celui qui peut lui être fourni compte tenu de l'environnement d'exécution [LAP06]. Il est donc naturel que la vision la plus haute du modèle structurel de la partie applicative soit celle des utilisateurs représentée par les groupes. Les deux niveaux se trouvant au-dessous permettent alors de spécifier la façon dont ces services sont rendus.

La partie applicative peut être adaptée par la plate-forme conformément à cette vision de la QdS. Suivant les paramètres de QdS connus, elle peut déterminer à quels niveaux elle doit intervenir lors des phases de reconfiguration. La Figure 18 résume cette approche.

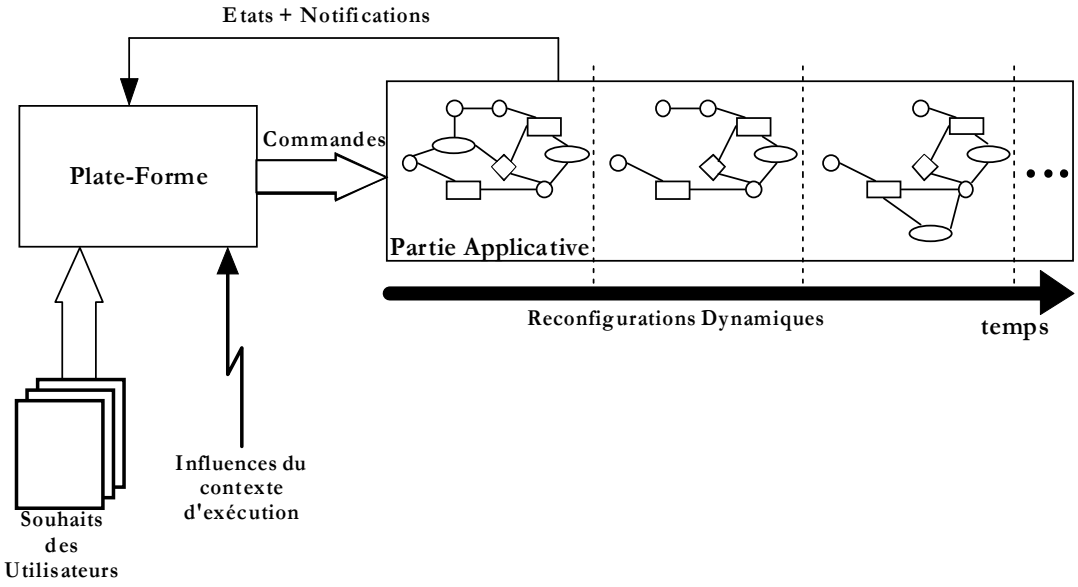


Figure 18 Approche Globale de la QdS dans les AMD

Nous avons abordé dans ce chapitre l'architecture globale des AMD. La plate-forme introduite permet de gérer la QdS en supervisant la partie applicative. Afin, qu'elle puisse mener des interventions ciblées, la partie applicative a été découpée en niveaux. Cette présentation ne serait pas complète sans donner une représentation de chacun de ces niveaux. La plate-forme d'exécution est livrée avec une méthode de conception de la partie applicative [LAP06]. Cette méthode se base sur une décomposition fonctionnelle de la partie applicative, elle utilise comme outil formel des graphes de données. Ces graphes sont intéressants car ils permettent de donner une description fonctionnelle de la partie applicative. Nous allons donc nous intéresser dans le chapitre suivant à cette représentation. Nous verrons qu'elle va nous permettre de justifier la définition des modèles que nous proposons dans cette thèse. En effet, nous allons dégager un certain nombre de spécifications qui vont nous servir de base pour la suite.

Chapitre 4 – Représentation des Applications Multimédias Distribuées

« Il n'y a pas de grande tâche difficile qui ne puisse être décomposée en petites tâches faciles. »

Adage bouddhiste, Dilgo Khyentsé Rinpoché, (1910-1991)

Nous avons présenté l'architecture globale des AMD ainsi que les principes de supervision utilisés par la plate-forme afin de gérer la QdS dans ces dernières. Nous allons maintenant focaliser notre attention sur la méthode de spécification et de conception qui va nous permettre de décrire l'architecture fonctionnelle de la partie applicative. Cette méthode, proposée avec la plate-forme d'exécution, se base d'une part sur la structuration définie dans le chapitre 3 et d'autre part sur l'élaboration de graphes qui permettent de représenter l'architecture logicielle de l'application comme une interconnexion de rôles. Ces rôles sont de granularité différente en fonction du niveau hiérarchique auquel on se situe. Cette méthode est spécifiée dans la thèse [LAP06] que le lecteur intéressé pourra consulter.

1 Introduction

La méthode de conception que nous détaillons dans le présent chapitre permet de décrire l'architecture fonctionnelle des AMD. Elle consiste à identifier les rôles que possèdent une AMD puis les liens entre ces rôles. Les rôles décrivent des fonctionnalités pouvant être réalisées par un ou plusieurs composants logiciels ou matériels. Les liens représentent les flux de données que les rôles échangent afin de réaliser leur fonctionnalité. Cette méthode se base sur une représentation des spécifications fonctionnelles à l'aide de graphes de données, elle se destine aux concepteurs des AMD. L'objectif de ce chapitre est de montrer que cette méthode de conception impose la définition de deux entités fonctionnelles nécessaires pour modéliser puis implémenter ces applications selon une approche basée sur les composants logiciels. La première entité sera définie pour l'implémentation des rôles, elle correspondra à la notion de composant. La seconde entité permettra de transporter les flux de données entre les composants, elle correspondra à la notion de connecteur. Ainsi, à l'aide de ces entités nous pourrons définir une AMD comme une interconnexion de composants et de connecteurs. Elles devront être manipulables par la plate-forme d'exécution afin

qu'elle puisse adapter la structure de la partie applicative selon les besoins. Ces deux entités sont déjà identifiées et proposées par le modèle structurel de la partie applicative (cf. [LAP06] et chapitre précédent).

Nous avons choisi de représenter les spécifications fonctionnelles des AMD à l'aide de graphes car ce sont des modes de représentation adaptés à l'informatique mais surtout car ce sont des outils utilisés pour des problématiques proches de la nôtre [LAY04], [SIN99]. De plus, ces représentations sont facilement adaptables et manipulables. La méthode de conception utilise deux types de graphe [LAP06] :

- le graphe des flots de contrôle permet au concepteur de spécifier les fonctionnalités d'une application à différents niveaux hiérarchiques ;
- le graphe fonctionnel représente l'une des décompositions fonctionnelles possibles d'une application, il est directement dérivé du précédent.

Nous commencerons par mettre en avant les principales propriétés de ces graphes puis nous présenterons les spécificités de chacun. Après quoi, nous nous attacherons à décrire la méthode de conception des AMD en se basant sur un exemple.

Enfin, dans la dernière partie nous allons mettre en évidence ce qu'implique une telle modélisation dans la définition d'un modèle de composants logiciels utilisables pour l'implémentation des AMD. Nous verrons, entre autre, qu'il nous faudra définir deux entités fonctionnelles afin de proposer des implémentations conformes aux spécifications apportées par la méthode de conception. Ce chapitre va nous permettre d'introduire la contribution de cette thèse.

2 Propriétés des Graphes

La représentation que nous utilisons est inspirée des graphes de processus conditionnels définis dans [ELE00]. Ils sont utilisés pour décrire une application comme un ensemble de processus interagissant entre eux. Nous utilisons donc le même principe pour décrire une AMD. Ils permettent d'introduire une représentation abstraite décrite par des graphes directs, polaires et orientés notés $G(V, E_s, E_c)$. Avant de décrire les propriétés des nœuds et des arcs, nous commençons par préciser la notion de rôle qui est centrale dans la méthode que nous présentons.

2.1 Les Rôles

Dans la structure d'une AMD telle que nous l'avons définie, chaque composant participe à la réalisation d'une fonctionnalité et donc d'un service (cf. chapitre 3).

Nous appelons rôle[46] – noté r_i – une fonction identifiable rendue par un ensemble de composants logiciels et matériels. Cette abstraction est utilisée par les graphes de flots de contrôle. Nous appelons rôle atomique (cf. note de bas de page n°1) – noté R_{i-j} – une fonction qui peut être rendue par un unique composant logiciel ou matériel. Ainsi, un rôle atomique représente ainsi le dernier niveau de structuration des AMD. Cette notion est utilisée par les graphes fonctionnels.

En général, les sous-groupes constituent une fonctionnalité observable d'un service, donc d'un groupe. Ceci traduit le fait que la fonctionnalité introduite peut être observée par les utilisateurs des AMD. Ces sous-groupes incorporent donc des composants observables[47]. Ainsi, nous définissons un rôle observable comme un rôle associé à un composant observable. Grâce à ce type de composants, les utilisateurs peuvent percevoir la qualité du service rendu, ils remplissent par conséquent un rôle essentiel dans la gestion de la QdS.

Les graphes sont donc composés d'un ensemble de rôles représentés par des nœuds.

2.2 Les Nœuds

Les nœuds sont notés $P_i \in V$, où V est l'ensemble des nœuds du graphe. Chaque nœud représente l'un des rôles de l'AMD. Ils sont notés r_i (rôle) sur les graphes des flots de contrôle et R_{i-j} (rôle atomique) sur les graphes fonctionnels.

Ces graphes sont polaires, ce qui veut dire qu'ils introduisent deux nœuds fictifs appelés nœud source et nœud puits. Ces deux nœuds sont reliés respectivement aux premières et dernières tâches de l'AMD. Ils représentent de manière conceptuelle l'origine et la destination des flux de données dans une application. Les autres nœuds du graphe sont des successeurs et des prédécesseurs des nœuds sources et puits.

Les nœuds des graphes sont reliés entre eux par des flux de données représentés par des arcs.

---

[46] Nous avons déjà introduit le concept de rôle dans le chapitre précédent afin de pouvoir décrire le modèle structurel de la partie applicative.

[47] Un composant observable est un composant qui propose une fonctionnalité, un rôle observable.

2.3 Les Arcs

Les arcs sont notés $e_{ij} \in E_s \cup E_c$ (avec $E_s \cap E_c = \varnothing$) où $E_s \cup E_c$ est l'ensemble des arcs du graphe. e_{ij} représente l'arc qui sort du nœud P_i et qui rentre dans le nœud P_j. Ces arcs symbolisent les échanges d'informations entre les rôles d'une AMD, ils représentent les flux de données des AMD. On distingue deux types d'arcs :

- les arcs simples – appartenant à E_s – servent à indiquer que les nœuds auxquels ils aboutissent appartiennent à l'AMD quelle que soit la configuration et la QdS choisie ;
- les arcs conditionnels – appartenant à E_c – indiquent que plusieurs choix de configurations sont possibles moyennant de respecter les contraintes d'utilisation de certains rôles. On symbolise de la sorte les dépendances entre les rôles à respecter lors des choix de configurations offrant des niveaux de QdS différents. Par exemple, il est nécessaire de mettre en place un rôle de décompression si un rôle de compression a été introduit auparavant. Ces dépendances – notées O_{Ri-j} – signifient que le rôle R_{i-j} doit être utilisé pour que la condition où se situe la dépendance puisse être validée. Les arcs conditionnels sont également associés à des conditions. Ainsi, un tel arc est franchi lorsque la condition associée – notée C_{Fi} – est validée. Les conditions associées aux arcs conditionnels sont des expressions booléennes exclusives, ce qui veut dire qu'un seul arc conditionnel peut être franchi lors des choix de configuration menés par la plate-forme.

La Figure 19 montre une partie d'un graphe fonctionnel qui utilise des arcs conditionnels. Suivant la QdS requise et celle que l'on peut fournir, on choisira la configuration décrite par le chemin C_{F1} (réduction de la taille initiale de la vidéo), C_{F2} (transmission directe de la vidéo) ou C_{F3} (transformation en noir et blanc de la vidéo). Les trois arcs représentent le même type de flux de données, qui est ici un flux vidéo.

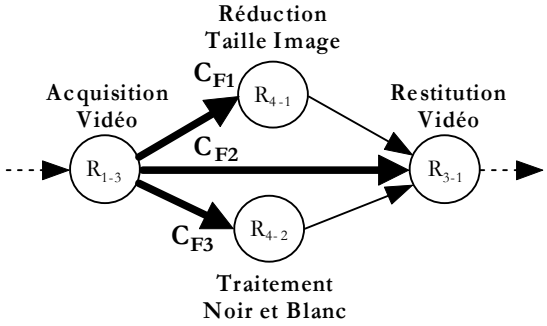


Figure 19 Exemple d'Arcs Conditionnels

Un nœud qui possède plusieurs arcs en sortie est appelé nœud de disjonction (cf. nœud $R_{1\text{-}3}$ sur la Figure 19). Un nœud qui accepte en entrée plusieurs arcs est appelé nœud de conjonction (cf. nœud $R_{3\text{-}1}$ sur la Figure 19).

2.4 Orientation et Cycles

Les graphes sont orientés selon la direction des flux de données. Tous les chemins du graphe trouvent leur origine dans le nœud source et se dirigent vers le nœud puits.

A l'inverse des travaux de [ELE00], les graphes que nous introduisons peuvent contenir des cycles afin de pouvoir spécifier, par exemple, des mécanismes de rétroaction parfois utiles pour l'implémentation d'AMD. C'est le cas par exemple pour implémenter des boucles de retour (feedback, cf. chapitre 1) qui peuvent être utiles pour asservir et adapter les transmissions réseau en fonction de la bande passante disponible (voir à ce sujet le protocole de transport VDP dans [CHE95] et aussi [CEN97]). Un autre exemple de cycle consiste à implémenter une détection de mouvement dans une vidéo afin de piloter une caméra et de mettre en œuvre un suivi automatique de locuteur. Toutefois, afin d'éviter les problèmes d'oscillation, notre méthode interdit à la plate-forme d'intervenir à l'intérieur d'un tel cycle. La justification de cette restriction étant que le rôle attendu n'est obtenu que par la présence du cycle de rétroaction de sorte qu'il doit être considéré comme un seul rôle qu'il est possible de remplacer par un autre mais pas de modifier dans son mode de fonctionnement interne. Dans cet exemple, le rôle de suivi du locuteur est constitué de plusieurs composants sur lesquels la plate-forme ne pourra pas intervenir pour les raisons évoquées.

3 Spécifications Fonctionnelles des AMD

La méthode de spécifications introduite dans [LAP06] utilise un ensemble de graphes. Nous en abordons deux qui sont liés aux travaux présentés dans cette thèse. Le premier est appelé graphe des flots de contrôle et permet de définir une AMD en terme de rôles. Le second est appelé graphe fonctionnel, il est dérivé du premier et correspond à une spécification de l'AMD en terme de rôles atomiques.

3.1 Le Graphe des Flots de Contrôle

Le graphe des flots de contrôle décrit une AMD comme un ensemble de rôles notés r_i. Chaque rôle peut représenter des fonctionnalités de granularité différente (un ou plusieurs composants).

Les rôles sont représentés par les nœuds du graphe. Les arcs, quant à eux, représentent les flux de données que les rôles échangent et introduisent des contraintes de précédence qui signifient qu'un rôle doit s'exécuter avant un autre pour assurer le fonctionnement correct de l'ensemble.

Le graphe des flots de contrôle décrit la composition d'une AMD telle qu'elle est perçue par son concepteur. Ce graphe a pour but de décrire une AMD à un haut niveau. Il correspond à un premier niveau de décomposition fonctionnelle. Il est par la suite dérivé afin de générer des graphes plus précis permettant de spécifier les composants logiciels et matériels nécessaires à la réalisation de chaque rôle exprimé. Ces spécifications sont données sur les graphes fonctionnels que nous présentons maintenant.

3.2 Le Graphe Fonctionnel

Le graphe fonctionnel correspond à un second niveau de décomposition fonctionnelle d'une AMD. Il permet de préciser chaque rôle décrit par le graphe des flots de contrôle en détaillant les rôles atomiques nécessaires à sa réalisation. En effet, ces rôles sont définis par un assemblage de rôles atomiques qui peuvent être réalisés par des composants logiciels et/ou matériels. Les rôles atomiques sont notés R_{i-j}, ils sont décrits par les nœuds du graphe. Ces graphes offrent une vision de bas niveau des AMD.

Les médias transitent dans les AMD sous la forme de flux de données et ce, quel que soit leur type (médias continus ou médias discrets). Ce mode assure une diffusion continue des données aux différents rôles atomiques qui composent une application. Nous reviendrons sur les raisons d'un tel choix dans le chapitre suivant. Nous avons vu dans le chapitre 1 que l'une des particularités des médias est qu'ils peuvent posséder des relations de synchronisation intra- et inter-médias. Les graphes fonctionnels permettent également de spécifier les relations de synchronisation inter-médias en apposant des liens de synchronisation entre les flux de données qui doivent être transportés de façon synchrone. Cette contrainte signifie que l'AMD doit transporter ces flux de façon synchrone, jusqu'à leurs puits ou jusqu'à un nouveau lien de synchronisation qui spécifie la fin de la contrainte sous peine de rendre le service inutilisable. Il

est important de remarquer que ces liens concerneront le plus souvent des flux appartenant à des sous-groupes différents (son et image par exemple).

Ce type de graphe propose une description plus détaillée des AMD puisqu'il la définit en termes de rôles atomiques et de flux de données entre ces rôles. Les groupes et les sous-groupes ainsi définis pourront évoluer par modification de leur structure en ajoutant, en supprimant ou en remplaçant des rôles atomiques. Enfin, l'implantation des composants sur les différents sites d'une AMD, qui n'apparaît pas sur le graphe, offre un dernier degré de liberté à l'adaptation.

La notion de proximité de service qui guide notre approche se retrouve dans ces différentes représentations dans la mesure où elles offrent des vues de plus ou moins haut niveau d'une AMD. Ainsi, lorsque la plate-forme adopte une restructuration issue du même graphe fonctionnel, elle transforme de façon moins profonde l'AMD que lorsqu'elle choisit d'adopter un nouveau graphe fonctionnel issu du même graphe de flots de contrôle.

4 Méthode de conception des Applications Multimédias Distribuées

La méthode de conception des AMD introduite dans [LAP06] est constituée de plusieurs étapes permettant de définir la structure globale d'une AMD et plus particulièrement :

- les spécifications fonctionnelles ;
- les relations de synchronisation de type inter-flux ;
- les spécifications des composants logiciels et matériels disponibles et nécessaires sinon.

Cette méthode utilise, entre autre, les graphes présentés précédemment afin de représenter ces informations.

4.1 Utilisation d'un exemple

Afin de présenter cette méthode, nous nous basons sur un exemple d'AMD. Nous utilisons l'exemple du chapitre 3 qui décrit une AMD de formation à distance. Cette application est constituée de deux types d'utilisateurs : les locuteurs et les téléspectateurs. La configuration initiale de l'application comporte un locuteur et deux té-

léspectateurs[48]. L'application est distribuée sur trois sites qui correspondent aux trois utilisateurs. Le locuteur présente un cours à destination des deux téléspectateurs. Les médias utilisés sont l'audio, la vidéo et éventuellement des sous-titres afin de pouvoir traduire dans une langue différente le discours du locuteur si besoin est.

4.2 Les étapes de la méthode

La première étape consiste à découper l'application en services fournis et fonctionnalités requises pour chaque service. Cette étape consiste en fait à dresser la liste des groupes et des sous-groupes nécessaires à l'AMD. Grâce à cette liste, on va pouvoir définir la structure de l'AMD sur chaque site de l'application.

Une fois ce découpage réalisé, il est nécessaire d'établir le graphe des flots de contrôle de l'AMD sans se soucier du caractère distribué de cette dernière. Cette étape consiste en fait à découper chaque sous-groupe identifié précédemment en un ensemble de rôles nécessaires.

La troisième étape va permettre d'identifier pour chaque rôle défini précédemment les rôles atomiques nécessaires à leur élaboration. Ainsi, le concepteur va pouvoir établir une liste exhaustive des composants logiciels et matériels qui vont permettre de réaliser chaque rôle atomique. Cette liste va lui permettre d'identifier les composants dont il dispose et ceux qu'il faudra implémenter ou se procurer dans le cas de composants matériels. Pour chaque composant, il sera nécessaire de décrire sa nature (logicielle ou matérielle), son rôle atomique, la nature des données consommées/produites, ses principales caractéristiques (performances, contraintes d'utilisation).

La quatrième étape consiste, à partir de ces informations, à établir le graphe fonctionnel en respectant les contraintes de précédences entre les rôles du graphe des flots de contrôle et la liste des composants établie dans l'étape précédente. Pour des raisons de lisibilité, le graphe fonctionnel peut être établi pour l'AMD toute entière ou bien pour chacun des sous-groupes. Dans le dernier cas, les liens de synchronisation, s'il y en a, devront être spécifiés entre les sous-groupes concernés en indiquant les flux de données concernés.

Ces étapes permettent de définir et de représenter de manière globale la structure d'une AMD. Elles sont résumées dans la Table 3.

[48] Ce nombre peut bien évidemment évoluer pendant l'exécution de l'application.

Table 3 Méthode de Conception d'une AMD [LAP06]

Etapes de la Méthode de Conception		
Numéro	**Tâche**	**Résultat**
1	Identification des Groupes et des Sous-Groupes	Structure globale de l'AMD composée de groupes, eux-mêmes composés de sous-groupes
2	Identification des rôles associés à chaque sous-groupe	Graphe des Flots de Contrôle
3	Identification des rôles atomiques associés à chaque rôle	Liste des Composants nécessaires
4	Fusion des Résultats des étapes 1, 2 et 3	Graphe Fonctionnel

4.3 Application de la méthode

Nous déroulons les différentes étapes de la méthode sur l'exemple de formation à distance présenté au § 4.1 et dans le chapitre 3.

4.3.1 Etape 1

L'AMD étudiée se compose de deux types d'utilisateur : les locuteurs et les téléspectateurs. En conséquence, nous identifions deux groupes d'utilisateurs : un groupe pour les locuteurs et l'autre pour les téléspectateurs. La Figure 20 présente ce découpage. Il représente la configuration canonique de l'AMD, c'est-à-dire sa structure de base en terme de services fournis et d'utilisateurs. Chacun de ces groupes peut être instancié plusieurs fois si de nouveaux utilisateurs joignent l'AMD en cours d'exécution.

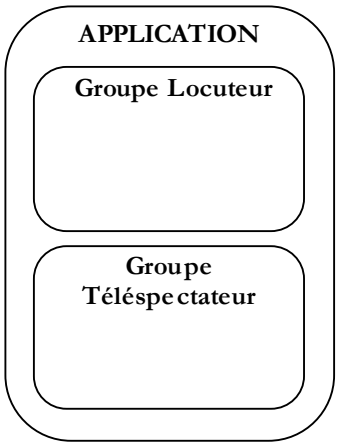


Figure 20 Configuration canonique de l'application

Le groupe locuteur représente l'utilisateur qui va dispenser le cours. Il doit pouvoir fournir l'image, le son et éventuellement une traduction par sous-titres aux téléspectateurs afin de leur permettre de suivre le cours dans une langue comprise. Le locuteur doit donc disposer des équipements nécessaires pour réaliser cette tâche. Le groupe téléspectateur représente un utilisateur qui va suivre le cours. Un tel groupe doit fournir les fonctionnalités nécessaires pour recevoir ce que peut lui fournir le locuteur c'est-à-dire le son, l'image et éventuellement des sous-titres. La Figure 21 montre les configurations canoniques de chacun des groupes de l'AMD. Chaque groupe se compose des sous-groupes permettant de fournir les fonctionnalités requises. Ainsi, chaque groupe est composé de trois sous-groupes : un pour la vidéo, un pour le son et un pour les sous-titres.

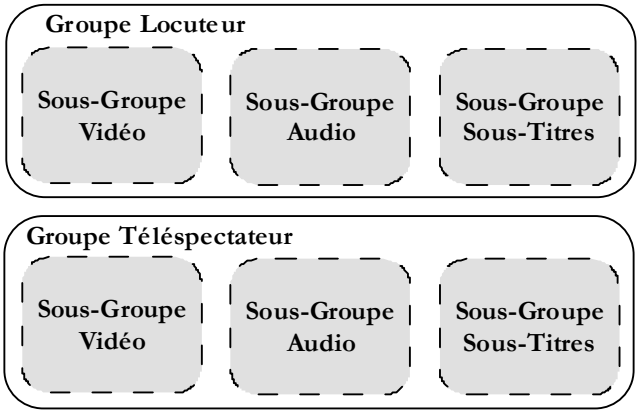


Figure 21 Configuration canonique des groupes

La composition des groupes n'est pas fixe, elle dépend non seulement des exigences des utilisateurs de l'AMD mais aussi des capacités de leur environnement d'exécution. De même, le nombre d'utilisateurs n'est pas fixe, c'est-à-dire qu'il peut y avoir un groupe pour chaque utilisateur. Dans l'AMD étudiée, on prévoit un seul locu-

teur. Par contre, il y aura probablement plusieurs téléspectateurs pour suivre le cours dispensé. Les groupes et les sous-groupes sont instanciés par la plate-forme lors de l'exécution de l'AMD. Cette composition est amenée à être remaniée pendant son exécution en adéquation avec la QdS souhaitée et possible par instanciation, ajout et retrait des groupes et des sous-groupes. Nous considérons qu'initialement cette AMD est structurée comme sur la Figure 22. Elle a trois utilisateurs, à savoir un locuteur et deux téléspectateurs. Elle est donc composée de trois groupes. Le téléspectateur 1 désire une traduction du cours à l'aide de sous-titres. Il utilise un ordinateur qui lui permet de recevoir à la fois la vidéo, l'audio et les sous-titres. Le téléspectateur 2 suit le cours sur son assistant personnel. Les capacités de traitement restreintes de son périphérique lui permettent de recevoir uniquement la parole du locuteur, c'est pour cela que le groupe qui lui correspond contient uniquement un sous-groupe audio.

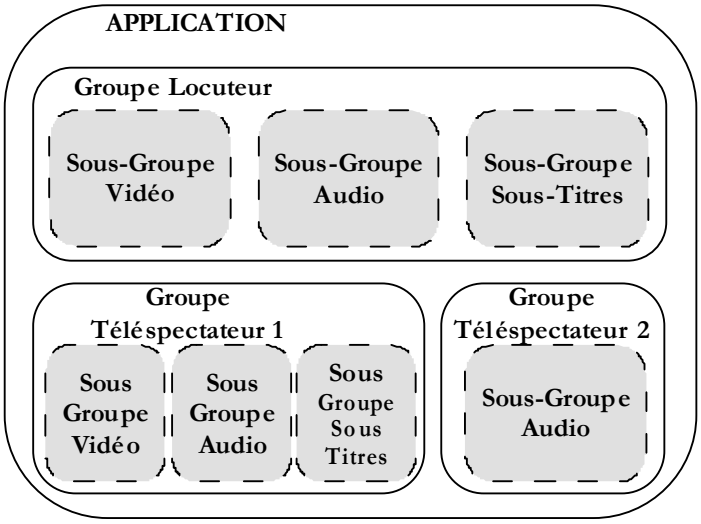


Figure 22 Configuration initiale de l'application

4.3.2 Etape 2

L'issue de l'étape 1 fournit les informations nécessaires pour proposer une décomposition fonctionnelle possible de ces sous-groupes à l'aide du graphe des flots de contrôle. La Figure 23 représente un graphe possible pour cette application. Afin de ne pas surcharger cette figure, nous n'avons pas délimité les sous-groupes.

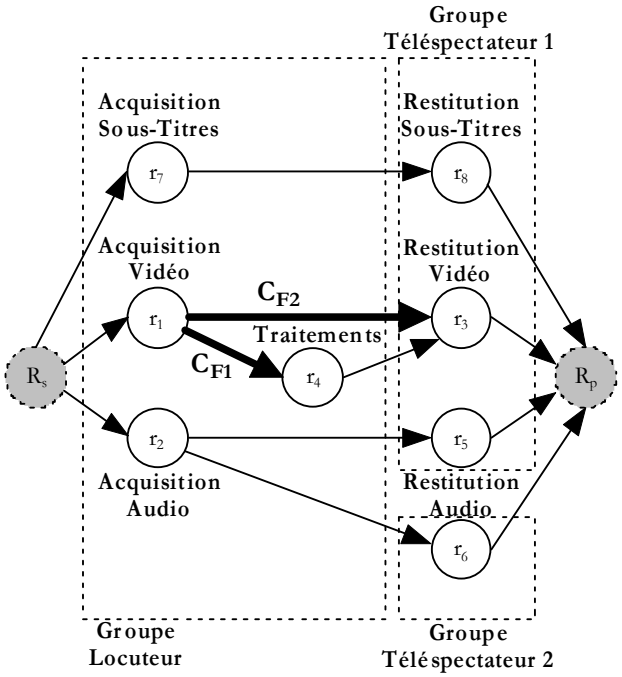


Figure 23 Graphe des Flots de Contrôle

Ce graphe spécifie, à l'aide d'arcs conditionnels, la possibilité d'ajouter des traitements sur la vidéo (chemin C_{F1}) au niveau du sous-groupe correspondant du groupe locuteur. Ces traitements pourront être utilisés, si nécessaire, pour réduire le volume des données (cf. § 2.3 et présentation de l'exemple dans le chapitre 3).

4.3.3 Etape 3

L'étape 3 consiste à établir la liste des rôles atomiques nécessaires pour réaliser chacun des rôles décrits sur le graphe des flots de contrôle de la Figure 23. La Table 4 donne cette description en précisant pour chaque rôle, les rôles atomiques nécessaires, la nature des composants ainsi que la nature des données manipulées.

Table 4 Rôles Atomiques de l'application

Liste des Rôles Atomiques			
Rôle	**Rôle Atomique**	**Nature du Composant**	**Données Manipulées**
r_1 Acquisition Vidéo	R_{1-1} Capture Vidéo (caméra)	matériel	vidéo
	R_{1-2} Acquisition Numérique (carte vidéo)	matériel	vidéo
	R_{1-3} Acquisition Vidéo	logiciel	vidéo
r_2 Acquisition Audio	R_{2-1} Capture Audio (microphone)	matériel	audio
	R_{2-2} Acquisition Numérique (carte son)	matériel	audio
	R_{2-3} Acquisition Audio	logiciel	audio
r_3 Restitution Vidéo	R_{3-1} Restitution Vidéo	logiciel	vidéo
	R_{3-2} Restitution Numérique (carte vidéo)	matériel	vidéo
	R_{3-3} Diffusion (écran)	matériel	vidéo
r_4 Traitements Vidéo	R_{4-1} Réduction de la taille de l'image	logiciel	vidéo
	R_{4-2} Transformation en Noir & Blanc	logiciel	vidéo
r_5 Restitution Audio	R_{5-1} Restitution Audio	logiciel	audio
	R_{5-2} Restitution Analogique (carte son)	matériel	audio
	R_{5-3} Diffusion (haut-parleurs)	matériel	audio
r_6 Restitution Audio	R_{6-1} Restitution Audio	logiciel	audio
	R_{6-2} Restitution Analogique (carte son)	matériel	audio
	R_{6-3} Diffusion (haut-parleurs)	matériel	audio
r_7 Acquisition Sous-Titres	R_{7-1} Acquisition Sous-Titres	logiciel	texte
r_8 Restitution Sous-Titres	R_{8-1} Acquisition Sous-Titres	logiciel	texte/vidéo
	R_{8-2} Restitution Numérique (carte vidéo)	matériel	vidéo
	R_{8-3} Diffusion (écran)	matériel	vidéo

Le concepteur peut, pour chaque rôle atomique réalisé de manière logicielle, implémenter les composants nécessaires. Ainsi, pour un même rôle atomique, il pourra fournir des composants offrant plusieurs niveaux de QdS (plusieurs composants ou des composants paramétrables).

4.3.4 Etape 4

La liste des rôles atomiques (cf. Table 4) établie lors de l'étape précédente sert de base à la réalisation du graphe fonctionnel. De plus, le graphe des flots de contrôle défini lors de l'étape 2 (cf. Figure 23) permet de préciser les relations de précédence

entre les rôles de l'application. A partir de ces informations, il est possible de proposer un graphe fonctionnel qui va décrire l'architecture fonctionnelle initiale de l'application. Il va servir de base pour l'implémentation de l'AMD. La Figure 24 décrit un graphe fonctionnel possible. Les groupes et les sous-groupes ne sont pas délimités afin de ne pas surcharger le graphe. Ces derniers peuvent être facilement identifiés grâce aux résultats des étapes précédentes. On remarque sur ce graphe que l'on a introduit des rôles atomiques logiciels et matériels.

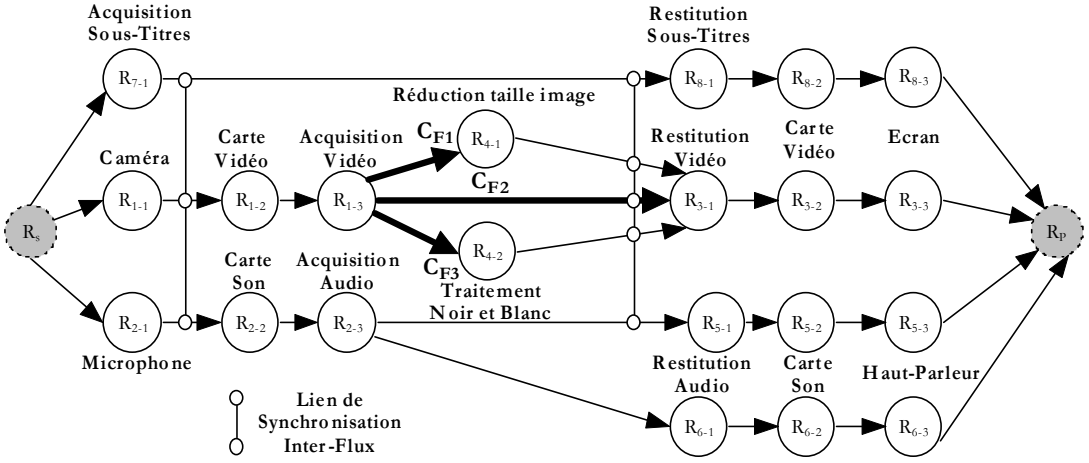


Figure 24 Graphe Fonctionnel de l'application

Il est clair que le téléspectateur 1 qui reçoit les sous-titres, le son et la vidéo doit recevoir, sur son périphérique, l'ensemble de ces médias de façon synchrone afin qu'il puisse suivre le cours correctement. Pour spécifier cette contrainte, nous avons utilisé sur le graphe de la Figure 24 des liens de synchronisation entre ces médias. Cette contrainte se traduit par la nécessité de transporter de façon synchrone les sous-titres, le son et la vidéo, et ce jusqu'à leur restitution.

5 Implémentation de la Partie Applicative d'une AMD

La méthode de conception que nous venons de présenter permet de définir les AMD selon les points de vue structurel et fonctionnel. Le point de vue structurel est utilisé par la plate-forme afin de fournir une vision hiérarchique des AMD. Ainsi, lors des phases de reconfiguration, la plate-forme peut intervenir à différents niveaux selon les violations de QdS à résoudre. Le point de vue fonctionnel permet de donner l'architecture des AMD en terme de fonctionnalités métier. Ces deux points de vue mettent en évidence les besoins pour l'implémentation des AMD conformément aux spécifications introduites par la méthode. Le point de vue fonctionnel permet de déduire les entités fonctionnelles à définir pour réaliser une implémentation des AMD.

Le point de vue structurel décrit la manière de composer ces entités dans les implémentations.

Si l'on regarde de plus près cette méthode, il apparaît de façon claire que l'on va avoir besoin de deux entités fonctionnelles pour réaliser de telles implémentations : une pour permettre la description et l'implémentation des rôles atomiques représentées par les nœuds du graphe fonctionnel et l'autre pour décrire et implémenter le transport des flux de données entre ces derniers représentés par les arcs du graphe fonctionnel.

5.1 Implémenter les Rôles Atomiques : les Nœuds des Graphes Fonctionnels

L'élaboration des graphes fonctionnels représente la dernière étape de la méthode de conception. Ils définissent les AMD comme une interconnexion de rôles atomiques. Il est nécessaire de pouvoir décrire à un niveau d'abstraction élevé un modèle permettant l'implémentation des rôles atomiques et ce quelle que soit la fonctionnalité qu'il propose. Nous avons donc besoin de décrire une entité abstraite chargée de cette tâche puis concrète pour permettre les implémentations.

Nous avons laissé entendre au chapitre 2 notre souhait d'utiliser le paradigme des composants logiciels afin de modéliser les AMD dans notre approche de QdS. Il paraît donc évident que les rôles atomiques seront réalisés par des composants. Ceci dit, certains rôles atomiques sont obtenus par utilisation de ressources matérielles. On distinguera donc les composants logiciels et matériels.

L'intérêt de l'utilisation de composants dans une approche de QdS réside dans le fait que l'on peut fournir plusieurs composants réalisant le même rôle atomique mais proposant des niveaux de qualité différents ou proposer un composant fournissant pour un même rôle plusieurs niveaux de qualité. Si les composants sont correctement définis, leur substitution ou leur paramétrage constitue une solution intéressante lorsque l'on veut augmenter ou dégrader la qualité d'un service. La plate-forme pourra alors utiliser ces entités dans sa tâche de gestion de la QdS.

Chaque composant possède un certain nombre d'entrées et/ou de sorties lui permettant d'interagir avec son environnement. D'après le modèle structurel de la partie applicative (cf. chapitre 3), ces entrées/sorties ne sont autres que les données des AMD, à savoir les médias existant sous la forme de flux de données. Le fonctionnement des composants logiciels est donc basé sur la consommation/production de données de façon continue, dés lors qu'elles existent sous la forme de flux. Ainsi,

l'implémentation des rôles atomiques peut se résumer de manière minimaliste à la manipulation ou au traitement de flux de données.

Il est donc nécessaire de définir le composant logiciel comme une entité fonctionnelle qui va permettre d'implémenter la partie métier d'une application. Il doit, bien entendu, être supervisable par la plate-forme afin que cette dernière puisse récupérer des états sur son fonctionnement mais aussi contrôler son cycle de vie lors des phases de reconfiguration.

Afin de fournir ces propriétés aux composants, nous proposons de les encapsuler dans un conteneur dont le rôle est :

- de fournir au composant métier qu'il contient les flux de données en entrée ;
- de récupérer les flux de données produits par le composant métier en sortie ;
- d'assurer la conservation de la synchronisation intra- et inter-flux de tous les flux qui y transitent ;
- de permettre à la plate-forme de superviser la partie applicative.

Nous reviendrons sur ce point dans le chapitre 6.

5.2 Implémenter le Transport des Flux de Données : les Arcs des Graphes Fonctionnels

Les graphes fonctionnels connectent les rôles atomiques entre eux à l'aide de flux de données. Ces connexions représentent le transport de ces flux entre les rôles atomiques et plus généralement au sein des AMD. Il est nécessaire de pouvoir décrire à un niveau d'abstraction élevé un modèle permettant l'implémentation de ces connexions et ce quelle que soit la nature des flux de données ainsi transportés. Nous avons donc besoin de spécifier une entité abstraite chargée de cette tâche, puis concrète pour permettre les implémentations.

Intuitivement, cette entité va correspondre au concept de connecteur de l'approche des composants logiciels. Ces connecteurs devront permettre le transport des flux de données entre les composants logiciels. Ils doivent donc pouvoir être connectés aux composants logiciels. Ce mode de communication est basé sur la diffusion en continu de données entre les composants.

L'un des manques de la méthode de conception est qu'elle ne considère pas explicitement la dimension réseau des AMD. Néanmoins, si l'on veut disposer d'une ap-

proche efficiente il est nécessaire d'anticiper ce paramètre et de le considérer dés maintenant. Du fait que nous nous intéressons au transport des flux de données entre les composants des AMD, nous pouvons d'ores et déjà affirmer que ce transport pourra être effectué soit localement sur un site, soit au travers du réseau. L'intérêt de l'utilisation du concept de connecteur réside dans le fait que l'on peut définir ces deux types de transport de la même façon. Ainsi, les deux types de connecteurs s'ils sont correctement définis doivent pouvoir être substituables et utilisables de la même façon par la plate-forme d'exécution.

Il est donc nécessaire de définir ce connecteur comme une entité fonctionnelle qui va permettre d'implémenter le transport des flux de données dans ces applications. Il doit également être supervisable par la plate-forme afin que cette dernière puisse récupérer ses états de fonctionnement mais aussi contrôler son cycle de vie lors des phases de reconfiguration. Son rôle est de connecter deux composants entre eux, de récupérer les données produites par l'un et de les acheminer vers l'autre avec des modalités de communication locales ou distribuées.

La définition de cette entité couplée à la définition des composants va permettre de compléter l'architecture logicielle de la partie applicative que nous avons commencé à décrire dans ces deux chapitres. Il doit également assurer la conservation de la synchronisation intra- et inter-flux des données qu'il transporte.

6 Synthèse

La méthode de conception présentée dans ce chapitre permet d'obtenir une décomposition fonctionnelle complète des AMD. En partant des services rendus par une application, on obtient les rôles atomiques et les liens entre eux permettant de définir ces services. Cette méthode offre une vision hiérarchique des AMD intéressante pour la gestion de la QdS qui nous incombe.

Cette méthode permet, en outre, de proposer l'utilisation de deux entités fonctionnelles pour l'implémentation des AMD en concordance avec les spécifications qu'elle impose. L'objectif de cette thèse est, entre autre, de définir un modèle de composants logiciels utilisables pour l'implémentation de ce type d'applications.

La première entité à définir va permettre l'implémentation des rôles atomiques à l'aide du concept de composants logiciels qui acceptent en entrée et en sortie des flux de données. Cette entité est en fait un conteneur de composant métier entièrement supervisable par la plate-forme d'exécution. La seconde entité permet le transport des flux de données entre les composants logiciels. Elle sera définie à l'aide du concept de

connecteur et permettra ainsi de connecter deux composants entre eux. Elle est également entièrement supervisable par la plate-forme d'exécution. Elle doit permettre le transport des flux aussi bien de façon locale que de façon distribuée. Nous touchons du doigt avec l'introduction de cette entité le caractère distribué de ces applications qui n'est pas considéré par la méthode. En effet, il apparaît que les choix d'implantation des composants sur les différents sites d'une AMD ne peuvent être pris en compte qu'au moment du déploiement. Les graphes représentant l'AMD ne peuvent donc pas intégrer les implantations géographiques des rôles. Il n'en reste pas moins vrai que le choix d'une distribution des composants peut avoir une influence non négligeable sur la qualité du service rendu en particulier sur la saturation du réseau. La plate-forme propose, bien entendu, des restructurations basées sur des déplacements de composants. Celles-ci constituent d'ailleurs le premier type de restructuration proposé puisqu'elles n'ont aucune incidence visible sur le service rendu. C'est pourquoi une autre de nos tâches sera, en parallèle avec la définition des modèles, d'intégrer la dimension du réseau.

Avant de se lancer dans la définition de ces entités fonctionnelles et par là même du modèle de composants que nous nous efforçons de définir, nous pensons qu'il est nécessaire de s'attarder sur un aspect capital dans les AMD : les médias. En effet, nous avons abordé dans le chapitre 1 la particularité de ces données et tout ce qu'implique leur manipulation dans ce type d'applications. Nous avons vu avec ces deux chapitres que les médias sont supportés dans les AMD que nous définissons par une structure de type flux de données. Dans la prochaine partie, nous allons aborder les raisons d'un tel choix. Les graphes fonctionnels de la méthode de conception permettent de spécifier des relations de synchronisation inter-flux. La synchronisation ainsi matérialisée doit aussi trouver sa place lors de l'implémentation des AMD. Elle constitue une propriété fonctionnelle qui se retrouve tant au niveau des arcs que des nœuds. A ce sujet, nous avions identifié dans le chapitre 1 deux sources de désynchronisation possibles lors du transport des médias à travers le réseau et lors du traitement des médias. La diversité des médias et la possibilité de pouvoir les traiter et les transmettre dans les AMD nous confortent dans l'idée qu'il est indispensable d'étudier de plus près ces données afin d'en fournir un modèle unique. Ce modèle sera utile au modèle de composants que nous allons définir et, en particulier, aux deux entités fonctionnelles mises en évidence précédemment c'est-à-dire autant au niveau du traitement que du transport. Ce modèle unique de données doit permettre de conserver les relations de synchronisation inter-flux et ce malgré les traitements de certains et les transferts réseaux. Nous plaçons de fait cet aspect au niveau structurel des médias en définissant ce mo-

dèle de données. Leur gestion et leur synchronisation doivent être réalisées de manière dynamique et non de manière statique comme beaucoup de travaux le prônent (cf. chapitre 1). Cette façon de faire nous permet de disposer d'un haut niveau d'intégration des médias dans les AMD à travers l'Internet. L'étude des médias réalisée dans le chapitre 1 ainsi que la méthode de conception présentée dans ce chapitre permettent de dresser la liste des exigences que nous devons prendre en considération. La Table 5 résume ces exigences. Les propriétés des médias doivent être intégrées dés la conception des AMD si l'on veut éviter la perte de leur sémantique. En conséquence, nous proposons un modèle qui permet une intégration aisée et forte des médias à l'aide d'une structure commune basée sur le concept de flux de données. Tous les flux de données possèdent des propriétés de séquence et des relations de synchronisation. Les données qui composent ces flux sont datées et ordonnées par des numéros de séquence. Les relations de synchronisation deviennent explicites grâce aux dates associées aux données des flux. Enfin, ce modèle nous permet de définir des politiques de synchronisation que l'on utilisera dans les AMD afin de conserver ces propriétés.

Table 5 Exigences et Solutions proposées

Exigences	Solutions
Une structure qui assure une intégration forte de tous les médias et qui plus est de toutes les données	Les médias et données existent sous la forme de flux
Considérer l'ordre physique des données dans les flux	Utilisation d'une approche basée sur une horloge logique
Considérer les relations de synchronisation intra-flux	Utilisation d'une approche basée sur une horloge physique mais aussi à travers la définition des politiques de synchronisation
Permettre la manipulation et le traitement des flux dans les applications	Définir une structure interne des flux manipulable par les unités d'implémentation du modèle de composants
Permettre le transfert des données à travers le réseau Internet	Définir les objets et mécanismes utiles à cette tâche
Classer les données	Définition basée sur les contraintes temporelles des flux
Assurer la synchronisation inter-flux	Définition des politiques de synchronisation

Partie 3 – Modèle de Flux et Modèle de Composants

Cette partie aborde la contribution essentielle de cette thèse en introduisant les modèles de flux et de composants que nous préconisons pour le développement des AMD. Elle s'articule autour de deux chapitres. Le premier chapitre décrit le modèle Korrontea qui permet la modélisation des données que l'on est susceptible de rencontrer dans les AMD. Ces données sont appelées médias et sont constituées de flux d'informations. Elles intègrent des propriétés qui donnent tout leur sens au contenu informationnel qu'elles supportent. Nous proposons, à travers la définition de ce modèle, de prendre en compte ces caractéristiques afin d'éviter une perte de sémantique dommageable à l'utilisation des AMD. Ce chapitre introduit également des politiques afin de conserver les relations de synchronisation dans les AMD.

Le second chapitre décrit un modèle de composants nommé Osagaia développé pour l'implémentation des AMD conformément aux spécifications fonctionnelles introduites par la méthode de conception. Ce modèle manipule les données des AMD sous la forme de flux et ce, conformément au modèle Korrontea présenté dans le chapitre suivant. Il s'articule autour de la définition de deux entités fonctionnelles qui vont permettre respectivement l'implémentation des rôles atomiques et le transport des données dans les AMD. De plus, nous allons voir que la méthode de conception et le modèle Korrontea imposent à Osagaia de fournir des services non-fonctionnels nécessaires et réutilisables dans de telles implémentations.

Chapitre 5 – Korrontea, un Modèle de Flux de Données

« Flow: To rhyme continuously in the same rhyme scheme without stopping. »

The Rap Dictionary

Le chapitre 1 de l'état de l'art nous a permis de toucher du doigt l'importance des données dans les AMD. Elles sont appelées médias et existent sous différentes formes. En effet, nous avons vu dans ce même chapitre qu'habituellement on les séparait en médias continus et médias discrets. Cette classification permet de donner une idée sur les propriétés spécifiques de ces données. Ces propriétés sont importantes à considérer dés lors que l'on veut profiter pleinement du contenu informationnel que les médias supportent. En effet, leur importance se justifie par le fait que ces propriétés sont directement liées à la sémantique des médias. Nous avons également montré dans le chapitre 1 que la distinction des médias continus et discrets était à notre sens peu précise et incomplète.

La partie précédente nous a permis de justifier de la nécessité de définir deux entités fonctionnelles afin de fournir une implémentation conforme aux spécifications fonctionnelles. De par l'architecture proposée, ces deux entités sont amenées à manipuler des médias afin de réaliser les fonctionnalités voulues des AMD. Le problème que nous nous posons est de définir un modèle générique c'est-à-dire faisant abstraction des types des données qui vont être manipulées. Les graphes fonctionnels nous permettent de spécifier certaines caractéristiques comme la synchronisation inter-médias. Cependant, ces spécifications sont incomplètes car d'autres propriétés implicites doivent également être considérées. La difficulté qui se présente est donc de fournir des abstractions fonctionnelles capables de manipuler les médias et ce, quel que soit leur type. En effet, la définition d'entités capables de manipuler les médias en fonction de leur type et de leurs caractéristiques est bien trop complexe et limitée. Il en est pour preuve le nombre conséquent de travaux sur les AMD où les types de média manipulés sont définis a priori (cf. chapitre 1). Par conséquent, nous pensons que la manipulation des médias dans les AMD ne peut être faite de façon efficiente que si ces données sont structurées de la même façon et possèdent des propriétés communes. Le présent chapitre propose une solution à ce problème.

Les relations de synchronisation constituent un exemple des propriétés particulières des médias. Ces relations sont de deux ordres, elles existent entre les données

d'un ou de plusieurs médias (cf. chapitre 1). Elles caractérisent une grande partie de la sémantique des médias. Ces relations permettent de lier plusieurs données entre elles afin d'enrichir le contenu informationnel qui est dans ce cas défini par l'ensemble des médias utilisés (par exemple son et images dans une vidéo). La conservation de ces propriétés dans les AMD n'est pas une tâche facile surtout lorsque les médias existent sous des formes différentes. Pour ce faire, certaines approches proposent des solutions basées sur un multiplexage fort des types de données connus (par exemple MPEG). Nous avons vu dans le chapitre 1 les inconvénients de ce type de solution. L'avantage de l'approche proposée est de pouvoir traiter et synchroniser tout types de format en proposant une structure unique qui facilite ces manipulations.

Ce chapitre présente le modèle de flux de données Korrontea[49] qui va servir de base pour l'implémentation d'applications qui manipulent des médias. Il se base sur les propriétés de ces derniers ainsi que sur les spécifications fonctionnelles définies par la méthode de conception.

1 Introduction

Typiquement, les AMD manipulent deux types de données communément appelées médias continus et médias discrets (cf. chapitre 1). Si les premiers existent en général sous la forme de flux de données continus intégrant des relations temporelles (synchronisation), les autres forment un ensemble indivisible de données finies dans le temps (le média existe à un instant précis et l'ensemble des données qui le compose est nécessaire pour sa représentation). Cette distinction implique donc des méthodes différentes pour la manipulation des médias dans les applications, ce qui complexifie les architectures logicielles. Lorsque l'on ne peut pas a priori dresser une liste exhaustive des médias qui seront utilisés par une AMD, cette manière de faire pose un problème. En effet, une AMD dotée d'un service de traduction ne sait pas a priori si ses différents utilisateurs vont recourir à ce service ou pas. Si la traduction est réalisée à l'aide de sous-titres, considérés comme un média discret, on n'a donc aucun moyen de savoir à l'avance si ce type de média sera manipulé ou pas. De plus il est fort probable qu'en cas d'utilisation de ce moyen de traduction, les sous-titres soient couplés sémantiquement avec les images et éventuellement l'audio. Ce lien sémantique ne peut être connu que pendant l'exécution de l'AMD car on n'a pas la connaissance a priori des sous-titres que l'on utilisera et de la langue de traduction. Une solution qui manipule

[49] Le mot « Korrontea » signifie « le flux de données » en langue basque.

les médias selon leur type n'est pas satisfaisante à nos yeux en raison du peu de flexibilité qu'elle propose. En ce qui nous concerne, nous proposons de nous placer à un niveau d'abstraction élevé en définissant une structure de données unique ainsi que des propriétés communes.

Nous proposons d'adopter la structure des médias continus sous la forme de flux de données qui semble intéressante pour cette tâche. Une telle structure impose de considérer des propriétés liées à la séquence des données et donc au facteur temps. Nous allons voir que ces propriétés constituent un atout considérable pour la manipulation des relations de sémantique entre les médias. Tous les médias, et qui plus est toutes les données possèderont donc ces propriétés. Elles seront définies de manière explicite et nous utilisons pour cela une approche basée sur un mécanisme d'horloges physiques et logiques. Ainsi, la sémantique des médias est conservée et les relations temporelles comme la synchronisation peuvent être prises en compte.

La notion de flux de données telle que nous la définissons est basée sur le facteur temps. Ceci implique que tous les médias possèderont un comportement temporel. Ce comportement sera défini lors des phases de spécifications pour distinguer les médias par rapport à ce critère. Nous verrons que cette distinction va nous aider à définir la tâche de manipulation des médias par les entités du modèle de composants. Grâce à cette propriété, nous pouvons prévoir le comportement de ces derniers.

Enfin, la dernière partie du modèle Korrontea introduit des politiques de synchronisation. Ces politiques vont permettre de répondre aux spécifications des liens de synchronisation dans les graphes fonctionnels. Ces liens traduisent le fait que plusieurs médias doivent être transportés de façon synchrone dans les AMD. Nous utilisons donc ces politiques pour assurer ce type de transport. Elles sont basées sur les comportements temporels des flux de données. Grâce à elles, on va pouvoir définir les algorithmes qui vont permettre de conserver ces liens.

Nous commençons par définir la structure de données que nous allons utiliser pour représenter les médias ainsi que toutes les données dans les AMD.

2 Une Structure Commune : Les Flux de Données

Les données susceptibles d'exister dans une AMD sont différentiées par la nature des informations supportées mais aussi par l'organisation de ces informations. Après un bref rappel sur les types de média, nous présentons la structure que nous allons utiliser pour les supporter dans les AMD auxquelles nous nous intéressons.

2.1 Rappel sur les Médias

Dans les AMD, on distingue généralement les médias continus et les médias discrets. Les médias continus sont composés d'une séquence continue d'échantillons a priori infinie et les médias discrets sont constitués d'un ensemble indivisible de données ponctuelles. Les deux types diffèrent principalement par la structure des données et par le facteur temps. Ceci implique différentes méthodes pour les manipuler et donc la nécessité de connaître a priori leur type et leur structure. Certains frameworks définis pour la programmation de telles applications procèdent de la sorte, c'est le cas par exemple de l'API JMF développée par Sun Microsystems [SUN99]. De plus, nous avons vu qu'il est possible de définir des relations de synchronisation inter-médias portant aussi bien sur des médias continus que sur des médias discrets. Cette possibilité implique de connaître les relations temporelles qui peuvent lier ces différents médias et donc d'associer la notion de temps aux médias discrets en considérant par exemple leur date de production. Ainsi, on peut établir des relations entre les médias qui possèdent des dates de production très proches. Cette remarque conforte notre choix dans le fait de disposer d'une structure commune.

2.2 Définition des Flux de Données

Utiliser des modalités différentes pour manipuler les médias dans les AMD n'est pas une solution satisfaisante. En effet, une telle approche est très complexe à mettre en œuvre et peu efficace en raison de la diversité des structures de données que l'on peut utiliser pour supporter les différents types de média. Il est donc plus sage de disposer d'une structure commune qui permettra une manipulation générique plus aisée des médias mais également facilitera l'implémentation des traitements et des transferts. De plus, la possibilité d'avoir des relations de synchronisation inter-médias entre médias de différents types nous conforte dans cette direction.

Par conséquent, notre proposition est d'utiliser la structure des médias continus. Ainsi, les médias seront supportés dans les AMD à l'aide de flux de données. Certaines études sur les systèmes multimédias affirment que le flux de données est une solution intéressante pour une intégration forte des médias [BLA96]. Ainsi, les AMD seront constituées de composants connectés entre eux par des flux de données correspondant aux arcs des graphes fonctionnels (cf. chapitre 3 et 4). Les traitements et les transferts sont déclenchés et s'exécutent lorsque les données issues des flux sont disponibles dans les entités responsables de ces tâches. Les données issues des flux transitent entre ces entités de manière continue. Ce type de fonctionnement est guidé par les données [CPU01] qui prennent donc une importance particulière dans cette ar-

chitecture. De plus, l'architecture logicielle choisie est similaire aux architectures de type « pipes & filters » [GAR94].

Chaque flux de données est produit par un unique composant localisé sur un site de l'application. Ces composants définissent les origines des flux de données. Ils correspondent à des composants d'acquisition ou de création des flux. Nous appelons ces composants particuliers des sources localisées.

Définition 1 *Source Localisée*

On appelle source localisée SL un couple (C, S) où C est un composant et S le site de l'application où se trouve C.

Cette définition sert de base au modèle Korrontea, on verra que les relations de synchronisation inter-flux ne peuvent être conservées qu'entre les flux dont les sources localisées sont situées sur le même site.

Cette définition apporte la caractéristique importante de distribution. Ainsi, ces composants représentent des entités concrètes de création des flux de données. Une propriété importante de ces composants est le site où ils sont localisés. Tout le modèle Korrontea est basé sur cette dernière.

Un flux de données se compose d'une suite continue d'informations appelées échantillons. Les échantillons sont, par exemple, une image issue d'une vidéo, une image fixe, du texte, des événements, etc.

Définition 2 *Flux de Données*

Un flux de données f est composé d'une séquence a priori infinie d'échantillons de taille finie. Chaque échantillon appartenant à f est produit par la même source localisée SL=(C, S).

Attributs d'un flux de données :

- *sourceLocalisee(f)=SL*
- *source(f)=C*
- *site(f)=S (désigne le site de l'application où le flux est produit)*

La création des flux de données par les sources localisées consiste en la production d'échantillons de manière continue régulière ou irrégulière. Les échantillons sont produits dans un format de codage correspondant au type des informations supportées par le flux. Les flux de données sont considérés comme une séquence a priori infinie d'échantillons. Bien entendu, certaines données peuvent exister sous la forme d'une séquence finie d'échantillons, c'est le cas par exemple des médias stockés dans des fichiers. Ceci se traduit par le fait que, lorsque tous les échantillons d'un tel flux sont restitués (son utilisation est terminée), le flux n'existe plus. En effet, tant qu'un

flux existe dans l'AMD, c'est que des échantillons peuvent y transiter ; dans le cas contraire il disparaît car il n'y a plus de rôles associés à son traitement. Il est important de noter que les opérateurs, propriétés et politiques définis par ce modèle sont applicables uniquement sur des flux considérés comme a priori infinis et non vides.

Les échantillons d'un flux de données possèdent une propriété de séquence. Cette séquence peut être constante (cas des médias continus et de certains médias discrets) ou non (cas général des médias discrets) en fonction de la nature du flux. Au moment de la création des flux mais aussi pendant leur manipulation, traitement et transport au sein d'une AMD, cette propriété doit être explicitement conservée. Afin de la définir, nous utilisons une horloge logique pour estampiller chaque échantillon ou ensemble d'échantillons à l'aide d'un entier qui permet de déterminer l'ordre de deux échantillons ou ensemble d'échantillons successifs. Cet entier permet de conserver la propriété de séquence. Chaque composant d'une AMD fournit, pour chaque flux qu'il produit et traite, un mécanisme qui délivre des entiers incrémentés. Nous avons décrit ce principe sur la Figure 25. Lorsqu'un échantillon ou un ensemble d'échantillons est produit par un composant, un compteur s'incrémente d'une unité. L'entier ainsi défini est attribué à la production du composant. Les données produites sont totalement ordonnées dés lors que l'on suppose que cette production s'opère de façon séquentielle. Ainsi, les traitements ne risquent pas d'altérer cette propriété de séquence puisque nous la mettons à jour après chacun d'eux. Cet entier est appelé le numéro de séquence des échantillons. Ce concept d'horloge logique a initialement été défini par Lamport [LAM78] et est habituellement utilisé dans les systèmes distribués afin de préserver l'ordre des données lors de leur production [FID88], [MAT88]. Ainsi, il est possible d'étendre certains mécanismes proposant des solutions locales (exclusion mutuelle, file d'attente, etc.) à des environnements distribués. Ce temps logique n'est lié qu'à la succession des données. Par exemple, si a et b sont des données, alors l'horloge logique permet de vérifier : $a<b \Rightarrow$ horlogeLogique(a)<horlogeLogique(b) où horlogeLogique(i) correspond au numéro de séquence de la donnée i. Cette expression n'est pas une équivalence car la relation entre a et b est une relation d'ordre partiel tandis que la comparaison des entiers horlogeLogique(a) et horlogeLogique(b) est une relation d'ordre total. Nous allons voir que cette solution va nous permettre de restituer les données en séquences ordonnées.

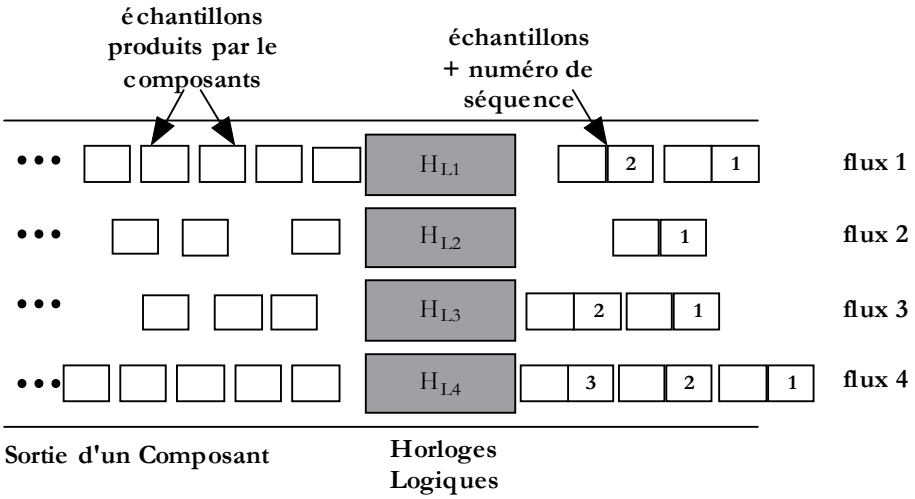


Figure 25 Principe des Horloges Logiques utilisées

La manipulation des médias dans les AMD dépend des caractéristiques intrinsèques de l'information supportée. En effet, nous avons vu dans le chapitre 1 que les données qui composent un média peuvent être décomposées en plusieurs niveaux hiérarchiques (cf. l'exemple de la vidéo dans ce même chapitre). Les applications sont définies afin de pouvoir manipuler l'un de ces niveaux. Cependant, pour certains médias il est nécessaire de manipuler des unités de plus haut niveau que l'échantillon si l'on veut pouvoir respecter de manière logicielle les relations de synchronisation intra-flux. Pour prendre en considération ce problème, un nombre suffisant d'échantillons sera rassemblé dans des unités d'information (similaire au concept d'unités d'information logiques définies par [BLA96]). Par exemple, l'audio est manipulée dans les programmes au moyen de segments audio qui rassemblent approximativement les échantillons qui correspondent à 200 millisecondes de son. Cette quantité d'information est associée à un numéro de séquence.

Définition 3 *Unités d'Information*

Nous appelons unité d'information UI, un couple (E, i) ou $E=\{e_1, e_2, \ldots, e_n\}$ *est un ensemble fini d'échantillons du même flux de données f et i un entier utilisé pour définir la propriété de séquence pour ce flux de données.*

Attributs des Unités d'information :

- *flux(UI)=f (flux de données auquel appartiennent les éléments de E)*
- *echantillon(UI)=E*
- *numeroSequence(UI)=i*
- *sourceLocalisee(UI)=sourceLocalisee(flux(UI))*
- *source(UI)=source(flux(UI))*

- *site(UI)=site(flux(UI)) (site de création de l'unité d'information)*

La constitution et donc la taille des unités d'information dépend des médias considérés mais aussi des spécifications de l'AMD à développer en termes de type de traitement et de manipulation. Ainsi, les unités sont lues et écrites seules ou en séquences synchrones et ce, de manière continue. Les traitements et les transports sont déclenchés quand les unités d'information sont disponibles dans les entités respectives. Ces unités mises bout à bout forment donc un flux de données. Aucune hypothèse n'est faite sur le codage des échantillons contenus dans E. Ainsi, il est possible d'utiliser des images codées en JPEG, des séquences de vidéo codées en MPEG mais aussi tout autre type d'informations comme du texte, des événements ou un seul plan de couleur d'une image. C'est l'une des volontés fortes de notre modèle que de n'être lié à aucun format ou standard de codage des informations. Nous proposons de mettre dans les unités d'information tous les échantillons créés au même moment par une source localisée. Par exemple, dans une vidéo nous rassemblerons tous les pixels qui constituent une image dans une même unité d'information. Selon les spécifications, on peut décider de composer les unités d'information avec une, deux ou trois images, etc.

2.3 Le Dilemme de l'Horloge

Le séquencement des données dans les flux introduit le facteur temps. Cette caractéristique existait déjà pour les médias de type continus, elle est désormais une propriété de tous les flux de données. En procédant de la sorte, les relations de synchronisation inter-médias pourront être facilement conservées.

Dans un flux de données deux unités successives sont séparées par une certaine durée qui correspond aux relations de synchronisation intra-médias. Cette propriété temporelle doit donc être exprimée de façon explicite dans les flux de données. Pour la mettre en œuvre, il faut être capable de mesurer le temps physique c'est-à-dire de mesurer des durées. Nous choisissons pour ce faire une approche basée sur une horloge physique.

Une horloge physique est un mécanisme utilisé pour définir des valeurs temporelles et pour estampiller les données dans les systèmes distribués. La différence principale avec une horloge logique est que, dans ce cas, nous manipulons le temps réel. Ainsi, grâce à une telle horloge on peut appréhender les concepts de valeur temporelle, d'intervalles de temps et de durée. Un modèle de temps physique a été défini par l'OMG (Object Management Group) dans la spécification d'un profil UML pour

l'analyse et la conception de systèmes temps réel (SPT[50]) [OMG05]. Nous avons reproduit ce modèle sur la Figure 26 car il permet d'illustrer la définition des horloges physiques. Une horloge physique génère périodiquement des événements appelés tics d'horloge. Les dates que l'on associe à chaque donnée correspondent à des valeurs temporelles de l'horloge physique.

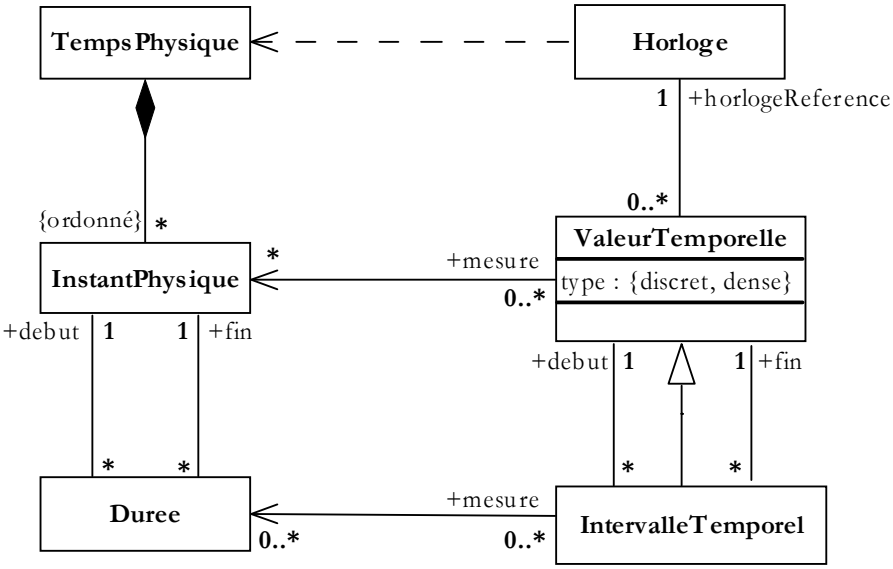


Figure 26 Modèle du Temps Physique [OMG05]

Dans les AMD, l'utilisation de telles horloges trouve son intérêt lorsque l'on veut conserver et restituer les relations de synchronisation des médias. Ces relations se traduisent par des intervalles de temps et des liens temporels entre les données. Par exemple, les relations intra-flux se traduisent par l'intervalle temporel entre deux unités d'information successives. De la sorte, on amène dans ces systèmes la connaissance du temps physique (au sens de la Figure 26). Ainsi, les temps de production (valeurs temporelles) des échantillons d'un flux peuvent être connus et utilisés pour conserver les relations de synchronisation. En conséquence, nous proposons d'utiliser de tels mécanismes dans notre propos.

Cependant, lorsque l'on s'intéresse aux systèmes distribués il est important de prendre en compte le problème de localité. En effet, ces systèmes sont distribués sur plusieurs sites qui connaissent tous des horloges physiques différentes. Cette remarque implique qu'une valeur ou un intervalle mesuré sur un site peut n'avoir aucune signification sur les autres. Ceci peut être pris en compte de deux façons différentes : soit en

[50] SPT signifie Schedulability, Performance and Time.

utilisant une horloge physique globale, soit à l'aide d'horloges physiques locales à chaque site du système distribué.

2.3.1 Horloge Physique Globale

Lorsque l'on considère, par exemple, deux événements qui se produisent sur des sites distincts d'un système distribué, on ne peut pas affirmer de façon certaine lequel a eu lieu avant l'autre ni même mesurer le temps qui sépare l'occurrence de ces deux événements sauf à utiliser une horloge physique globale permettant de disposer d'une référence temporelle unique pour le système considéré. La valeur temporelle donnée par ce type d'horloge décrit le même temps physique sur tous les sites où un tel système est déployé. Cette horloge peut être choisie parmi les horloges physiques de chaque site d'un système où être une horloge extérieure utilisée à cet effet.

Cependant, la définition d'une horloge globale n'est pas une tâche si facile que l'on pourrait le croire. En effet, disposer d'une référence unique impose que chaque site doit connaître cette référence. Une façon de faire est d'utiliser les horloges physiques de chaque site et de les convertir vers une référence unique en déterminant l'offset ou le décalage qui existe entre ces horloges et l'horloge globale que l'on veut utiliser. En effet, il est peu probable que les horloges des différents sites possèdent la même valeur temporelle au même moment. De la même manière, les horloges et les montres que nous utilisons dans la vie de tous les jours ne fournissent pas exactement le même temps physique. Ce phénomène est connu comme la dérive des horloges les unes par rapport aux autres [ABA95] et donc le décalage ou offset entre les horloges physiques d'un système distribué est variable dans le temps.

Pour appuyer notre argumentation, nous utilisons un exemple où l'on désire synchroniser un flux audio et un flux vidéo créés sur des sites différents. Cet exemple est décrit à l'aide du schéma de la Figure 27. Chacun des sites représentés dispose de sa propre horloge physique : $H_1(t)$ pour le site 1, $H_2(t)$ pour le site 2 et $H_3(t)$ pour le site 3. Les échantillons qui constituent chacun des flux sont estampillés par rapport à l'horloge physique de leur site de création : $H_1(t)$ pour l'audio et $H_2(t)$ pour la vidéo. Si l'on veut proposer des relations de synchronisation inter-flux sur le site 3 en prenant pour référence temporelle $H_3(t)$, alors il faut être capable de définir l'offset entre $H_1(t)$ et $H_3(t)$ et l'offset entre $H_2(t)$ et $H_3(t)$ afin de disposer de la même référence temporelle et donc de pouvoir synchroniser les deux flux par rapport à cette dernière. De plus, il faut tenir compte des temps de transfert Δ_1 et Δ_2 des échantillons issus de chacun des flux sur le réseau. Ces paramètres sont difficiles à évaluer en raison du non déterminisme des réseaux de communication. On pourrait imaginer d'autres solutions qui consisteraient par exemple à utiliser une horloge globale différente de $H_1(t)$, $H_2(t)$

et $H_3(t)$. Les mêmes problèmes se poseraient alors. Des approximations ou des approches statistiques sont utilisées afin de pouvoir définir tous les paramètres nécessaires à l'utilisation d'une horloge globale. Plusieurs recherches s'orientent vers ce type de solution [ABA95], [ELS02].

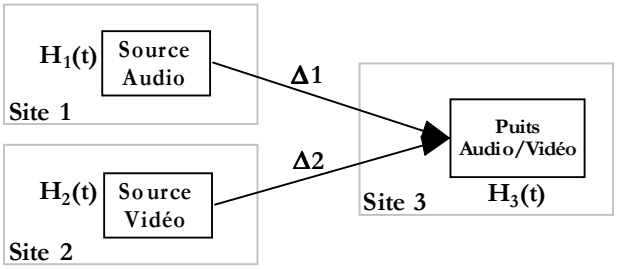


Figure 27 Synchronisation à l'aide d'une Horloge Globale

Dans le domaine du multimédia, ces approches vont influencer le degré de synchronisation des flux dans une AMD car il est actuellement impossible d'avoir une synchronisation des horloges parfaite et absolue à travers un réseau comme l'Internet [IEE04]. Il est impossible d'utiliser une horloge globale sans introduire une marge d'erreur. Ces approches sont donc intéressantes si l'on veut créer de la synchronisation entre des flux faiblement dépendants sémantiquement. Par exemple, dans le cas de la Figure 27, le site 2 pourrait produire un flux vidéo et le site 1 pourrait produire un flux audio qui serait alors une musique à synchroniser avec l'image. Ce type de synchronisation est plus ou moins artificiel et ne nécessite pas d'avoir des relations temporelles strictes. Notre propos n'est pas de créer de la synchronisation mais bien de la conserver de sorte qu'il s'agit de ne pas perdre une information (celle de synchronisation) qui a existé lors de la création des flux. Néanmoins, nous verrons dans le chapitre suivant que le modèle de composants que nous définissons autorise ce genre de synchronisation à l'aide des composants de traitement. Elle n'est donc pas exclue de notre champ d'action. Dans le modèle Korrontea, nous n'utiliserons pas d'horloge physique globale.

2.3.2 Horloge Physique Locale

Une solution à base d'horloge locale consiste à utiliser l'horloge physique propre à chaque site d'un système distribué pour dater les événements et données produits sur ce site. Ainsi, des relations temporelles peuvent être exprimées seulement entre données ou événements provenant du même site. On peut savoir, par exemple, le temps qui sépare l'occurrence de deux événements produits sur le même site.

Si l'on considère le cas d'utilisation décrit sur la Figure 28, avec une telle approche on peut conserver les relations temporelles entre des flux créés sur le même site.

Cet exemple est plus probable que celui de la Figure 27 car en principe on conserve les relations de synchronisation inter-flux entre des médias couplés sémantiquement. Sur la Figure 28, les médias peuvent être issus, par exemple, de l'enregistrement du son et de l'image d'un locuteur. Les échantillons de ces médias sont datés par l'horloge $H_1(t)$ du site 1. Ils sont ensuite transmis vers le site 2 sur lequel on peut retrouver la synchronisation à l'aide des dates apposés sur les échantillons issus des médias. Ces dates n'ont pas de signification par rapport à l'horloge $H_2(t)$ du site 2, ce sont en fait des étiquettes temporelles relatives au site qui les a produites.

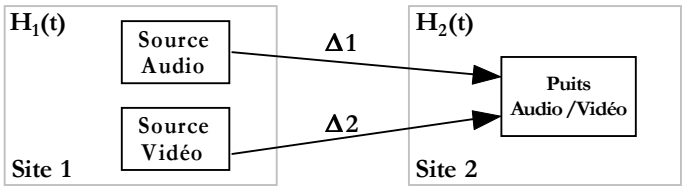


Figure 28 Synchronisation à l'aide d'une Horloge Locale

Dans cet exemple, les temps de transfert réseau Δ_1 et Δ_2 ne sont plus à considérer pour conserver les relations temporelles entre ces médias. Par contre, ces temps doivent être pris en compte pour compenser les retards d'un flux par rapport à un autre si on les transmet séparément.

Cette approche est bien évidemment plus facile à mettre en œuvre que la précédente. Elle nous semble plus intéressante car elle permet d'exprimer des relations entre des flux de données liés sémantiquement c'est-à-dire provenant d'un même site. Nous basons donc toute notre approche de synchronisation sur ce type d'horloge. Des flux ne pourront être conservés synchrones que si leur source localisée (cf. définition 1) est physiquement située sur le même site. Par conséquent, nous utilisons des horloges physiques locales.

Définition 4 *Horloges Physiques Locales*

Tous les sites d'une application multimédia distribuée possèdent un composant unique H_p (horloge physique locale du site S). Les échantillons produits par la source localisée SL=(H_p, S) sont des entiers strictement croissants appelés étiquettes temporelles.

Ces entiers considérés indépendamment les uns des autres n'ont pas de significations propres. Ils sont liés au temps réel par une relation de proportionnalité. Ainsi, si SL produit l'entier n_1 à l'instant t_1 et l'entier n_2 à l'instant t_2, alors $(n_2-n_1)=k_S(t_2-t_1)$ où k_S est une constante de temps connue sur le site S.

Hypothèse 1 *Constante de Temps*

Sur chaque site d'une application multimédia distribuée, la constante k_S est la même. Le composant H_p utilise l'horloge système mais il adapte les informations fournies par celle-ci de façon à ce que la

constante k_S soit toujours la même (par exemple 10 millisecondes). Ainsi, on a k_S=k où k est une constante de temps définie pour une application.

Cette hypothèse traduit le fait que tous les H_p d'une application possèdent la même vitesse. Les étiquettes temporelles produites par $SL_1=(H_{p1}, S_1)$ et par $SL_2=(H_{p2}, S_2)$ ne peuvent pas être comparées. On peut seulement comparer les différences entre des étiquettes temporelles produites respectivement par ces sources localisées. C'est une caractéristique importante de notre modèle. En effet, la connaissance de la constante de temps k permettra de respecter les relations de synchronisation lors de la diffusion des médias et ce, quel que soit le site d'où ils proviennent.

2.4 Définition des Flux Synchrones

L'horloge physique locale dont nous dotons le modèle Korrontea va nous permettre désormais de dater les informations dans les flux de données. Les relations de synchronisation entre des flux de données provenant du même site peuvent ainsi être conservées pendant leur transport dans les AMD grâce aux étiquettes temporelles délivrées par les horloges physiques locales. Nous définissons un objet appelé tranche synchrone qui a pour but de rassembler un ensemble d'unités d'information créées ou capturées sur le même site et qui correspondent au même instant de création.

Définition 5 *Tranches Synchrones*

Soit $F=\{f_1, f_2, \ldots, f_n\}$ un ensemble de flux de données tel que $\forall f_i \in F$, on a $site(f_i)=S$. On appelle une tranche synchrone, définie sur F, un objet qui contient une quantité d'informations qui correspond au même intervalle de temps et que l'on définit de la manière suivante : $TS=(t, U)$ où t est une étiquette temporelle créée par $SL=(H_p, S)$ et U un ensemble fini d'unités d'information tel que $\forall x \in U$, $flux(x) \in F$. Les numéros de séquence des unités d'information sont définis de sorte que : soit $A_f=\{e \in U / flux(e)=f\}$ alors $\forall e \in A_f$, $1 \leq numeroSequence(e) \leq |A_f|$ (cardinal de l'ensemble A_f) et $\forall e_1, \forall e_2 \in A_f$, $e_1 \neq e_2 \Leftrightarrow numeroSequence(e_1) \neq numeroSequence(e_2)$.

Attributs des Tranches Synchrones :

- *$flux(TS)=F$ (cet attribut permet de connaître les flux de données contenus dans une tranche synchrone)*
- *$etiquetteTemporelle(TS)=t$*
- *$unitesInformation(TS)=U$*
- *$site(TS)=S$*

Dans ces tranches synchrones, on va donc regrouper une étiquette temporelle et des unités d'information issues des flux de F. Ces unités d'information seront donc

ordonnées entre elles par les numéros de séquence délivrés par les horloges logiques. Une tranche synchrone ne doit pas nécessairement contenir des unités d'information de chaque flux de F. Cette propriété nous permettra de regrouper dans une même tranche synchrone des médias continus et des médias discrets pour lesquels il n'y aura pas d'unités d'information associées à certaines dates.

Les étiquettes temporelles sont associées aux tranches après la production des unités de traitement qu'elles contiendront. Les unités d'information sont estampillées avec des numéros de séquence. Dans une même tranche, ces numéros partent de 1 pour la première unité d'information et vont jusqu'à n[51] pour la dernière unité et ce pour chaque flux.

Les tranches synchrones vont être en fait les unités de transport synchrone des données dans les applications. Ainsi, les composants des AMD vont s'échanger des tranches synchrones entre eux dans lesquelles ils seront capables de récupérer les unités d'information des flux qu'ils doivent traiter. Les autres unités des autres flux sont dans les tranches uniquement dans le souci de conserver les relations de synchronisation. Ces tranches mises bout à bout forment également des flux que nous appelons flux synchrones.

La Figure 29 donne l'exemple d'une tranche synchrone composée de quatre flux de données f_1, f_2, f_3 et f_4. Nous pouvons remarquer que pour certains flux les tranches synchrones peuvent ne pas contenir d'unités d'information, c'est le cas du flux f_3 sur la Figure 29.

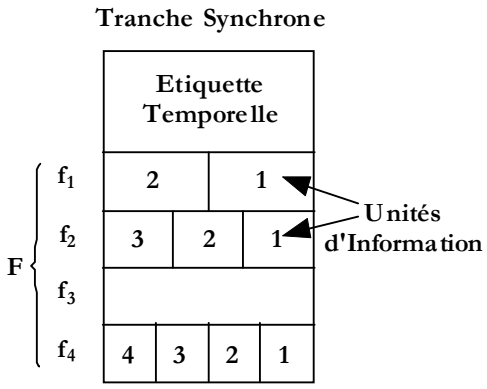


Figure 29 Exemple de Constitution d'une Tranche Synchrone

[51] On considère que chaque flux de données appartenant à une tranche synchrone contient n unités d'information.

Définition 6 *Flux Synchrones*

Un flux synchrone FS est composé d'une séquence a priori infinie de tranches synchrones qui possèdent les mêmes flux c'est-à-dire que $\forall TS \in FS, flux(TS)=F$ *(cf. définition 5). Les tranches synchrones d'un même flux synchrone ont toutes des étiquettes temporelles différentes :* $\forall TS_1, \forall TS_2 \in FS, TS_1 \neq TS_2 \Leftrightarrow etiquetteTemporelle(TS_1) \neq etiquetteTemporelle(TS_2)$.

Propriété des Flux Synchrones :

- $\forall TS_1, \forall TS_2 \in FS, site(TS_1)=site(TS_2)=S$
- *Preuve : D'après la définition 6, on sait que* $\forall TS_1, \forall TS_2 \in FS$, *on a :* $flux(TS_1)=flux(TS_2)=F$. *D'après la définition 5, F est un ensemble de flux de données tel que* $\forall f_i \in F, site(f_i)=S$.

Attributs des Flux Synchrones :

- $flux(FS)=F$ *où F est l'ensemble des flux de données (cf. définition 2) qui composent les tranches synchrones*
- $site(FS)=S$ *(tous les flux de données qui composent FS proviennent du même site)*

Ainsi, les unités d'information des flux de données seront transportées dans des tranches synchrones sous la forme de flux synchrones. A ce sujet, une unité d'information appartient à une et une seule tranche synchrone. De la même manière, une tranche synchrone est un élément d'un et d'un seul flux synchrone.

En fonction de la composition des tranches synchrones, on distingue deux sortes de flux synchrones : les flux primitifs et les flux composés.

Définition 7 *Flux Synchrones Primitifs*

Un flux synchrone est défini comme primitif quand $|flux(TS)|=1$.

Un flux primitif ne contient qu'un seul type de données (bande son, sous-titres, etc.), il ne possède aucune relation de synchronisation inter-flux. Ce type de flux peut se rencontrer dans une AMD mais son utilité est surtout importante pour les traitements puisque les composants traitent des flux primitifs.

Définition 8 *Flux Synchrones Composés*

Un flux synchrone est défini comme composé quand $|flux(TS)|>1$.

Les flux de données liés par des relations de synchronisation inter-flux sont rassemblés dans des flux composés. Pour chaque flux de données, les tranches synchrones des flux composés contiennent les unités d'information qui correspondent au même intervalle de temps. Les politiques de synchronisation que nous donnons dans le présent chapitre servent à créer les flux composés.

Dans les AMD, toutes les données existent sous la forme de flux primitifs ou de flux composés. La Figure 30 récapitule les concepts introduits en schématisant la composition des flux primitifs et des flux composés.

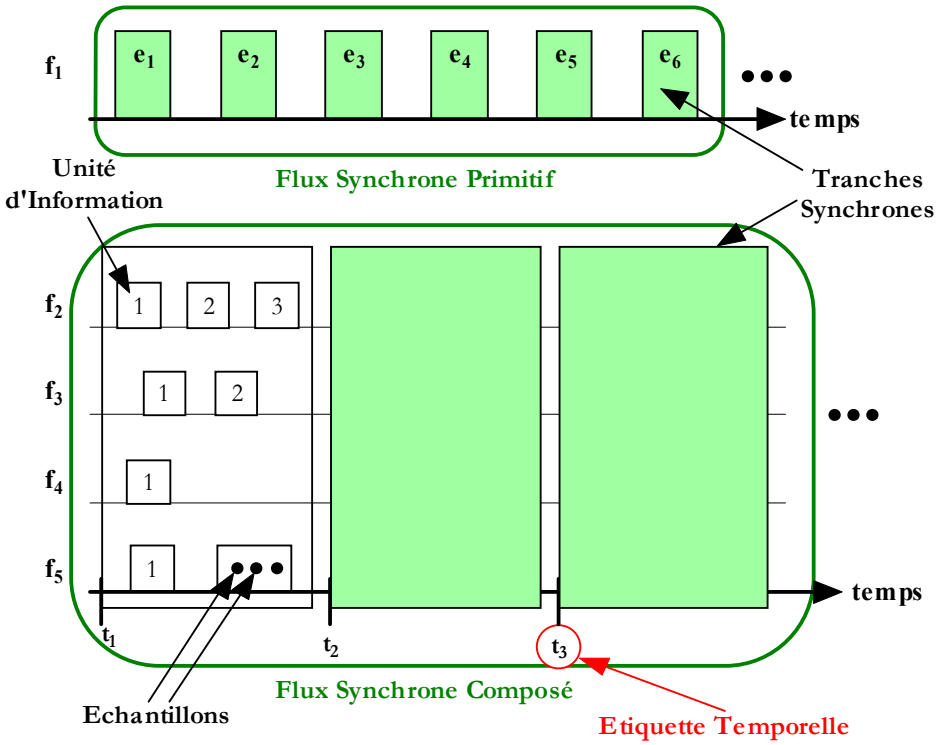


Figure 30 Composition des Flux Primitifs et des Flux Composés

2.5 Relations d'ordre des Flux Synchrones

Dans les flux synchrones tels qu'ils sont définis dans la section précédente, les données sont séquencées par un numéro ainsi que par une étiquette temporelle (cf. Figure 30). Ainsi, les unités d'information des flux synchrones peuvent être ordonnées par le temps et par un numéro. On peut donc définir des relations d'ordre total et strict entre les unités d'information d'un même flux synchrone. Cette propriété sera utilisée pour permettre une diffusion des flux synchrones ordonnée.

Propriété 1 *Relations d'ordre total strict (< et >) entre les unités d'information d'un flux de données à l'intérieur d'un flux synchrone FS*

Nous définissons < (respectivement >) une relation d'ordre total strict entre les unités d'information d'un même flux de données à l'intérieur d'un flux synchrone. Soit UI_1 *et* UI_2 *deux unités d'information* $\in FS$*, avec* $flux(UI_1)=flux(UI_2)$*, et* $UI_1 \in unitesInformation(TS_1)$*,* $UI_2 \in unitesInformation(TS_2)$*. On a :*

- $t_1=etiquetteTemporelle(TS_1)$, $n_1=numeroSequence(UI_1)$

- t_2=*etiquetteTemporelle*(TS_2), n_2=*numeroSequence*(UI_2)

$UI_1 < UI_2$ (respectivement $UI_1 > UI_2$) $\Leftrightarrow$ ($t_1 < t_2$) OU (($t_1 = t_2$) ET ($n_1 < n_2$)) (respectivement ($t_1 > t_2$) OU (($t_1 = t_2$) ET ($n_1 > n_2$))). Si $t_1 \neq t_2$, l'ordre est donné par l'étiquette temporelle. Si $t_1 = t_2$, l'ordre est donné par le numéro de séquence.

<u>Preuve :</u> Une relation d'ordre strict est irréflexive et transitive. De plus, cet ordre est total si : $\forall$ UI_1 et UI_2 deux unités d'information $\in$ FS, avec flux(UI_1)=flux(UI_2), $UI_1 < UI_2$ OU $UI_1 = UI_2$ OU $UI_2 < UI_1$.

- *<u>Irréflexivité :</u> $\forall$ x $\in$ unitesInformation(TS). etiquetteTemporelle(trancheSynchrone(x)) < etiquetteTemporelle(trancheSynchrone(x)) est faux car ces deux étiquettes temporelles sont égales. De la même manière, numeroSequence(x)<numeroSequence(x) est faux car ces deux nombres sont égaux. Ainsi, la relation x<x est fausse.*
- *<u>Transitivité :</u> $\forall$ UI_1, UI_2 et UI_3 trois unités d'information $\in$ FS, avec flux(UI_1)=flux(UI_2)=flux(UI_3), et $UI_1 \in$ unitesInformation(TS_1), $UI_2 \in$ unitesInformation(TS_2), $UI_3 \in$ unitesInformation(TS_3) tel que $UI_1 < UI_2$ et $UI_2 < UI_3$. On a : t_1=etiquetteTemporelle(TS_1), n_1=numeroSequence(UI_1) et t_2=etiquetteTemporelle(TS_2), n_2=numeroSequence(UI_2) et t_3=etiquetteTemporelle(TS_3), n_3=numeroSequence(UI_3).*
  - *Si $UI_1 < UI_2$ est dû au fait que $t_1 < t_2$, $UI_2 < UI_3 \Rightarrow t_2 \leq t_3$, donc on a $t_1 < t_2 \leq t_3 \Rightarrow UI_1 < UI_3$.*
  - *Si $UI_1 < UI_2$ est du au fait que $t_1 = t_2$ ET $n_1 < n_2$ et $UI_2 < UI_3$ est du au fait que $t_2 < t_3$, alors on a $t_1 = t_2 < t_3 \Rightarrow UI_1 < UI_3$. Si $UI_2 < UI_3$ est du au fait que $t_2 = t_3$ ET $n_2 < n_3$, alors on a $t_1 = t_2 = t_3$ et $n_1 < n_2 < n_3 \Rightarrow UI_1 < UI_3$.*

Ainsi, < est une relation d'ordre strict. Le même type de preuve peut être établi pour la relation >.

Nous devons maintenant prouver que ces relations sont totales. Soit t_1=etiquetteTemporelle(TS_1), n_1=numeroSequence(UI_1) et t_2=etiquetteTemporelle(TS_2), n_2=numeroSequence(UI_2). t_1 et t_2 sont produites par une horloge physique locale, ce sont des entiers et nous savons que < est une relation d'ordre total sur les entiers, donc on a :

- *soit $t_1 < t_2 \Rightarrow UI_1 < UI_2$*
- *soit $t_2 < t_1 \Rightarrow UI_2 < UI_1$*
- *soit $t_1 = t_2$: UI_1 et UI_2 sont des unités d'information qui vérifient flux(UI_1)=flux(UI_2)=f. D'après la définition 5, on sait que $\forall$ e_1, $\forall$ $e_2 \in A_f$, $e_1 \neq e_2 \Leftrightarrow$ numeroSequence(e_1)$\neq$numeroSequence(e_2), donc on a $UI_1 \neq UI_2 \Leftrightarrow n_1 \neq n_2$ et par contraposition $UI_1 = UI_2 \Leftrightarrow n_1 = n_2$. La relation < est un ordre total sur les entiers, donc on a :*
  - *$n_1 < n_2 \Rightarrow UI_1 < UI_2$*
  - *ou $n_2 < n_1 \Rightarrow UI_2 < UI_1$*

- *ou $n_2=n_1 \Rightarrow UI_2=UI_1$.*

Ainsi, < est une relation d'ordre total strict. Le même type de preuve peut être établie pour la relation >.

En fait, la relation < signifie « antérieur à » et la relation > signifie « postérieur à ».

De la même manière, on peut définir des relations d'ordre total strict entre les tranches synchrones des flux synchrones.

Propriété 2 *Relations d'ordre total strict (< et >) entre les tranches synchrones d'un flux synchrone FS*

On définit < (respectivement >) comme une relation d'ordre total strict entre les tranches synchrones d'un même flux synchrone par : $\forall TS_1 \in FS$, $\forall TS_2 \in FS$, $TS_1<TS_2$ (respectivement $TS_1>TS_2$) $\Leftrightarrow$ etiquetteTemporelle(TS_1) < etiquetteTemporelle(TS_2) (respectivement estiquetteTemporelle(TS_1) > etiquetteTemporelle(TS_2)).

<u>Preuve :</u> $\forall TS_1, TS_2 \in FS$ avec $TS_1 \neq TS_2$, on a etiquetteTemporelle(TS_1) $\neq$ etiquetteTemporelle(TS_2) (cf. définition 6). De plus, etiquetteTemporelle() est un entier et donc on a etiquetteTemporelle(TS_1)<etiquetteTemporelle(TS_2) ou etiquetteTemporelle(TS_1)>etiquetteTemporelle(TS_2). Ce qui fait que l'on a $TS_1<TS_2$ ou $TS_1>TS_2$. Ainsi, toutes les tranches synchrones d'un flux synchrone peuvent être ordonnées par les relations < et > $\Rightarrow$ ces relations sont des relations d'ordre total strict entre les tranches synchrones d'un flux synchrone.

Ces relations permettent de compléter la définition des flux synchrones par une propriété définie sur les objets qui rentrent dans leur composition.

Propriété 3 *Propriétés des Flux Synchrones (2)*

Les tranches synchrones d'un flux synchrone sont totalement ordonnées par les relations d'ordre total strict < et > (cf. propriété 2). Les unités d'information d'un même flux de données dans les tranches synchrones sont totalement ordonnées par les relations d'ordre total strict < et > (cf. propriété 1).

Grâce à ces relations d'ordre, les propriétés de séquence et de temps des flux synchrones peuvent être manipulées. Ainsi dans un flux synchrone, on peut obtenir, pour chaque tranche, la tranche qui la précède et la tranche qui la suit.

Définition 9 *Opérateurs précédent (prec) et suivant (suiv)*

D'après la propriété 2, les tranches synchrones d'un flux synchrone SF sont totalement ordonnées par les relations < et >. Ainsi, on peut définir :

- *$\forall TS_1 \in FS$, $\exists TS_2 \in FS$ on appelle prec(TS_1) la tranche synchrone TS_2 / $TS_2<TS_1$ et $\forall TS_3 \in FS$, on a : $TS_3<TS_2$ ou $TS_3>TS_1$*

- *$\forall TS_1 \in FS, \exists TS_4 \in FS$ on appelle suiv(TS_1) la tranche synchrone TS_4 / $TS_4 > TS_1$ et $\forall TS_3 \in FS$, on a : $TS_3 > TS_4$ ou $TS_3 < TS_1$*

A l'aide de ces opérateurs et de la propriété 2, on peut définir des intervalles temporels entre des tranches synchrones. L'utilisation d'une horloge physique locale impose que ces intervalles soient définis entre des tranches dont les données proviennent du même site.

Définition 10 *Intervalles de Temps*

On peut exprimer des intervalles de temps entre les tranches synchrones d'un flux synchrone FS de la façon suivante : $\forall TS_1, TS_2 \in FS$, on définit $<TS_1, TS_2> = |etiquetteTemporelle(TS_1) - etiquetteTemporelle(TS_2)|$.

Grâce à la constante de temps k définie par l'hypothèse 1, ces intervalles de temps ont la même mesure sur tous les sites d'une AMD.

La propriété temporelle des flux synchrones introduit un comportement temporel explicite que nous nous proposons d'étudier.

3 Les Contraintes Temporelles

Précédemment, nous avons vu que tous les médias et, de façon générale, toutes les données existent sous la forme de flux synchrones et possèdent des propriétés communes. La question qui se pose alors est de savoir si tous les flux de données vont avoir le même comportement dans les AMD.

Classiquement les travaux sur les AMD (cf. chapitre 1) distinguent les médias continus et les médias discrets. La distinction est alors basée sur la nature des données. Or, le modèle Korrontea propose une structure commune pour tous les médias et rend cette distinction obsolète.

Si l'on met de côté la nature des données, il ne reste plus qu'à étudier la propriété de séquence et les relations temporelles des flux de données. En effet, on pourrait être tenté de distinguer les flux selon la régularité des données, ce qui nous amènerait à définir des flux réguliers et des flux irréguliers. Cette solution n'est pas satisfaisante car elle nous conduirait à considérer de la même façon un flux régulier d'images où chacune serait séparée, par exemple, d'un temps de l'ordre de la minute et un autre flux régulier où les images seraient séparées d'un temps de l'ordre de la seconde. Même si ces flux sont tous les deux réguliers, ils sont tout de même différents car ils génèrent des contraintes d'ordre temporel très différentes. Nous laisserons donc de côté ce critère.

La remarque précédente est cependant intéressante car il apparaît que la différence entre les deux flux pris pour exemple se fait sur les relations temporelles intra-flux. Certains travaux distinguent les médias par rapport à ce critère, c'est le cas de Blakowski *et al.* [BLA96] qui distinguent les médias avec et sans dépendance temporelle. En ce qui nous concerne, les flux de données que nous définissons possèdent tous des dépendances temporelles. Nous ne pouvons donc pas adhérer à cette classification. Nous nous devons donc de distinguer les flux de données par rapport à un critère temporel, nous parlons alors de contraintes temporelles. Elles définissent le comportement temporel des flux de données. Ceci s'avère important pour la manipulation des flux dans les AMD. En fonction de ces contraintes, les flux seront manipulés de façon différente dans les AMD. Nous le verrons lorsque nous présenterons les politiques de synchronisation ainsi que le modèle de composants Osagaia. Nous avons choisi de distinguer les flux à contrainte temporelle faible et les flux à contrainte temporelle forte. Par nature, nous considérons que les flux supportant des médias continus sont nécessairement à contrainte temporelle forte (les études sur la perception humaine des médias appuient ce point de vue [GHI98], [STE96]). D'un autre côté, les flux qui supportent des médias discrets peuvent être à contrainte temporelle forte ou faible suivant leurs relations temporelles.

Propriété 4 *Contrainte Temporelle Forte*

Un flux synchrone est dit à contrainte temporelle forte si et seulement si, θ étant défini pour l'application, $\forall TS \in FS, <TS, suiv(TS)> \leq \theta$.

Nous introduisons ici un paramètre θ qui représente le temps limite entre deux tranches synchrones successives, au-delà duquel un flux est considéré comme à contrainte temporelle faible. Ce temps ne peut pas être défini de manière générale car il dépend des médias et des donnés utilisés ou susceptibles de l'être dans les AMD. Par conséquent, nous ne donnons pas une valeur particulière pour θ. C'est un paramètre des AMD qui doit être défini lors de leur conception.

Lorsque le temps entre deux tranches successives peut dépasser le paramètre θ, alors le flux correspondant est considéré comme à contrainte temporelle faible.

Propriété 5 *Contrainte Temporelle Faible*

Un flux synchrone est dit à contrainte temporelle faible si et seulement si, θ étant défini pour l'application, $\exists TS \in FS / <TS, suiv(TS)> > \theta$.

Bien entendu, cette classification est applicable aussi bien aux flux primitifs qu'aux flux composés.

Cette classification est utilisée pour la manipulation des flux de données. Les politiques de synchronisation que nous allons définir se basent sur ces propriétés. Les traitements et le transport des données dans les AMD sont également définis en fonction de ces contraintes.

4 Les Politiques de Synchronisation

Nous avons vu tout au long de ce mémoire que la synchronisation est une propriété essentielle des systèmes multimédias. Les médias désynchronisés paraissent artificiels, étranges et quelquefois dénués de sens [GHI98], [STE96]. Les graphes fonctionnels permettent de spécifier les relations de synchronisation inter-flux, c'est donc une spécification des AMD que l'on se doit de fournir à l'implémentation. Ces spécifications signifient que les données liées par des liens de synchronisation doivent être gardés synchrones pendant leur transport et leur traitement dans les AMD.

Les relations de synchronisation de type intra-flux ne sont pas clairement précisées lors des phases de spécification des applications. Cependant, les étiquettes temporelles des flux synchrones permettent de les définir en tant que propriétés des flux de données. Pour certains médias, elles se révèlent très importantes.

Les politiques que nous proposons permettent de conserver ces relations de synchronisation en particulier lors de la diffusion des données. Nous allons maintenant détailler les méthodes utilisées pour conserver ces relations.

4.1 La Synchronisation Intra-Flux

Les relations de synchronisation intra-flux correspondent en fait au débit du flux. Elles décrivent les relations temporelles entre les données qui composent un flux. Par exemple, une vidéo avec un débit de 25 images par seconde impose de diffuser en moyenne une image toutes les 40 millisecondes. Ces relations ne sont pas strictes, des tolérances peuvent être acceptées [GHI98], [STE96], [WEI98]. Le modèle Korrontea possède les propriétés nécessaires pour assurer ce type de relations lors de la restitution des flux. En effet, la propriété de séquence et les relations temporelles permettent d'assurer ce type de synchronisation. La restitution des flux doit être définie au moment de la conception des AMD, elle est implémentée dans des composants dédiés à cette tâche.

Pour chaque flux synchrone, on peut définir les relations de synchronisation intra-flux en utilisant le numéro de séquence des unités d'information et les étiquettes

temporelles des tranches synchrones. Les relations d'ordre total strict permettent d'ordonner toutes les unités d'information des flux et donc tous leurs échantillons. De plus, les contraintes temporelles déterminent le type de synchronisation intra-flux du flux considéré en relation avec le paramètre θ (cf. propriété 4 et 5).

Les informations traduites par le numéro de séquence, les étiquettes temporelles et les contraintes temporelles sont définies lors de la création des flux synchrones par une source localisée (cf. définition 1). Il se peut que ces informations soient modifiées par les traitements et les manipulations des flux dans les AMD. Nous prévoyons de mettre à jour ces dernières à l'aide de services attachés aux composants afin de conserver des informations pertinentes pour toutes les entités des AMD susceptibles de manipuler des flux. Ces services sont décrits dans le prochain chapitre qui traite du modèle de composants [BOX05].

Il n'est pas nécessaire de conserver strictement ces relations pendant le transport des flux dans les AMD car en fait il n'est pas très difficile de les retrouver lors de la restitution. Le modèle est conçu avec cet objectif. Par contre, il est plus difficile voire impossible de récupérer les relations de synchronisation inter-flux lors de la restitution si on ne les a pas conservées en amont. C'est pourquoi nous avons défini les flux synchrones composés. Les politiques présentées par la suite permettent de définir les flux composés et donc de figer ces relations de synchronisation.

4.2 La Synchronisation Inter-Flux

La synchronisation inter-flux correspond aux relations temporelles qui peuvent exister entre les données de plusieurs médias. Par exemple, les relations qui lient l'audio et l'image dans une vidéo sont de ce type. Les choix opérés précédemment imposent que ces relations puissent uniquement être conservées entre des flux capturés ou créés sur le même site d'une AMD car notre propos n'est pas de créer de la synchronisation, mai de conserver celle qui existe initialement.

Afin de mener à bien cette tâche, il est capital de pouvoir établir des relations temporelles entre les tranches synchrones qui appartiennent à différents flux. Dans ce but, nous définissons les opérateurs dateMinimale et dateMaximale.

Définition 11 *Opérateurs dateMinimale et dateMaximale*

On définit des opérateurs qui, à partir d'un ensemble de tranches synchrones E, retournent les étiquettes temporelles les plus petites et les plus grandes. Soit $E=\{e_1, e_2, \ldots, e_n\}$ avec $\forall e_i, \forall e_j \in E$, $site(e_i)=site(e_j)$, on a :

- $dateMinimale(E) = min_{e_i \in E}(etiquetteTemporelle(e_i))$

- $dateMaximale(E) = max_{e_i \in E}(etiquetteTemporelle(e_i))$

Plusieurs tranches synchrones dont les données proviennent du même site et appartenant à des flux différents peuvent avoir la même étiquette temporelle. Par conséquent, plusieurs tranches synchrones de E peuvent avoir comme étiquette temporelle dateMinimale(E) (respectivement dateMaximale(E)).

Les opérateurs dateMinimale et dataMaximale sont définis pour être appliqués sur un ensemble de tranches synchrones. Les flux synchrones sont des ensembles a priori infinis de tranches. Lorsque l'on appliquera ces opérateurs sur des flux, ils ne disposeront pas de la totalité des tranches synchrones puisque ces dernières arrivent séparément et de façon continue. Les opérateurs que nous définissons s'appliquent donc sur les tranches présentes sur chacun des flux à partir d'un instant précis. Dans ce qui suit, le paramètre t désigne l'instant auquel un opérateur examine les tranches synchrones disponibles sur les flux qu'il traite. Ce temps est évidemment relatif aux temps exprimés par les étiquettes des flux à ce moment là.

Dés lors que la synchronisation inter-flux s'applique à un ensemble de flux synchrones, nous définissons des opérateurs basés sur les mêmes principes que dateMinimale et dateMaximale mais qui s'appliquent cette fois-ci sur un ensemble de flux synchrones.

Définition 12 *Opérateur de première occurrence sur un ensemble de flux synchrones*

Soit $GS=\{f_1, f_2, \ldots, f_n\}$ un ensemble de flux synchrones tel que $\forall f_i \in GS$, $site(f_i)=S$. Sur un tel ensemble, on définit un opérateur appelé $procc_{GS}(t)$ de la manière suivante :

- *$procc_{GS}(t)=dateMinimale(E)$ avec $E=\{e_i \in f / etiquetteTemporelle(e_i) \geq t$ avec $f \in GS\}$.*

De la même manière, on définit un opérateur de dernière occurrence sur un ensemble de flux synchrones.

Définition 13 *Opérateur de dernière occurrence sur un ensemble de flux synchrones*

Soit $GS=\{f_1, f_2, \ldots, f_n\}$ un ensemble de flux synchrones tel que $\forall f_i \in GS$, $site(f_i)=S$. Sur un tel ensemble, on définit un opérateur appelé $docc_{GS}(t)$ de la manière suivante :

- *$\forall f \in GS$, on définit l'ensemble $premiereTranche_f(t)$ par : $premiereTranche_f(t)=\{TS / TS \in f / etiquetteTemporelle(TS) \geq t$ et $etiquetteTemporelle(prec(TS))<t\}$. Cet ensemble contient la première tranche synchrone du flux f dont l'étiquette temporelle est supérieure ou égale à t.*
- *$docc_{GS}(t)=dateMaximale(E')$ avec $E' = \bigcup_{f_i \in GS} premiereTranche_{f_i}(t)$*

Les opérateurs d'occurrence permettent, parmi un ensemble de flux synchrones, d'obtenir les dates minimales et maximales des tranches synchrones de cet ensemble. Afin d'illustrer les opérateurs introduits par les définitions 12 et 13, nous avons représenté sur la Figure 31 un ensemble de flux synchrones. Cet ensemble est composé de quatre flux synchrones primitifs ou composés[52] notés f_1, f_2, f_3, f_4. Chaque flux est représenté comme une succession de tranches synchrones ordonnées dans le temps. Les tranches synchrones sont représentées par des rectangles. Nous désignons sur cet ensemble les tranches synchrones dont les dates correspondent à la première et à la dernière occurrence. Chacun des deux opérateurs est appliqué à partir de l'instant t. On considère que les tranches représentées sont les tranches disponibles sur chacun des flux à partir de cet instant.

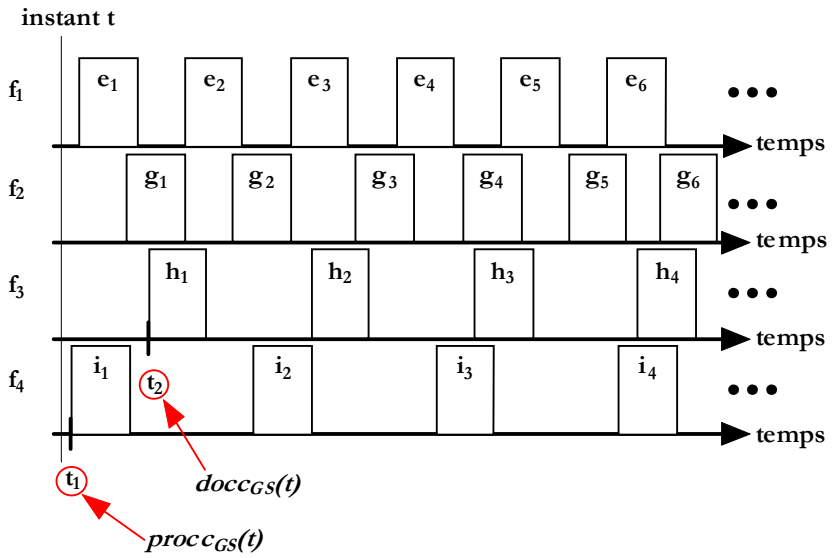


Figure 31 Première et Dernière Occurrences sur un ensemble de Flux

Ces opérateurs possèdent plusieurs propriétés que nous allons présenter.

Propriété 6 *Propriétés des Opérateurs d'Occurrence*

- $doccGS(t) \geq proccGS(t) \geq t$
- *Preuve :* $\exists\, e \in f$ *tel que :*

$etiquetteTemporelle(e) = dateMinimale(E) = min_{e_i \in E}(etiquetteTemporelle(e_i))$. *Nous démontrons que* $e \in premiereTranche_f(t)$. *On sait que* $etiquetteTemporlle(e) \geq t$ *(cf. définition 13).* $prec(e)$ *vérifie* $etiquetteTemporelle(prec(e)) < t$ *car sinon cela voudrait dire que* $prec(e) \in E$ *et dans ce cas on aurait* $dateMinimale(E) = etiquetteTemporelle(prec(e)) < etiquetteTemporelle(e)$ *car* $prec(e) < e$

[52] Le fait que ces flux soient primitifs ou composés ne change rien puisque les opérateurs définis s'appliquent indifféremment sur les deux sortes de flux synchrones.

(cf. définition 9). Ainsi, on a e ∈ premiereTranche$_f$(t), alors docc$_{GS}$(t)=dateMaximale(E') avec $E' = \bigcup_{f_i \in GS} premiereTranche_{f_i}(t)$*, donc e ∈ E' et dateMaximale(E')≥etiquetteTemporelle(e).*

Ainsi, on a docc$_{GS}$(t)≥proccc$_{GS}$(t). De plus, proccc$_{GS}$(t)=dateMinimale({e$_i$ ∈ f / etiquetteTemporelle(e$_i$)≥t})≥t car cet opérateur est défini comme la tranche synchrone ayant l'étiquette temporelle la plus petite supérieure ou égale à t.

Le but de ces politiques est de permettre la composition des flux synchrones et donc de leurs tranches synchrones. Ainsi, à partir de plusieurs flux synchrones (primitifs ou composés) provenant du même site, on pourra constituer un flux composé. Cette composition dépend des contraintes temporelles des flux synchrones utilisés. La propriété suivante va nous aider à distinguer les différentes politiques que nous utiliserons.

Propriété 7 *Propriété d'un ensemble de Flux Synchrones*

- *Un ensemble de flux synchrones GS peut être décomposé en deux sous-ensembles : GS={GS$_{forte}$}∪{GS$_{faible}$}. Ainsi, GS$_{forte}$={f$_i$ ∈ GS / f$_i$ est un flux à contrainte temporelle forte} et GS$_{faible}$={f$_i$ ∈ GS / f$_i$ est un flux à contrainte temporelle faible}.*

Grâce à cette propriété, nous allons définir les méthodes utilisées pour constituer les tranches synchrones des flux composés. En fonction des contraintes temporelles des flux synchrones que nous voulons fusionner, nous définissons trois politiques différentes :

- la première, appelée politique forte, est appliquée lorsque l'ensemble des flux synchrones ne contient que des flux à contrainte temporelle forte ;
- la seconde, appelée politique mixte, est appliquée lorsque l'ensemble des flux synchrones contient des flux à contrainte temporelle forte et faible ;
- la dernière, appelée politique faible, est appliquée lorsque l'ensemble des flux synchrones contient uniquement des flux à contrainte temporelle faible.

Réunir plusieurs flux synchrones dans un seul a pour objectif de les lier par une relation de synchronisation inter-flux. Pour ce faire, nous constituons des tranches synchrones qui regroupent les informations de ces différents flux ayant une date proche. Il est évident que la taille de ces tranches doit être aussi petite que possible si l'on veut conserver la fluidité des médias. Toutefois en constituant des tranches synchrones petites, on court le risque qu'elles ne contiennent aucune unité d'information pour certains des flux de données. Ceci est parfaitement acceptable par notre modèle mais présente l'inconvénient de la perte des relations de synchronisation lorsque certains flux ne sont pas représentés. Notre objectif est alors de faire en sorte que les tranches synchrones constituées contiennent des unités d'information pour chacun des flux.

Toutefois, la présence de flux à contrainte temporelle faible sur lesquels le temps séparant deux tranches synchrones n'est pas borné (cf. propriété 5) rend cette approche impossible. Nous avons donc choisi de constituer des tranches synchrones en regroupant au moins une unité d'information de chacun des flux à contrainte temporelle forte et en acceptant qu'il puisse ne pas y en avoir pour les flux à contrainte temporelle faible. Cette solution est justifiée par le fait que la compensation des relations de synchronisation inter-flux est moins cruciale par nature pour les flux à contrainte temporelle faible que pour les autres.

Lorsqu'une fusion sera mise en place entre plusieurs flux synchrones, il faudra tenir compte du fait que les tranches synchrones des flux à fusionner parviennent à l'opérateur qui applique ces politiques par des chemins différents et donc de ce fait subissent des retards différents et a priori imprévisibles.

C'est pourquoi nous couplons nos politiques à des mécanismes de compensation de la gigue. En effet, nous devons prendre en considération le fait que les flux synchrones d'un même ensemble peuvent arriver en retard ou en avance les uns par rapport aux autres. Ces délais temporels entre les flux sont pris en compte afin de ne pas fausser les relations de synchronisation. Ces délais peuvent être dus aux traitements et aux transferts des flux sur le réseau. En effet, il est tout à fait possible qu'un flux subisse un traitement de plus que les autres, ce qui le retarde par rapport à eux. De la même manière un flux peut passer par un site intermédiaire et donc être retardé en raison des temps de transfert. Pour ce faire, nous introduisons un délai maximum α qui représente le temps pendant lequel on va attendre l'arrivée des tranches synchrones sur les différents flux. Ce temps concrétise les délais entre les flux. Nous allons voir dans chaque politique comment ce temps est utilisé.

Nous détaillons maintenant les trois politiques en donnant l'algorithme utilisé dans chaque cas.

4.2.1 Politique Forte

Cette politique est appliquée lorsque l'on souhaite réunir des flux synchrones d'un ensemble GS vérifiant $GS_{forte} \neq \emptyset$ ET $GS_{faible} = \emptyset$ c'est-à-dire ne contenant que des flux synchrones à contrainte temporelle forte.

Le principe de la politique forte est de constituer les tranches synchrones du flux composé résultant en rassemblant toutes les unités d'information des tranches dont les étiquettes temporelles sont comprises entre la première et la dernière occurrence. Cette façon de faire garantit que les tranches synchrones ainsi fabriquées contiendront au moins une unité d'information sur chaque flux de données (cf. défini-

tion 13). Sur un flux synchrone à contrainte temporelle forte, on est sûr de recevoir une tranche dans un délai maximum de $\alpha+\theta$ (cf. propriétés 4 et 5). A partir d'un instant t, on attend jusqu'à disposer de l'ensemble appelé premiereTranche$_f$(t) (cf. définition 13) pour chaque flux de l'ensemble de flux synchrones.

L'algorithme de cette politique est donné sur la Figure 32. Sa première tâche consiste à attendre au moins une tranche synchrone sur chacun des flux. Ensuite, on applique l'opérateur de dernière occurrence sur l'ensemble des tranches disponibles afin de déterminer la valeur temporelle T_{MAX}. Pour compenser les délais susceptibles d'exister entre les flux de l'ensemble, on attend sur chaque flux une tranche qui vérifie etiquetteTemporelle(TS)>T_{MAX}. Une fois tout ceci terminé, on peut constituer la tranche résultante en ajoutant toutes les unités d'information des tranches reçues qui vérfient etiquetteTemporelle(TS)≤T_{MAX}. Enfin, nous assignons à chaque unité d'information de chacun des flux de données des numéros de séquence successifs. L'étiquette temporelle de la tranche résultante sera fournie par l'opérateur de première occurrence appliqué sur les tranches utilisées pour la composer.

répeter

- Attendre jusqu'à avoir une tranche synchrone sur chaque flux synchrone ∈ GS (cette étape permet de disposer de premiereTranche$_f$(t) pour tous les flux f du groupe GS)
- $T_{MAX} \leftarrow$ docc$_{GS}$(t)({tranches synchrones reçues})
- Attendre jusqu'à avoir, sur chaque flux synchrone ∈ GS, une tranche synchrone TS vérifiant etiquetteTemporelle(TS)>T_{MAX}
- Constituer la tranche synchrone composée avec :
  - toutes les unités d'information des tranches synchrones reçues vérifiant etiquetteTemporelle(TS)≤T_{MAX} en les dotant de numéros de séquence consécutifs commençant à 1 pour chaque flux
  - une étiquette temporelle= procc$_{GS}$(t)({tranches synchrones utilisées})

fin répéter

Figure 32 Algorithme de la Politique Forte

La Figure 33 donne un exemple de l'opérateur politique forte appliqué sur trois flux synchrones à contrainte temporelle forte. Les contraintes des flux f_1, f_2 et f_3 sont respectivement 47 ms, 203 ms et 175 ms. Elles représentent le temps moyen entre deux tranches successives de chaque flux.

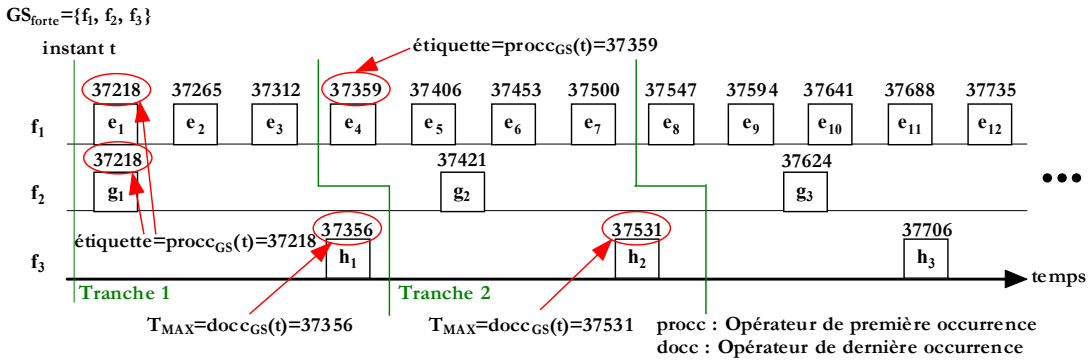


Figure 33 Exemple de Constitution des Tranches par la Politique Forte

Cet algorithme permet d'obtenir des tranches synchrones les plus petites possibles contenant au moins une unité d'information de chacun des flux de données.

4.2.2 Politique Mixte

Cette politique est appliquée lorsque l'on souhaite réunir les flux synchrones d'un ensemble GS vérifiant $GS_{forte} \neq \varnothing$ ET $GS_{faible} \neq \varnothing$ c'est-à-dire contenant des flux synchrones à contrainte temporelle forte et des flux synchrones à contrainte temporelle faible.

Dans la politique mixte, les tranches synchrones du flux composé sont définies en rassemblant toutes les unités d'information des tranches dont l'étiquette temporelle est comprise entre la première occurrence et la dernière occurrence appliquées sur le sous-ensemble GS_{forte}. Ainsi, on garantit que la tranche résultante contiendra au moins une unité d'information sur chaque flux appartenant à GS_{forte} (cf. définition 13). Sur les flux appartenant à GS_{faible}, on pourra avoir des unités d'information ou pas. Dans cette politique, nous utilisons un délai α qui permet de prendre en compte le possible retard de réception des tranches synchrones des flux à contrainte temporelle faible. Après ce délai, on considèrera que toutes les tranches de ces flux qui doivent appartenir à la tranche en cours de constitution sont reçues.

Comparé au précédent, cet algorithme est plus complexe car il doit pouvoir prendre en compte des flux à contrainte temporelle faible sur lesquels il n'est pas possible de savoir s'il y aura ou non des données. Pour ce faire, nous utilisons un principe comparable à celui des sémaphores dans les systèmes d'exploitation. Chaque arrivée d'une tranche synchrone sur l'un des flux de GS_{forte} déclenche un délai α au terme duquel une autorisation de constitution d'une tranche synchrone contenant ces données sera délivrée. Cette autorisation concrétise la prise en compte du délai possible de réception des tranches synchrones de même date sur les autres flux. Bien entendu, toutes les tranches synchrones reçues au cours de ce délai ne seront pas obligatoirement

placées dans la même tranche synchrone que celle qui l'a déclenché. En effet, elles ne le seront que si leur étiquette temporelle vérifie les propriétés adéquates.

L'algorithme qui correspond à la politique mixte est donné sur la Figure 34. La première tâche consiste à initialiser la variable autorisation à 0. Suite à quoi, on attend sur chaque flux à contrainte temporelle forte au moins une tranche synchrone. A chaque arrivée d'une tranche sur l'un de ces flux, on lance un timer avec un délai α. A chaque fin de délai, la variable autorisation est incrémentée d'une unité. T_{MAX} est défini grâce à l'opérateur de dernière occurrence appliqué sur l'ensemble des tranches synchrones reçues des flux à contrainte temporelle forte. On attend ensuite que chaque flux à contrainte temporelle forte possède une tranche synchrone qui vérifie etiquetteTemporelle(TS)>T_{MAX}. Enfin, la tranche est composée à l'aide de toutes les unités d'information des tranches reçues dont l'étiquette temporelle vérifie etiquetteTemporelle(TS)≤T_{MAX}. Les unités d'information sont associées à des numéros de séquence successifs. L'étiquette temporelle de la tranche ainsi formée est définie à l'aide de l'opérateur de première occurrence appliqué sur les tranches synchrones, des flux à contrainte temporelle forte, utilisées. La variable autorisation est décrémentée du nombre de tranches des flux à contrainte temporelle forte utilisées.

- *autorisation* ← 0
répeter
  répéter
    - A chaque arrivée d'une tranche synchrone sur l'un des flux synchrones ∈ GS_{forte} lancer un timer pour une durée α
  jusqu'à avoir une tranche synchrone sur chaque flux synchrone ∈ GS_{forte} (cette étape permet de disposer de premiereTranche$_f$(t) pour tous les flux synchrones ∈ GS_{forte})
  - T_{MAX} ← $docc_{GSforte}$(t)({tranches synchrones des flux à contrainte forte reçues})
  - Attendre jusqu'à avoir, sur chaque flux synchrone ∈ GS_{forte}, une tranche synchrone TS vérifiant etiquetteTemporelle(TS)>T_{MAX}
  - Attendre que *autorisation* = nombre de tranche synchrone des flux à contrainte forte dont l'étiquette est ≤ T_{MAX}
  - Constituer la tranche synchrone composée avec :
    - toutes les unités d'information des tranches synchrones reçues (tant sur les flux à contrainte temporelle forte que faible) vérifiant etiquetteTemporelle(TS)≤T_{MAX} en les dotant de numéros de séquence consécutifs commençant à 1 pour chaque flux
    - une étiquette temporelle= $procc_{GSforte}$(t)({tranches synchrones des flux à contrainte forte utilisées})
  - *autorisation* ← *autorisation* - nombre de tranches synchrones des flux à contrainte forte utilisées
fin répéter

A chaque fin de timer, on fait : *autorisation* ← *autorisation* + 1

Figure 34 Algorithme de la Politique Mixte

La Figure 35 donne un exemple de l'opérateur politique mixte appliqué sur un ensemble de flux synchrones dont trois sont à contrainte temporelle forte et deux sont à contrainte temporelle faible. Les contraintes temporelles des flux f_1, f_2 et f_3 sont res-

pectivement 400 ms, 200 ms et 350 ms. Les paramètres utilisés pour cet exemple sont θ=1000 ms et α=500 ms.

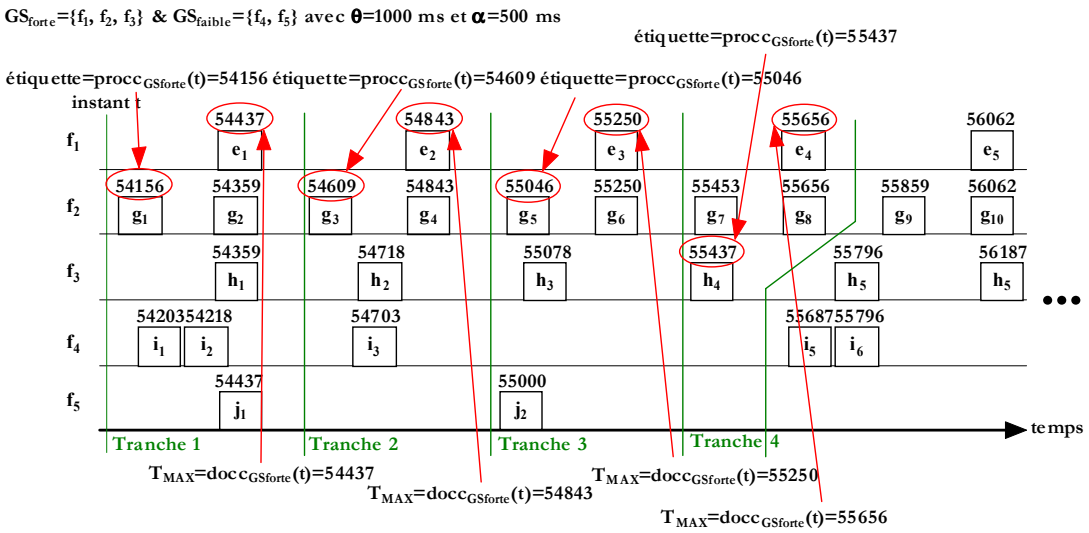


Figure 35 Exemple de Constitution des Tranches par la Politique Mixte

Comme le précédent, cet algorithme permet d'obtenir des tranches synchrones les plus petites possibles contenant au moins une unité d'information de chacun des flux de données à contrainte temporelle forte. Les unités d'information des tranches synchrones des flux à contrainte temporelle faible dont l'étiquette temporelle est inférieure ou égale à T_{MAX} sont associées à la tranche synchrone constituée, il peut ne pas y en avoir.

4.2.3 Politique Faible

Cette politique est appliquée lorsque l'on souhaite réunir des flux synchrones d'un ensemble GS vérifiant $GS_{forte}=\varnothing$ ET $GS_{faible}\neq\varnothing$ c'est-à-dire ne contenant que des flux à contrainte temporelle faible.

La politique faible consiste à composer les tranches résultantes en rassemblant toutes les unités d'information des tranches reçues dont l'étiquette temporelle est comprise entre la première occurrence et la première occurrence + θ. Ainsi, on garantit que la tranche résultante contiendra au moins une unité d'information (cf. définition 12). Dans ce cas, on attend la première tranche synchrone sur l'un des flux, ainsi on pourra utiliser l'opérateur première occurrence afin de définir la tranche correspondante. Nous utilisons également un délai α à la fin duquel on considère que toutes les tranches synchrones de même étiquette temporelle sont reçues.

Dans cette politique, nous constituons des tranches synchrones contenant toutes les unités d'information des flux synchrones de GS dont les étiquettes temporelles

sont comprises dans une fenêtre de longueur θ après l'arrivée de la première d'entre elles. L'utilisation de θ qui constitue la frontière entre les contraintes intra-flux considérées comme forte et faible fait que nous considérons comme synchrones deux unités d'information produites dans un tel intervalle de temps.

Comme dans l'algorithme précédent, nous utilisons un mécanisme d'autorisations de constitution de tranches synchrones basé sur le déclenchement d'un délai $\alpha+\theta$ à chaque arrivée d'une tranche synchrone sur l'un des flux de GS. Ce délai correspond à la fenêtre choisie (θ) augmenté du retard possible d'occurrence (α). Toute tranche reçue dans ce délai peut donc être candidate à rentrer dans la même tranche synchrone que celle qui l'a déclenché. Elle ne sera admise que si son étiquette temporelle vérifie les propriétés adéquates.

L'algorithme de la politique faible est détaillé sur la Figure 36. En premier lieu, la variable autorisation est initialisée à 0. Dés qu'une tranche est disponible sur l'un des flux, un timer est lancé avec un délai $\alpha+\theta$. Lorsque la variable autorisation est strictement positive, on peut commencer à composer la tranche synchrone en ajoutant toutes les unités d'information des tranches reçues qui vérifient etiquetteTemporelle(TS)$\leq$procc$_{GS}$(t)+θ. Comme dans les autres politiques, les unités d'information sont associées à un numéro de séquence. L'étiquette temporelle de la tranche ainsi formée est égale à première occurrence appliquée sur les tranches utilisées. Enfin, la variable autorisation est décrémentée du nombre de tranches utilisé.

- *autorisation* ← 0
répeter
  - A chaque arrivée d'une tranche synchrone sur l'un des flux synchrones ∈ GS lancer un timer pour une durée $\alpha+\theta$
  si *autorisation* > 0
  - Constituer la tranche synchrone composée avec :
    - toutes les unités d'information des tranches synchrones reçues vérifiant etiquetteTemporelle(TS)$\leq$procc$_{GS}$({tranches synchrones reçues})+θ en les dotant de numéros de séquence consécutifs commençant à 1 pour chaque flux
    - une étiquette temporelle= procc$_{GS}$(t)({tranches synchrones utilisées})
  - *autorisation* ← *autorisation* - nombre de tranches synchrones utilisées
  fin si
fin répéter

A chaque fin de timer, on fait : *autorisation* ← *autorisation* + 1

Figure 36 Algorithme de la Politique Faible

La Figure 37 donne un exemple de l'opérateur politique faible appliqué sur un ensemble de flux synchrones à contrainte temporelle faible. Les paramètres utilisés pour cet exemple sont θ=1000 ms et α=500 ms.

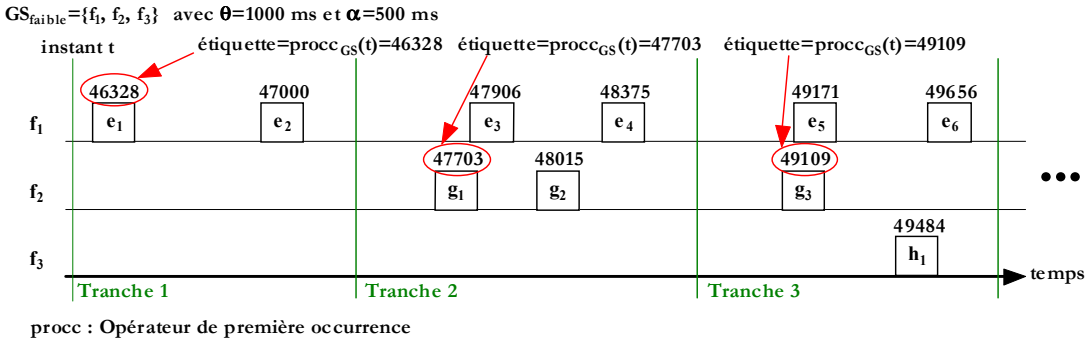


Figure 37 Exemple de constitution des Tranches par la Politique Faible

Cet algorithme permet d'obtenir les tranches synchrones les plus petites possibles contenant au moins une unité d'information de l'un des flux synchrones de GS. La « taille » de la tranche synchrone est limitée à θ qui constitue le seuil de granularité séparant les flux à contrainte forte et ceux à contrainte faible.

4.3 Simulation des Politiques de Synchronisation

Afin de valider les politiques de synchronisation introduites par Korrontea, nous avons développé un simulateur. Ce simulateur permet d'appliquer l'une des trois politiques présentées sur des flux générés selon leurs contraintes temporelles. Il permet de générer autant de flux que l'on souhaite et transmet les tranches synchrones issues de ces flux vers l'opérateur de synchronisation pour créer un flux synchrone composé. Ce simulateur permet de choisir les propriétés des flux synchrones. Les tranches synchrones des flux sont représentées par des objets ne contenant que les étiquettes temporelles qui sont attribuées lors de la création. En effet, seule cette information est nécessaire car le seul but de ce prototype est de valider la constitution des tranches synchrones en sortie de l'opérateur. Ces résultats sont restitués sur la fenêtre principale du simulateur et stockés dans un fichier texte afin de pouvoir les analyser plus en détails.

Pour créer des flux synchrones, le simulateur utilise plusieurs paramètres. Le premier permet de définir la valeur de θ afin de différentier les flux à contrainte temporelle forte et ceux à contrainte faible. Un paramètre α permet de donner le délai maximal d'attente des tranches synchrones en entrée de l'opérateur chargé d'appliquer les politiques de synchronisation. Ce paramètre est utilisé par les politiques mixtes et faibles qui traitent des flux à contrainte temporelle faible. En effet, on a vu que les flux synchrones pouvaient arriver en retard les uns par rapport aux autres et donc qu'il est nécessaire de considérer ces retards afin de fournir une synchronisation inter-flux correcte. Enfin, les deux derniers paramètres permettent de définir le nombre de flux

à contrainte temporelle forte et le nombre de flux à contrainte temporelle faible que l'on veut fusionner.

Chaque flux est créé avec trois paramètres. Le premier paramètre est un entier qui désigne le numéro du flux afin que ce dernier puisse être créé puis manipulé par le simulateur. Le deuxième représente la relation de synchronisation intra-flux entre les tranches du flux. Pour les flux à contrainte temporelle forte, il définit le temps entre deux tranches successives. Pour les flux à contrainte temporelle faible, il définit le temps maximal entre deux tranches. Ce temps est variable et le simulateur le définit de manière aléatoire. Enfin, il est possible d'introduire à l'aide du dernier paramètre un temps représentant le retard maximal d'arrivée des tranches à l'opérateur de synchronisation. Ainsi, les retards entre les flux peuvent être pris en compte.

Le simulateur comporte un ensemble de fenêtres graphiques représenté sur la Figure 38. La fenêtre principale du simulateur donne la constitution des tranches créées. Chaque tranche indique, pour chacun des flux, le nombre de tranches utilisé lors de sa constitution. Les tranches utilisées sont repérées à l'aide de leur étiquette temporelle. Le temps T_{MAX} défini pour former la tranche est également indiqué (sauf pour la politique faible puisqu'on ne l'utilise pas). Les flux désignés par une lettre majuscule sont des flux à contrainte temporelle forte et les flux désignés par une minuscule sont ceux à contrainte faible. Pour une tranche donnée, il est possible de voir apparaître sur les flux à contrainte temporelle faible la description suivante : {f0=} (cf. Figure 38). Cette description signifie que des tranches pour ce flux sont disponibles mais que pour l'instant elles ne remplissent pas les critères pour être intégrées à la tranche en cours de constitution (tranches arrivées trop tôt).

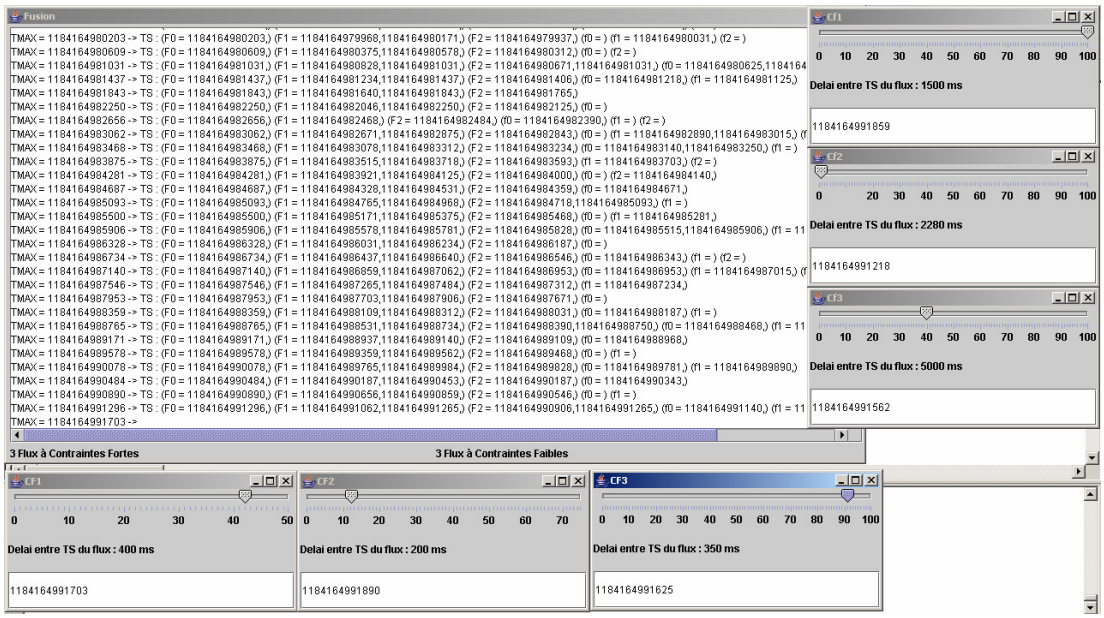


Figure 38 Interfaces Graphiques du simulateur de Politiques

Les autres fenêtres que l'on voit apparaître en bas et à droite de la fenêtre principale (cf. Figure 38) sont celles des flux. Chacune d'entre elles est dédiée à un flux synchrone particulier. Elle affiche la relation temporelle qui lie les tranches du flux ainsi que les étiquettes temporelles associées aux tranches créées. Une réglette permet de faire varier le retard maximum apporté à ce flux pendant l'utilisation du simulateur.

Ce simulateur nous a permis de procéder à une batterie de tests afin de vérifier que les politiques de synchronisation permettent de fournir le résultat attendu. Nous avons testé différentes valeurs pour les paramètres θ et α. Son principal intérêt réside dans le fait que l'on peut retarder l'arrivée des tranches synchrones de certains flux. Ainsi, on a pu vérifier que ces retards étaient correctement pris en compte par l'opérateur proposé et donc que malgré ces retards les tranches étaient correctement formées. Les exemples que nous avons donnés pour chaque politique dans le paragraphe précédent ont été réalisés à l'aide de ce simulateur. Nous donnerons les résultats concrets dans la dernière partie de ce mémoire qui concerne l'implémentation du prototype et les résultats d'expérimentation.

5 Synthèse

Avant de refermer ce chapitre, nous récapitulons les principaux aspects du modèle Korrontea. Nous avons représenté ce modèle à l'aide d'un diagramme de classes UML représenté sur la Figure 39. L'abstraction de base du modèle est le flux de données qui permet de modéliser l'ensemble des données susceptibles d'exister dans les AMD. Ces flux sont composés d'une séquence d'unités d'information.

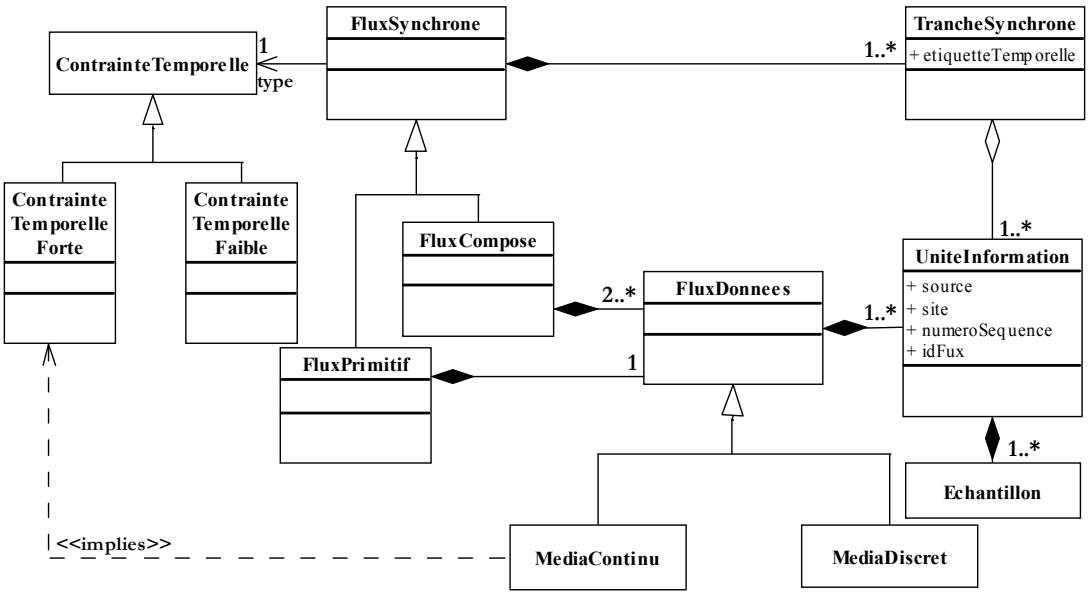


Figure 39 Modèle Conceptuel Korrontea

Chaque unité contient un ou plusieurs échantillons, une référence à la source localisée qui les a produite, un numéro de séquence qui permet de déterminer l'ordre des unités et un identifiant du flux de données auxquelles elles appartiennent. Ces unités sont rassemblées dans des tranches synchrones afin d'ajouter la propriété temporelle propre à certains médias. En procédant de la sorte, nous rendons temporelles toutes les données des AMD. Cette propriété est exprimée à l'aide d'une étiquette temporelle délivrée par une horloge physique locale. Les AMD manipuleront des structures que nous appelons flux synchrones et qui sont une succession ordonnée par le temps de tranches synchrones. Les flux synchrones peuvent prendre deux aspects différents. On les appelle primitifs, lorsqu'ils transportent des unités d'information appartenant à un et un seul flux de données. On les appelle composés, lorsqu'ils transportent des unités d'information issues de plusieurs flux de données provenant du même site. Cette solution, introduite ici, permet de conserver les liens de synchronisation inter-flux définis sur les graphes fonctionnels. Nous répondons de la sorte aux besoins mis en évidence lors de la spécification fonctionnelle en définissant des politiques de synchronisation qui vont permettre de constituer les flux composés. Ces politiques se basent sur le comportement temporel des flux synchrones. En effet, nous pensons que cet aspect constitue le critère essentiel de classification des flux pour les raisons évoquées tout au long de ce chapitre et plus amplement détaillées dans le chapitre 1. Ainsi, nous distinguons les flux à contrainte temporelle forte et les flux à contrainte temporelle faible. De par leur nature, nous pensons que les médias continus sont nécessairement à contrainte temporelle forte. Nous avons modélisé sur la Figure 39 cette affirmation à l'aide de la dépendance <<implies>> [OMG03]. Par contre, toutes les autres données peuvent être, selon le cas, à contrainte forte ou faible. Nous allons voir dans le chapi-

tre suivant que ces comportements temporels vont être prépondérants pour la manipulation des flux dans l'architecture logicielle.

Les modèles de données permettent de décrire de façon abstraite la façon dont sont représentées les données dans un applicatif. Ils sont très présents dans les systèmes d'information et les systèmes de gestion de base de données. Dans le cas des applications multimédias distribuées, nous n'avons pas rencontré beaucoup de modèles de données à l'exception des travaux que nous avons présenté dans le chapitre 1. Toutefois dans ces travaux, l'accent n'est pas mis sur la notion de modélisation des données. Les modèles décrits servent plutôt à assurer des présentations de documents multimédias en synchronisme. Les spécifications avancées concernent uniquement la diffusion des données (cf. chapitre 1). Cette notion de modèle liée au multimédia est cependant très présente dans les travaux qui s'intéressent aux bases de données multimédias. Par exemple, dans [LEE99] les auteurs présentent un modèle de données pour spécifier des présentations multimédias sous la forme de graphes directes et acycliques. Ils introduisent à l'occasion des langages permettant d'effectuer des requêtes sur ces graphes. Ces requêtes sont basées sur le contenu des documents multimédias utilisés dans les modèles de présentation. Dans [LIT93], les auteurs se sont attachés à modéliser les données multimédias dépendantes du temps. Ainsi, ce modèle est destiné à être utilisé dans les systèmes d'information multimédia. [GIB94] donne l'aperçu d'un modèle de données qui adresse la représentation des médias basés sur le temps. Les auteurs définissent les concepts d'objets média, d'éléments de média et de flux temporels. Ce modèle est destiné à l'implémentation de systèmes de base de données capables de supporter des médias basés sur le temps.

Même si nous n'avons pas axé Korrontea sur cet aspect de présentation des données, nous ne l'excluons pas de notre approche car le modèle proposé permet de définir aisément des modèles de présentation utilisables lors de la restitution des données. Le point fort de notre approche se retrouve dans le fait que nous proposons de fixer dynamiquement les propriétés des données et non de les définir a priori. Ainsi, nous amenons une grande flexibilité, nous reviendrons sur ce point dans le paragraphe suivant.

A travers la définition d'un tel modèle, notre objectif est double. Dans un premier temps nous l'avons introduit afin de répondre à une spécification fonctionnelle donnée par les graphes qui est celle de la synchronisation inter-flux des données dans les AMD. Nous nous intéressons à la conception et au développement des AMD basés sur une architecture logicielle flexible. Le but étant de posséder des architectures adaptables dynamiquement dans un souci de gestion de la QdS des AMD. Dans un second temps, nous pensons qu'un fort degré d'intégration des données dans les AMD

est un premier pas vers une solution à ce type de problématique. En effet, dés lors que la gestion des données est effectuée de manière uniforme, un bon nombre de problèmes liés au caractère particulier des données multimédias peut être résolu. C'est le cas en particulier de la manipulation des différents types de données tout en considérant leurs propriétés respectives. Dans ce sens, nous avons choisi une structure de données possédant des propriétés de séquence et de relations temporelles. Ainsi, toutes les données posséderont ces propriétés même si pour certaines elles ne sont pas premières. De plus, les entités logicielles de l'architecture que nous définissons manipuleront toutes les données en considérant ces propriétés de la même façon sans avoir à se préoccuper de leur type. Ainsi, des liens sémantiques pourront être définis entre des données de différents types et c'est là que nous rejoignons le premier objectif. Tout ceci prendra tout son sens dans le chapitre suivant dans lequel nous montrerons l'aspect central de ce modèle de flux dans notre approche.

Afin de valider ce modèle et plus particulièrement les politiques de synchronisation qu'il définit, nous avons développé un simulateur dont le but est de tester les politiques proposées. Nous avons pu ainsi tester les cas critiques et prendre en considération les situations susceptibles de se produire dans des cas réels d'utilisation comme par exemple les retards des flux les uns par rapport aux autres. Les résultats sont probants puisque la constitution des tranches synchrones des flux composés est réalisée comme nous le voulions c'est-à-dire en rassemblant les unités d'information des flux dont les dates sont proches. Lorsque certains flux sont en retard, les temps α et θ permettent de compenser ces retards. Nous donnons dans la dernière partie de ce mémoire de plus amples détails sur les expérimentations à l'aide du simulateur.

Le modèle proposé est donc à tout point de vue intéressant car il offre une grande flexibilité tout en étant facile à mettre en œuvre. Bien au-delà de nos préoccupations, nous pensons que ce modèle peut être réutilisé dans un bon nombre d'applications manipulant des médias où des traitements doivent être mis en œuvre sans en perdre les propriétés de synchronisation. Par exemple, l'adaptation de contenus est une approche très utilisée dans l'informatique pervasive dans laquelle les données multimédias peuvent être transmises vers des périphériques très divers. En raison des capacités disparates des équipements proposés dans de tels réseaux, il est souvent utile d'adapter le contenu des médias à transmettre vers les différentes sources. En effet, on n'utilisera pas des vidéos de la même taille sur un ordinateur portable ou sur un assistant personnel. Ceci dit, de telles adaptations nécessitent l'intégration de traitements spécialisés en amont de la diffusion. Toutefois, ces traitements, bien que souvent simples, ne doivent pas compromettre la sémantique des données transmises par

perte des relations de synchronisation. Nous pensons que dans ce genre de problématique, un modèle comme Korrontea prend tout son intérêt.

Maintenant que nous avons défini les modes de définition et de gestion des données dans les AMD, nous allons pouvoir aborder le dernier point de notre problématique qui est la définition d'une architecture logicielle complète utilisable dans la conception et le développement des AMD. Cette définition va se faire par le biais d'un modèle de composants logiciels pour les raisons évoquées dans les chapitres 3 et 4. Cette architecture va se baser naturellement sur les spécifications fonctionnelles définies par la méthode de conception (cf. chapitre 4) dans lesquelles nous avions alors mis en évidence la nécessité de disposer d'au moins deux entités fonctionnelles pour les implémentations. Néanmoins, un nouveau paramètre vient s'ajouter à la liste des spécifications utiles pour le développement du modèle de composants : le modèle de flux de données Korrontea. En effet, les structures introduites dans ce chapitre vont devoir être manipulables par les entités constituant l'AMD si nous voulons en assurer un fonctionnement efficace. Ainsi, nous allons voir dans le prochain chapitre les implications de la méthode de conception et du modèle Korrontea dans la définition du modèle de composants. Nous abordons cette tâche en proposant un graphe, directement dérivé des graphes de la méthode de conception, qui intègre ces nouvelles informations.

Chapitre 6 – Osagaia, un Modèle de Composants Multimédias

« La composition doit avoir ses lois, ou elle ne serait qu'une fantaisie, qu'un caprice ; or, sans m'occuper de ce qui concerne la peinture, la sculpture et la musique (bien qu'il soit possible, me semble t-il, de définir les règles qui doivent s'imposer dans les compositions des musiciens, des sculpteurs et des peintres), s'il s'agit des arts appliqués à l'architecture, aux diverses industries, il est évident que la composition doit tenir compte de deux éléments, de la matière mise en œuvre et des procédés qui peuvent lui être appliqués. »

Histoire d'un dessinateur – Comment on apprend à dessiner, Eugène-Emmanuel Viollet Le Duc, 1880

Le chapitre précédent a introduit le modèle Korrontea qui permet de donner une description des données dans les AMD. Grâce à lui nous sommes capables de savoir comment elles vont pouvoir être manipulées et traitées. La prochaine étape de ces travaux consiste à définir les entités qui vont permettre l'implémentation des AMD. L'utilisation de deux entités fonctionnelles est justifiée par la méthode de conception. La première doit être utilisée pour l'implémentation des rôles atomiques logiciels décrits sur les graphes fonctionnels tandis que la seconde doit permettre le transport des données entre ces unités d'implémentation. Cependant, lorsque ces justifications ont été introduites, le modèle Korrontea n'était pas encore défini. La première tâche de ce chapitre va être d'étudier l'impact de ce modèle sur la définition des entités fonctionnelles et globalement sur la définition du modèle de composants.

Les graphes fonctionnels fournissent une approche intéressante car ils permettent de définir les configurations des AMD. Ils sont manipulables par la plate-forme afin qu'elle puisse procéder à des reconfigurations de la partie applicative. Le modèle de composants proposé est directement dérivé de ces graphes. Ainsi, les modifications appliquées par la plate-forme ont une répercussion directe sur l'architecture logicielle de l'application. La définition d'entités manipulables représentant les arcs et les nœuds des graphes est donc une nécessité dés lors qu'elle va permettre d'amener de la flexibilité dans l'implémentation des AMD. Cette architecture consiste donc en une connexion de composants et de connecteurs. Elle ressemble fortement au style « pipes & filters » qui consiste à implémenter des traitements asynchrones reliés entre eux par des flux de données [GAN94], [GAR94]. Cet asynchronisme permet une synchronisation implicite des traitements [LEE94]. L'utilisation des flux de données permet de les

alimenter de manière continue. Ainsi, les tâches implémentées sont exécutées lorsqu'un nombre suffisant de données est disponible. Ce type d'approche possède également des points communs avec les architectures guidées par les données [CPU01]. Les traitements et les transferts d'informations sont déclenchés par les données. L'utilisation de flux de données implique qu'ils soient réalisés en parallèle et de façon continue.

Nous commençons par aborder les implications de la méthode ainsi que celles amenées par l'intégration du modèle Korrontea dans les spécifications fonctionnelles des AMD.

1 Introduction

Le modèle Korrontea (cf. chapitre 5) propose de manipuler les données des AMD sous la forme de flux synchrones primitifs ou composés. Nous avons défini une composition verticale des données à l'aide des flux composés afin de pouvoir conserver les relations de synchronisation inter-flux.

Les flux de données sont décrits sur les graphes à l'aide des arcs. Ils permettent de spécifier les échanges d'information entre les rôles. Ces graphes introduisent les spécifications fonctionnelles des AMD. La première tâche qui incombe est de dériver de cette méthode de conception un modèle de composants pour l'implémentation des AMD en concordance avec ces spécifications. Ces graphes introduisent trois types de spécifications fonctionnelles auxquelles nous devons répondre : la spécification des rôles, la spécification des échanges de données et les liens de synchronisation inter-flux qui les unissent. La dernière spécification est assurée grâce au modèle Korrontea. Il reste donc à définir les modèles d'implémentation des rôles et des échanges de données. Pour définir ces modèles, il faut tenir compte de Korrontea puisque ces derniers sont destinés à la manipulation et au traitement de données. Ce chapitre débute par la définition de règles de transformation qui vont permettre d'obtenir des graphes de transition où les notions de composants logiciels et de flux synchrones sont définies explicitement. Ces transformations vont révéler des spécifications que le modèle Osagaia[53] devra intégrer. Les transformations, ainsi introduites, sont basées sur un ensemble de règles directement applicables sur les graphes fonctionnels.

Le modèle de composants logiciels Osagaia sa base donc sur la méthode de conception (cf. chapitre 4), sur le modèle Korrontea (cf. chapitre 5) et sur les graphes

[53] Osagaia signifie « le composant logiciel » en langue Basque.

de transition. Osagaia permet de mettre en évidence et de considérer les différents aspects de l'implémentation d'une AMD. On distingue, à ce titre, les aspects fonctionnels et les aspects non-fonctionnels. La partie fonctionnelle s'articule autour de deux entités qui sont le processeur élémentaire, dont le but est d'exécuter la partie métier des AMD c'est-à-dire les composants qui implémentent les rôles atomiques, et le conduit, qui permet de relier chacune des entités du modèle afin de transporter entre elles les flux synchrones. Cette entité n'est autre qu'un connecteur de composants logiciels. De plus, il représentera l'entité distribuée du modèle. La partie non-fonctionnelle définit un ensemble d'opérateurs de flux synchrones dont la nécessité se justifie à l'aide des règles de transformation des graphes fonctionnels. Certains de ces aspects sont également définis au niveau des composants afin de permettre une exécution correcte. Nous mettrons en œuvre la synchronisation des entités du modèle dans une configuration afin de fournir une coordination correcte de ces dernières et un fonctionnement des AMD prévisible. La dernière partie du modèle est dédiée à l'évaluation et à la délivrance d'informations de QdS traduisant les états de fonctionnement des entités fonctionnelles du modèle.

2 Les Graphes de Transition

Ce paragraphe introduit une transformation des graphes fonctionnels afin d'exprimer l'architecture logicielle des AMD à l'aide de composants connectés entre eux par des flux synchrones primitifs ou composés. L'ensemble des règles de transformation permet de produire des graphes de transition et par la même occasion d'introduire de nouvelles entités nécessaires à l'implémentation. Ces entités sont appelées opérateurs afin de les distinguer des entités fonctionnelles puisqu'ils n'ont pas les mêmes objectifs.

Le modèle de composants défini par la suite est dédié à la conception et à l'implémentation des AMD. Les composants doivent permettre l'implémentation des rôles atomiques de type logiciel. Les rôles réalisés de façon matérielle sont laissés de côté, même si les périphériques qu'ils définissent doivent être intégrés à ces applications. Un autre objectif des graphes de transition est d'introduire le modèle Korrontea à travers la représentation des flux synchrones primitifs et composés. Ainsi, les graphes de transition sont composés de composants logiciels interconnectés par des flux synchrones. La Figure 40 résume ce principe en situant les règles de transformation introduites par rapport aux modèles déjà définis.

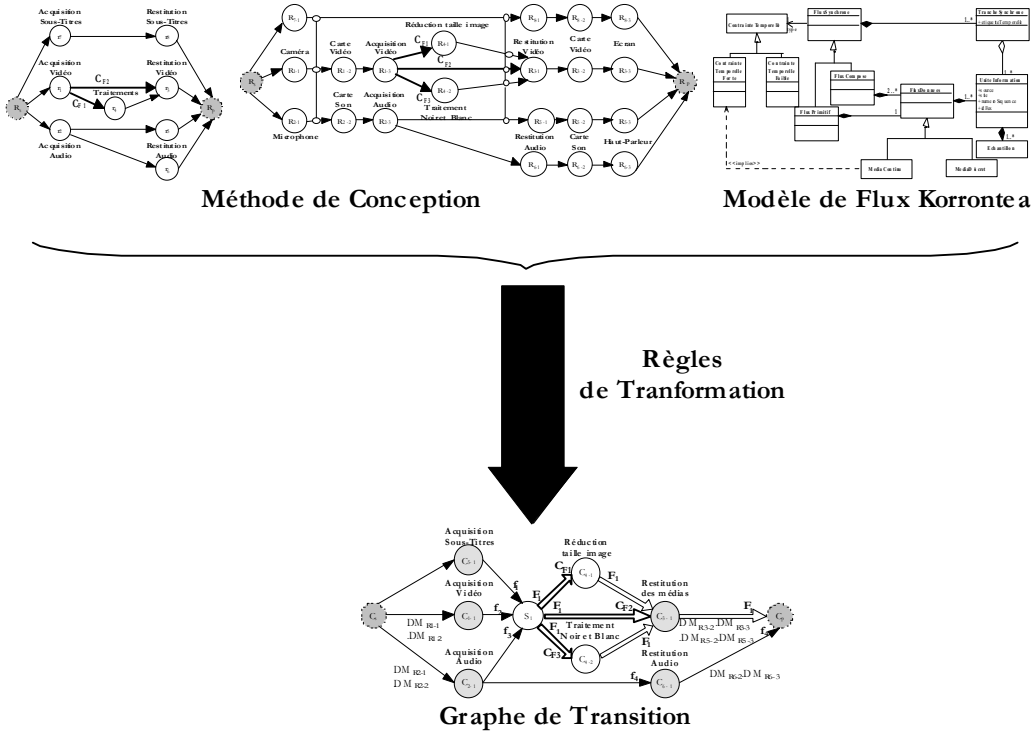


Figure 40 Définition des Graphes de Transition

Ainsi, le modèle de conception et le modèle Korrontea vont nous permettre d'obtenir les graphes de transition.

2.1 Les Dépendances Matérielles

Les moyens logiciels sont destinés à être implémentés par des composants et les moyens matériels décrivent les ressources nécessaires à une application[54]. L'ensemble de ces moyens est recensé par la méthode car ils influent sur la QdS globale d'une AMD. Les propriétés et caractéristiques des ressources matérielles sont à considérer car elles constituent des informations précieuses pour la plate-forme d'exécution. En effet, les fonctionnalités des composants logiciels déployés ne possèderont pas les mêmes caractéristiques sur un ordinateur de bureau et sur un assistant personnel. Si par exemple, on veut diffuser un flux vidéo alors la taille de l'image ne sera pas la même selon que l'on envoie le flux sur l'ordinateur ou sur l'assistant personnel. La connaissance de ce type de rôle permet donc de posséder une liste exhaustive dont on devra disposer pour le déploiement d'une AMD. L'exemple d'AMD introduit dans le

[54] Les moyens auxquels nous faisons allusion ne sont autres que les rôles identifiés en amont par la méthode de conception.

chapitre 3 (cf. § 3) impose au locuteur d'avoir un ordinateur équipé d'une caméra, d'une carte vidéo, d'un micro et d'une carte son.

Les ressources matérielles ne sont pas exclues de la démarche. En effet, il est probable qu'un composant d'acquisition vidéo utilisera les services d'une caméra et indirectement d'une carte vidéo. On dit alors que ce type de composants présente des dépendances vis-à-vis de certaines ressources matérielles. Ils sont donc désignés grâce à la notion de dépendances matérielles. Ces dépendances sont précisées sur les arcs entrants et sortants des composants concernés sur les graphes de transition selon que le flux provient ou se dirige vers des ressources matérielles. Elles seront notées $DM_{Ri\text{-}j}$ où $R_{i\text{-}j}$ désigne le rôle matériel nécessaire à la fonctionnalité implémentée par un composant logiciel. Les dépendances matérielles sont exprimées sous la forme d'expressions booléennes devant être VRAI. Ces expressions traduisent le fait que tous les composants matériels spécifiés doivent être disponibles et opérationnels afin que le composant logiciel associé puisse être correctement utilisé.

Ces dépendances sont utilisées par la plate-forme d'exécution car les composants logiciels affichant ces dépendances ne seront pas traités de la même manière que les autres lors des phases de configuration. Certains cas de reconfiguration consistent à déplacer un composant d'un site vers un autre. De par leur nature, le déplacement de ce genre de composants est délicat car il faut que le site cible se dote des ressources matérielles nécessaires. Par conséquent, les déplacements de ces composants ne constituent pas des solutions primordiales utilisables par la plate-forme.

Les rôles réalisés de façon matérielle se retrouvent sur les graphes de transition sous la forme de dépendances matérielles.

2.2 Mise en évidence des Flux Synchrones

Un autre aspect important de la transformation proposée est la mise en évidence des flux synchrones sur les graphes de transition et, qui plus est, de la synchronisation entre certains flux. Chaque type de flux synchrone défini par le modèle Korrontea correspond à des spécifications définies par les graphes fonctionnels. Ainsi, les flux primitifs sont utilisés pour représenter des flux modélisant un seul type de données. Les flux composés, quant à eux, sont utilisés pour désigner des flux de données liés par des relations de synchronisation inter-flux. Ces flux peuvent se retrouver ainsi liés à deux niveaux. Le premier résulte d'une spécification des graphes fonctionnels à l'aide des liens de synchronisation inter-flux. Le second est un lien de synchronisation plus implicite dés lors qu'il peut apparaître comme le résultat d'un traitement particulier. A ce titre, on peut citer l'exemple d'un composant d'acquisition audio/vidéo qui produi-

rait alors un flux composé des flux audio et vidéo en figeant les relations inter-flux définies par le traitement. Nous verrons ensuite dans le modèle Osagaia les implications de ces deux possibilités.

Enfin, ces données ainsi représentées doivent être manipulables par les entités fonctionnelles. Les flux synchrones possèdent des implications sur ces entités car elles doivent être capables de manipuler indifféremment les flux primitifs et les flux composés. Ces modalités de manipulation des flux seront définies par le modèle Osagaia.

2.3 Représentation des Graphes de Transition

Le principe de base de cette transformation est de dériver les graphes fonctionnels par rapport aux critères évoqués précédemment afin d'obtenir des graphes de transition qui symbolisent le passage d'une vue fonctionnelle de l'application à une vue proche de l'implémentation. Cependant, ils restent des outils de modélisation abstraite dés lors qu'ils ne font aucune hypothèse sur les façons concrètes de réaliser les implémentations qu'ils décrivent. Selon cette approche, une application est maintenant décrite en termes de composants logiciels, d'opérateurs de flux composés et de flux primitifs. Afin d'assurer la compatibilité et la faisabilité de cette transformation, nous conservons les mêmes propriétés que celles des graphes fonctionnels. Ils sont donc directs, polaires et orientés et sont notés GT(CL, F_p, F_{comp}, F_{cond}).

Les graphes de transition introduisent de manière explicite le concept de composants logiciels qui devient désormais un concept central.

2.3.1 Les Composants Logiciels

Chaque rôle atomique Ri-j identifié sur les graphes fonctionnels décrit une fonctionnalité particulière d'une AMD qui peut être réalisée par voie logicielle ou matérielle. Les rôles matériels sont modélisés à l'aide des dépendances matérielles. Les rôles logiciels sont, quant à eux, réalisés à l'aide de composants logiciels [ALD02], [HEI01], [SAM97].

Les composants logiciels sont donc les briques de base des AMD, ils sont assemblés en parallèle et/ou en séquence à l'aide des flux synchrones. Un composant – noté C_{i-j} – est une entité logicielle qui fournit une implémentation dont le but est d'assurer l'un des rôles atomiques dont l'AMD doit se doter. A chaque rôle atomique correspond donc un composant logiciel. La correspondance entre ces deux notions est donnée à l'aide d'une table de correspondance. La Table 6 est un exemple d'une telle table qui correspond à l'exemple introduit lors du chapitre 3. Cette table énumère les rôles atomiques obtenus lors de l'étape 4 de la méthode de conception (cf. chapitre 4).

Pour chaque rôle, elle indique sa fonction. Ainsi, chaque rôle est mis en correspondance avec le composant chargé de le réaliser. Les cases grisées indiquent que les composants désignés possèdent des dépendances matérielles. Pour chacun d'eux, la table indique les expressions associées. La table de correspondance met en évidence les composants fonctionnels nécessaires à l'implémentation.

Table 6 Table de Correspondance entre Rôles atomiques et Composants

Rôles Atomiques		**Composants Logiciels**	
Nom	***Fonction***	***Nom***	***Fonction***
R_{1-1}	Caméra	C_{1-1}	Composant d'Acquisition Vidéo $DM_{R1-1}.DM_{R1-2}$
R_{1-2}	Carte Vidéo		
R_{1-3}	Acquisition Vidéo		
R_{2-1}	Microphone	C_{2-1}	Composant d'Acquisition Audio $DM_{R2-1}.DM_{R2-2}$
R_{2-2}	Carte Son		
R_{2-3}	Acquisition Audio		
R_{3-1}	Restitution Vidéo	C_{3-1}	Composant de Restitution Vidéo $DM_{R3-2}.DM_{R3-3}$
R_{3-2}	Carte Vidéo		
R_{3-3}	Ecran		
R_{4-1}	Réduction Taille Image	C_{4-1}	Composant de Réduction de la Taille de l'Image
R_{4-2}	Traitement Noir & Blanc	C_{4-2}	Composant de Traitement Noir & Blanc
R_{5-1}	Restitution Audio	C_{5-1}	Composant de Restitution Audio $DM_{R5-2}.DM_{R5-3}$
R_{5-2}	Carte Son		
R_{5-3}	Haut-Parleur		
R_{6-1}	Restitution Audio	C_{6-1}	Composant de Restitution Audio $DM_{R6-2}.DM_{R6-3}$
R_{6-2}	Carte Son		
R_{6-3}	Haut-Parleur		
R_{7-1}	Acquisition Sous-Titres	C_{7-1}	Composant d'Acquisition des Sous-Titres
R_{8-1}	Restitution Sous-Titres	C_{8-1}	Composant de Restitution des Sous-Titres $DM_{R8-2}.DM_{R8-3}$
R_{8-2}	Carte Vidéo		
R_{8-3}	Ecran		

La Table 6 définit les composants fonctionnels c'est-à-dire les composants liés aux aspects métier de l'application. Cependant, les composants logiciels des graphes

de transition ne représentent pas uniquement ce type de fonctionnalités. Les règles de transformation vont faire apparaître d'autres types de composants. Ces derniers sont nécessaires à la réalisation des spécifications fonctionnelles telles qu'elles sont définies par les graphes fonctionnels. Ces composants concernent l'implémentation des propriétés non-fonctionnelles du modèle défini. Ces composants sont nommés opérateurs car ils se destinent à la réalisation d'opérations sur les flux synchrones.

Cette transformation conduit donc à distinguer deux grands types de composants. L'un traite des aspects fonctionnels et l'autre des aspects non-fonctionnels. Cette distinction est guidée par la séparation des préoccupations[55] [LOP95]. La programmation orientée aspects [BOU01], [KIC97] est une de ces approches. Nous utilisons ce point de vue dans nos travaux. La Figure 41 présente les différents types de composants logiciels que l'on identifie.

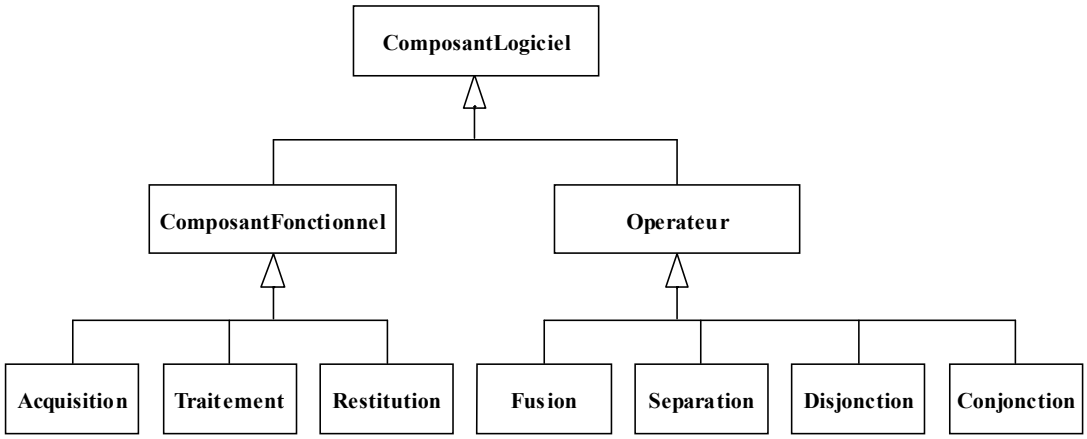


Figure 41 Les Types de Composants Logiciels

La Figure 41 donne des exemples de chacun des types de composants.

Les graphes de transition décomposent une AMD comme un ensemble de composants – représentés par les nœuds – connectés entre eux par des flux synchrones composés et primitifs – représentés par les arcs.

2.3.2 Les Nœuds

Les nœuds des graphes de transition représentent les composants logiciels. Les composants fonctionnels sont notés $C_{i-j} \in CL$ sur les graphes de transition, où CL est l'ensemble des nœuds du graphe donc l'ensemble des composants logiciels du modèle. Par conséquent, les opérateurs font également partie de l'ensemble CL, ils sont notés M_i, S_i, D_i et Co_i.

---

[55] En anglais Separation of Concerns (SoC).

Le caractère polaire des graphes fonctionnels est conservé dans les graphes de transition. Il est défini à l'aide de nœuds fictifs appelés nœud source et nœud puits. Ces nœuds représentent les sommets de tout graphe de transition, ils sont notés respectivement C_s et C_p.

2.3.3 Les Arcs

Les arcs représentant les flux synchrones primitifs (f_i) et composés (F_i) se décomposent en trois types :

- ceux appartenant à l'ensemble F_p désignent les flux primitifs ;
- ceux appartenant à l'ensemble F_{comp} concrétisent les flux composés ;
- enfin, ceux appartenant à l'ensemble F_{cond} représentent les arcs conditionnels introduits par les graphes fonctionnels. Ils représentent indifféremment des flux primitifs ou composés. Ils permettent d'exprimer des choix de configuration donc des informations de QdS.

$F_p \cup F_{comp} \cup F_{cond}$ (avec $F_p \cap F_{comp} \cap F_{cond} = \varnothing$) représente l'ensemble des arcs d'un graphe de transition.

Les arcs appartenant à l'ensemble $F_p \cup F_{comp}$ indiquent que le nœud auquel ils aboutissent appartient à l'AMD quelle que soit la configuration et la QdS choisie.

Les arcs appartenant à l'ensemble F_{cond} modélisent plusieurs choix de configurations possibles en respectant les contraintes d'utilisation de certains composants notées $O_{Ci\text{-}j}$. De telles contraintes permettent de rendre obligatoire l'utilisation du composant spécifié en amont de la spécification de ces contraintes. De plus, ils possèdent des conditions associées qui permettent de définir les contraintes à respecter dans le choix d'une configuration plutôt qu'une autre. Ces conditions sont exclusives et notées $Cond_i$.

Les caractéristiques d'orientation et de cycle des graphes fonctionnels sont les mêmes pour les graphes de transition (cf. §2.4 du chapitre 4). Nous ne les détaillons pas à nouveau.

Nous précisons, sur la Figure 42, les représentations des arcs et des nœuds utilisés par les graphes de transition.

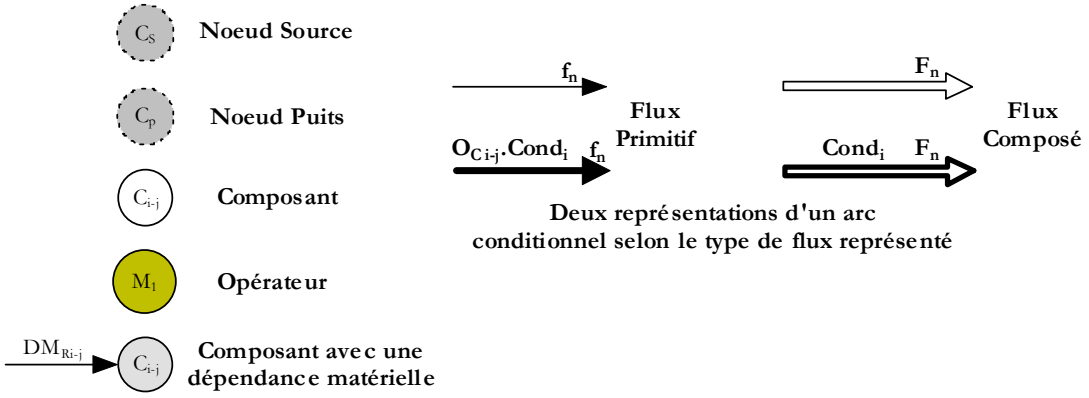


Figure 42 Représentations des éléments du Graphe de Transition

2.4 Transformation des Graphes Fonctionnels

Les graphes de transition sont dérivés des graphes fonctionnels à l'aide des règles de transformation permettant d'introduire le concept de composants logiciels et les flux synchrones primitifs et composés. Chaque règle permet de passer de la représentation fonctionnelle à la représentation des graphes de transition. Nous décrivons chacune d'elles en donnant des représentations possibles équivalentes.

Règle 1 *Les Composants Logiciels*

Chaque rôle atomique est remplacé par un composant logiciel chargé de réaliser la fonctionnalité qu'il représente. La Figure 43 résume cette règle. Les flux entrants et sortants des composants logiciels peuvent être primitifs ou composés. Sur la Figure 43, seuls des flux primitifs sont représentés.

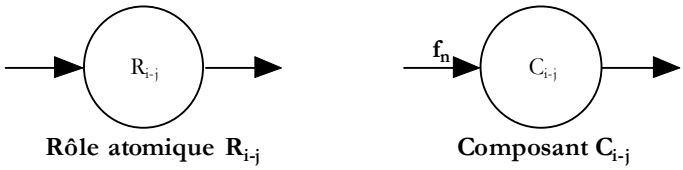


Figure 43 Passage des Rôles Atomiques aux Composants Logiciels

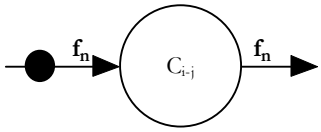
Un rôle atomique de transformation d'une vidéo en noir et blanc sera réalisé par un composant de traitement de la vidéo équivalent. Il requiert en entrée un flux vidéo et fournit en sortie le flux vidéo en noir et blanc.

Les traitements implémentés par les composants logiciels sont susceptibles de nécessiter plusieurs flux en entrée. Cette contrainte signifie que le traitement nécessite plusieurs types de données pour être exécuté. Nous avons vu à maintes reprises que nous faisons abstraction totale du type des données dans le modèle Korrontea. La question qui se pose alors est de savoir comment reconnaître les flux à traiter en entrée d'un composant ? De plus, lorsqu'un composant nécessite plusieurs entrées, c'est

que les données produites en sortie dépendent du traitement des données en entrée. Ce traitement s'effectuant dans un certain sens. En effet, un algorithme de traitement décrit une sémantique particulière donnée par le séquencement des tâches à réaliser et l'ordre des opérations à effectuer sur un ou plusieurs types de données. Le problème est donc de connaître les flux qui vont être traités, quand ils vont l'être et dans quel ordre. Par conséquent, nous proposons de traiter cette dimension sémantique des traitements en identifiant des flux prépondérants pour chaque composant d'une AMD.

Règle 2 *Les Flux prépondérants*

Dans les traitements réalisés par les composants logiciels, nous introduisons de la sémantique en définissant le concept de flux prépondérants. Un flux prépondérant est un flux désigné en entrée d'un composant afin de déterminer l'importance de ce flux dans le traitement implémenté. Ainsi, tout flux connecté en entrée d'un composant, qu'il soit primitif ou composé, peut être défini comme prépondérant mais un composant possède en entrée au plus un tel flux. Lorsqu'un flux est défini de la sorte, cela signifie que le traitement du composant auquel il est connecté porte sur ce flux. Lorsqu'aucun flux prépondérant n'est indiqué, on considère que le composant crée un nouveau flux à partir des flux d'entrée ou pas. En effet, certains composants ne possèdent pas d'entrées et donc pas de flux prépondérant, c'est le cas des sources localisées. Sur les graphes de transition, les flux prépondérants sont indiqués à l'aide d'une pastille noire comme sur la Figure 44. Lorsqu'un flux composé est désigné comme prépondérant, il faut comprendre que c'est l'un des flux traités qui représentera la référence dans le traitement.



Composant C_{i-j}

Figure 44 Flux prépondérant en entrée d'un Composant

Si on considère le cas d'un composant qui reçoit deux flux vidéo correspondant à un présentateur de journal pour l'un et à une vue d'une rue de la ville dont il parle pour l'autre. Le traitement implémenté par ce composant procède à une incrustation du journaliste sur un fond défini par la rue de la ville. Chacun de ces flux est un flux composé qui contient les images et le son associé. Dans ce cas là, le flux d'images produit par le composant sera placé dans un flux composé contenant également les deux bandes sons initiales. La notion de flux prépondérant permet de résoudre ce problème en introduisant de la sémantique dans ce type de traitement. Ainsi, si l'on choisit, lors de la conception de l'AMD, que le flux prépondérant est celui correspondant au journaliste, pour le composant cela signifie que le son restera synchrone des images qu'il produit alors que l'autre bande son sera resynchronisée en sortie du composant. De plus, l'implémentation du composant sera plus facile à écrire dés lors que

le traitement se résumera à l'incrustation du flux vidéo prépondérant (le journaliste) sur l'autre flux vidéo (celui de la rue). Par ce choix de flux prépondérant, le concepteur indique que la vidéo liée au journaliste contient l'information importante et que le composant d'incrustation doit privilégier ce flux. Nous verrons l'utilité de cette démarche lors de la présentation du modèle de composants qui traite des aspects de traitement des flux.

Nous remarquons sur la Figure 44 qu'un flux prépondérant conserve son identité en sortie du composant. En effet, ce dernier n'altère pas cette information, il applique seulement un traitement sur les données de ce flux. Lorsqu'un composant ne possède pas de flux prépondérant en entrée, on a affaire à une source localisée.

Règle 3 *Les Sources Localisées*

On définit une source localisée comme un composant dont le but est de créer ou de capturer des flux synchrones. Sur un graphe de transition, une source localisée se distingue par le fait qu'elle ne possède pas de flux prépondérant en entrée c'est-à-dire que la source localisée produit un nouveau flux à partir de flux d'entrée ou pas. La Figure 45 donne la représentation d'une source localisée. Les informations liées aux flux produits sont donc initialisées ou modifiées par cette dernière. Dans ces cas de figure, le ou les flux sortants d'une source localisée sont toujours différents du ou des flux entrants.

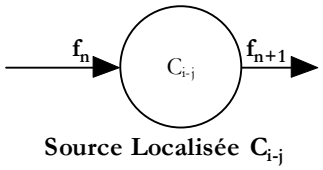


Figure 45 Représentation des Sources Localisées

Une source localisée est capable de créer indifféremment des flux synchrones primitifs et composés.

Les sources localisées correspondent à des composants de création ou d'acquisition de flux synchrones. Ils se distinguent sur les graphes de transition par l'absence de flux prépondérant en entrée.

Certains rôles atomiques sont destinés à être réalisés de façon matérielle. Nous avons vu que ces composants sont ignorés sur les graphes de transition. Nous spécifions seulement les composants logiciels associés en définissant leurs dépendances vis-à-vis du matériel : c'est le concept de dépendances matérielles abordé précédemment (cf. § 2.1).

Règle 4 *Les Dépendances Matérielles*

Les composants qui sollicitent des ressources matérielles pour réaliser leur tâche sont représentés d'une couleur différente sur les graphes de transition. Les ressources nécessaires sont indiquées à l'aide de dépendances matérielles exprimées par une expression logique sur les flux entrants ou sor-

tants, suivant que ces ressources sont nécessaires en amont ou en aval du composant concerné (cf. Figure 46 et § 2.1). Ces informations sont importantes car elles constituent des contraintes à respecter impérativement lors du déploiement de ces composants.

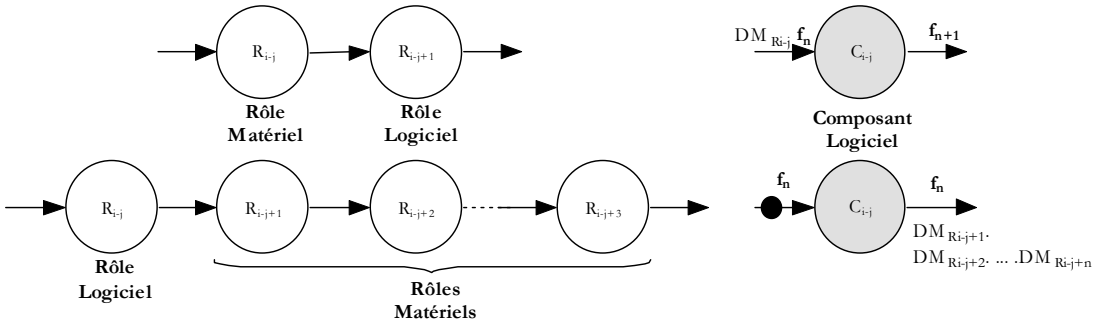


Figure 46 Représentation des Dépendances Matérielles

Un composant de restitution audio nécessite d'être déployé sur un site équipé d'une carte son et de haut-parleurs pour assurer la restitution du média concerné à l'utilisateur.

Les graphes fonctionnels introduisent des liens entre les flux afin de spécifier les relations de synchronisation inter-flux. Ces relations sont définies à l'aide des politiques de synchronisation appliquées sur un ensemble de flux synchrones provenant du même site.

Règle 5 *Opérateur de Fusion*

Les flux liés par des liens de synchronisation sur les graphes fonctionnels sont rassemblés dans des flux composés afin de figer ces relations. Les flux composés sont notés F_n sur les graphes de transition. Cette règle de transformation permet d'introduire un opérateur nécessaire à la réalisation de cette spécification fonctionnelle. Nous l'appelons opérateur de fusion et le notons M_i. Il permet de produire un unique flux composé. La Figure 47 décrit la transformation et introduit cet opérateur. Pour mener à bien son rôle, M_i utilise les politiques de synchronisation définies par le modèle Korrontea (cf. chapitre 5) qu'il applique en fonction des contraintes des flux synchrones qu'il reçoit en entrée. Conformément à ce modèle, ces flux doivent provenir du même site. Un tel opérateur est capable de traiter indifféremment des flux synchrones primitifs voire même composés.

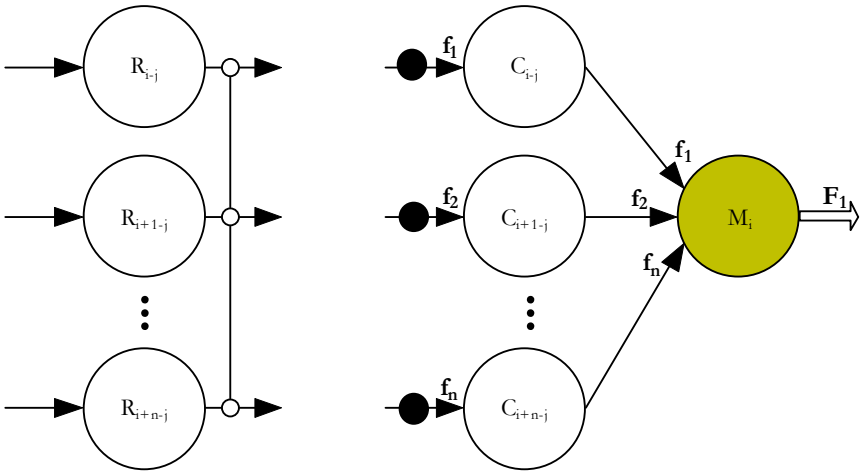


Figure 47 Opérateur de Fusion

Sur la Figure 47, les flux prépondérants en entrée des composants sont obligatoires puisque les flux f_1 à f_n doivent être rassemblés dans un flux composé. Si un ou plusieurs de ces composants qui acceptent ces flux en entrée sont des sources localisées, cela veut dire que les flux produits ne pourront pas être mis dans le même flux composé que les autres. Pour que des flux soient fusionnés par les politiques de synchronisation, il faut qu'ils proviennent tous du même site afin de pouvoir comparer leurs étiquettes temporelles. Or, si un des flux entre sur une source localisée, le flux résultant ne possèdera plus les mêmes références, en terme de provenance, que les autres et donc il ne pourra plus être synchronisé avec eux puisque ses étiquettes temporelles n'auront aucune signification par rapport à celles des autres.

Par exemple, lorsqu'un flux audio et un flux vidéo, produits sur le même site, doivent être transportés et manipulés de façon synchrone, on crée un flux composé qui fige les relations de synchronisation inter-flux qu'ils possèdent.

On définit l'opérateur inverse qui permet de séparer un flux composé en un ensemble de flux synchrones primitifs. Cet opérateur correspond à la fin du transport synchrone des flux dans une AMD spécifié par un nouveau lien de synchronisation entre ces mêmes flux.

Règle 6 *Opérateur de Séparation*

L'opérateur de séparation est la fonction inverse de l'opérateur de fusion. Il permet de séparer le flux composé qu'il reçoit en un ensemble de flux primitifs. Cet opérateur peut être utilisé sur tout flux composé. Il est employé également lorsque la contrainte de synchronisation inter-flux prend fin sur les graphes fonctionnels. En effet, le premier lien indiqué entre plusieurs flux signifie qu'ils doivent être transportés de façon synchrone. Si un nouveau lien est appliqué sur les mêmes flux, cela veut dire que la contrainte prend fin (cf. partie gauche de la Figure 48) et donc que le transport synchrone n'est plus imposé. Cet opérateur se nomme S_i.

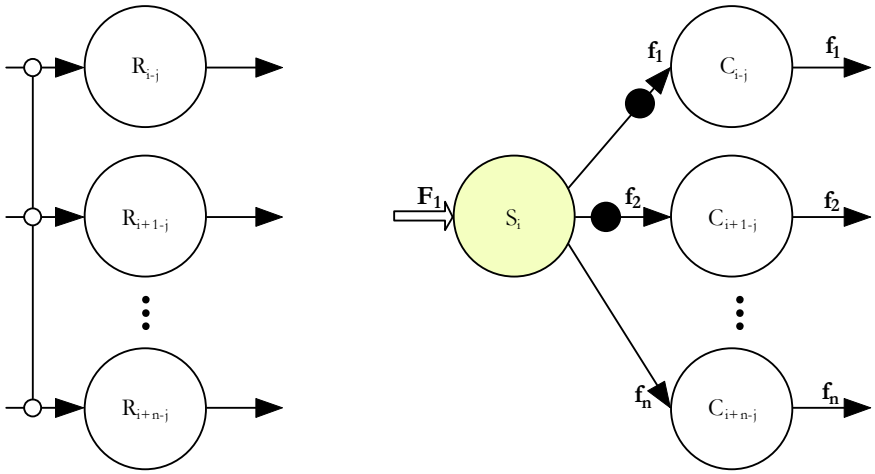


Figure 48 Opérateur de Séparation

La Figure 48 introduit un exemple de représentation possible. Il est clair que les flux sortants de S_i ne sont pas forcément prépondérants pour les composants suivants. Cette possibilité dépend des spécifications nécessaires dans la suite du graphe.

L'opérateur de séparation peut être utilisé lorsque l'on dispose de composants de restitution différents pour les différentes données incluses dans un flux composé. Par exemple, un flux composé d'un flux vidéo et d'un flux audio où chacun doit être diffusé par un composant différent devra être séparé par ce type d'opérateur afin d'envoyer chaque média vers les composants adéquats.

La règle suivante permet de mettre en évidence l'impact du transport synchrone des flux de données sur l'architecture logicielle des AMD. Elle permet d'éviter une des sources de désynchronisation identifiées.

Règle 7 *Traitement des Flux liés par des relations de synchronisation inter-flux*

Certains rôles atomiques des graphes fonctionnels décrivent des fonctionnalités de traitement des flux de données liés par des relations de synchronisation inter-flux. Sur les graphes de transition, ces rôles correspondent à des composants de traitement qui acceptent en entrée un flux composé bien que leur traitement ne s'applique pas nécessairement sur la totalité des flux de données qu'ils contiennent. Cette description est schématisée sur la Figure 49. Cette solution permet de pallier le premier cas de désynchronisation inter-flux identifié lors de l'état de l'art (cf. § 2.4.1 du chapitre 1). Nous avions mis en évidence que les traitements des flux de données synchrones introduisent des temps de latence par rapport à ceux qui ne le sont pas. Ainsi, les composants reçoivent l'ensemble des flux par l'intermédiaire d'un flux composé puis traitent les flux concernés en maintenant les liens de synchronisation avec les autres qui ne font alors que transiter dans le composant. Les relations de synchronisation inter-flux qui les unissent ne sont donc pas détruites.

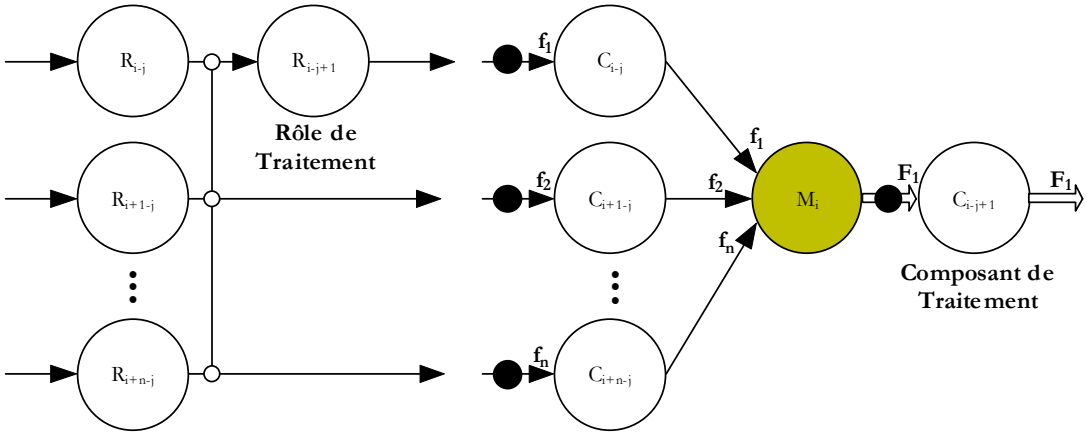


Figure 49 Traitement des Flux de Données Synchrones

Sur la Figure 49, le flux composé F_1 qui entre dans un composant de traitement n'est pas forcément prépondérant, cela dépend des spécifications de l'AMD à modéliser. Le cas échéant, le composant de traitement est considéré comme une source localisée et donc le flux de sortie est un nouveau flux composé fabriqué à partir des données du flux F_1.

Lorsque l'on veut transformer une vidéo en noir et blanc liée à un flux audio dans un flux composé, alors on envoie le flux composé en entrée du composant de traitement. Le composant fournit en sortie le flux composé qui contient la vidéo traitée synchrone de l'audio.

Certaines structures introduites sur les graphes fonctionnels nécessitent d'être considérées sur les graphes de transition. C'est le cas, par exemple, lorsque les flux qui constituent un même flux composé doivent être traités en parallèle. Ce type de structure se matérialise sur les graphes de transition par une disjonction. Nous introduisons, pour ce faire, un opérateur qui autorise ces types de traitement.

Règle 8 *Opérateur de Disjonction*

Cette règle est en fait une généralisation de la règle 7 qui concernait alors le traitement d'un seul flux de données appartenant à un flux composé. Avec cette transformation, on émet la possibilité de traiter plusieurs flux de données d'un même flux composé en parallèle. Les traitements ainsi appliqués ne doivent pas détruire les relations de synchronisation qui les lient (cf. règle 7). La Figure 50 donne une représentation de cette transformation. Pour réaliser ce type de structure sur les graphes de transition, nous introduisons un opérateur de disjonction qui permet de produire n flux synchrones à destination des composants de traitement comme le montre la Figure 50. Les flux qui sortent de cet opérateur peuvent être primitifs ou composés selon les besoins. Lorsqu'un tel opérateur est mis en place alors les flux ainsi « éclatés » sont par défaut prépondérants car il faut qu'à la fin de la disjonction on puisse retrouver les relations de synchronisation qui existaient alors.

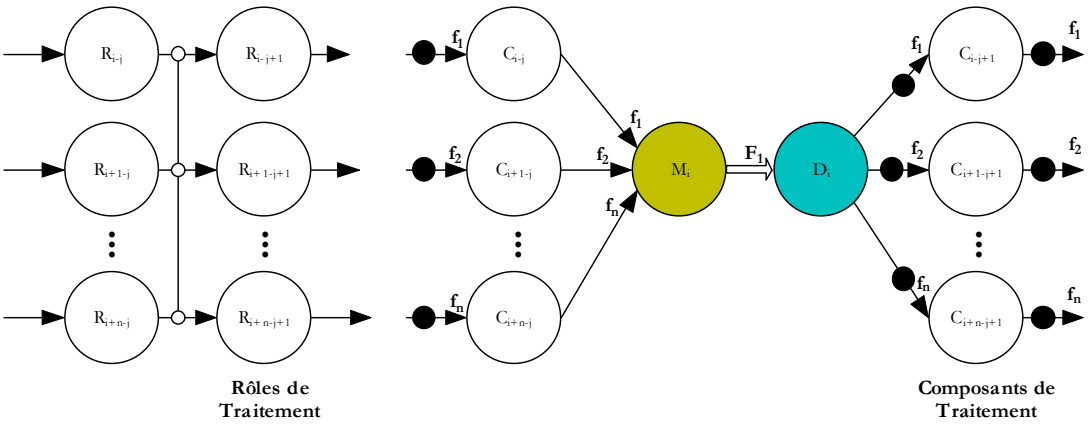


Figure 50 Opérateur de Disjonction

Une solution plus simple à mettre en œuvre consisterait à faire transiter le flux composé à travers les différents composants de traitement assemblés en séquence. Nous ne retenons pas cette solution car elle n'est pas satisfaisante puisqu'elle ne correspond plus aux spécifications fonctionnelles données par les graphes (cf. graphe de gauche de la Figure 50). De plus, en terme de QdS, les deux représentations ne sont pas équivalentes. En effet, la parallélisation de ces composants permet un gain de temps, de performances et donc de QdS. La QdS étant une propriété importante dans nos travaux, nous préférons donc adopter la solution décrite par la règle 8.

On peut, par exemple, vouloir traiter la vidéo et le son d'un flux composé en parallèle. Dans ce cas de figure, nous utiliserons un opérateur de disjonction qui séparera le flux composé en deux flux primitifs, un pour chaque média. Ainsi, chacun de ces flux sera envoyé vers les composants de traitement concernés.

Les disjonctions sont, en général, suivies de conjonction. Les flux ainsi traités en parallèle se rejoignent en un nœud du graphe. Nous introduisons un nouvel opérateur qui permet de traiter les conjonctions.

Règle 9 *Opérateur de Conjonction*

Lorsque les traitements en parallèle (cf. règle 8) de plusieurs flux synchrones sont terminés, il est nécessaire de reconstituer le flux composé tel qu'il existait avant la disjonction. Pour ce faire, nous introduisons un opérateur de conjonction que l'on note Co_i*. Son rôle est de reproduire le flux composé, c'est l'opérateur inverse de l'opérateur de disjonction. Son principe est donné sur la Figure 51. Cet opérateur ne peut être utilisé que si l'on a utilisé auparavant l'opérateur de disjonction. Cette condition est spécifiée à l'aide de la contrainte d'utilisation* O_{Di}*.*

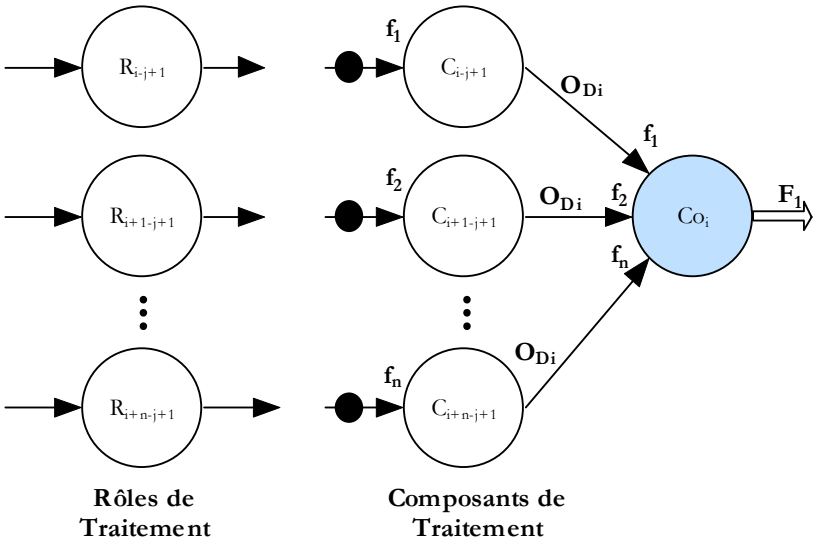


Figure 51 Opérateur de Conjonction

Ainsi, les disjonctions et les conjonctions sont modélisées à l'aide d'opérateurs sur les flux (cf. règle 8 et 9). Nous remarquons sur la Figure 51 que par défaut tous les flux appartenant à ce type de chemin sur le graphe sont prépondérants (cf. règle 8). Le cas échéant, cela voudrait dire qu'une source localisée est définie dans ces chemins de disjonction et donc que l'opérateur de conjonction ne pourrait pas reconstituer le flux composé tel qu'il existait avant cette structure.

Les quatre opérateurs qui viennent d'être présentés font partie de l'implémentation non-fonctionnelle du modèle de composants Osagaia. Leur existence se justifie par le fait qu'ils permettent de mettre en œuvre des spécifications fonctionnelles de la méthode de conception.

Enfin, nous nous intéressons aux arcs conditionnels introduits par les graphes fonctionnels. Ces arcs sont conservés car ils permettent de spécifier différentes configurations dans les AMD représentant différents niveaux de QdS. Ils constituent donc des informations intéressantes pour la plate-forme d'exécution.

Règle 10 *Les Arcs Conditionnels*

Dans les graphes de transition, les arcs conditionnels sont représentés de la même manière. A la différence que dans ces graphes, ces arcs peuvent maintenant représenter un flux synchrone primitif ou un flux synchrone composé. Les choix de configuration sont réalisés à l'aide des conditions apposées sur chacun de ces arcs ($Cond_i$). La Figure 52 décrit ces représentations. De plus, des contraintes d'utilisation de certains composants peuvent être indiquées dans ce type de spécification.

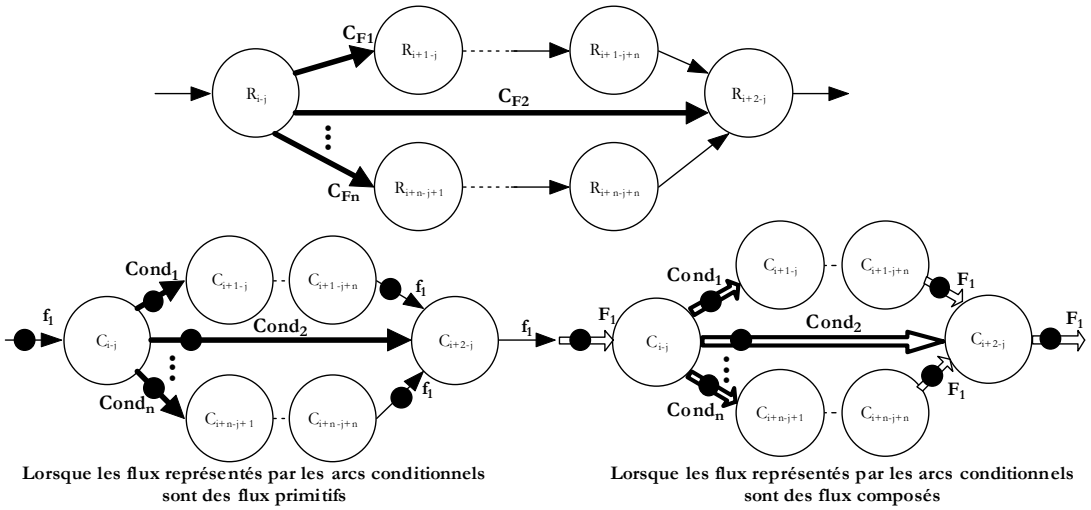


Figure 52 Représentation des Arcs Conditionnels

Sur la Figure 52, il n'est pas obligatoire que tous les flux soient prépondérants.

Les règles de transformation, ainsi introduites, ont pour objectif de dériver les graphes fonctionnels afin de les transformer en graphes de transition dont les éléments sont des composants logiciels, des opérateurs et des flux synchrones primitifs et composés. L'ensemble de ces éléments va être décrit par le modèle de composants Osagaia. Ce modèle a pour objectif de spécifier des unités d'implémentation utilisables dans la description d'une architecture logicielle similaire à celle donnée sur les graphes de transition. Ce modèle va également permettre de préciser l'interconnexion de ces éléments et donc le fonctionnement global d'une telle architecture. Les entités de ces modèles devront savoir manipuler les flux synchrones afin de réaliser leurs tâches.

Avant de se lancer dans la définition de ce modèle, nous donnons un exemple complet de graphe de transition obtenu par application des règles de transformation.

2.5 Exemple de Transformation

L'exemple utilisé est celui du chapitre 3. Les principes du fonctionnement de cette AMD sont décrits dans ce chapitre. Un des graphes fonctionnels possibles est donné dans le chapitre 4. Le graphe de transition obtenu après application des règles de transformation est représenté sur la Figure 53.

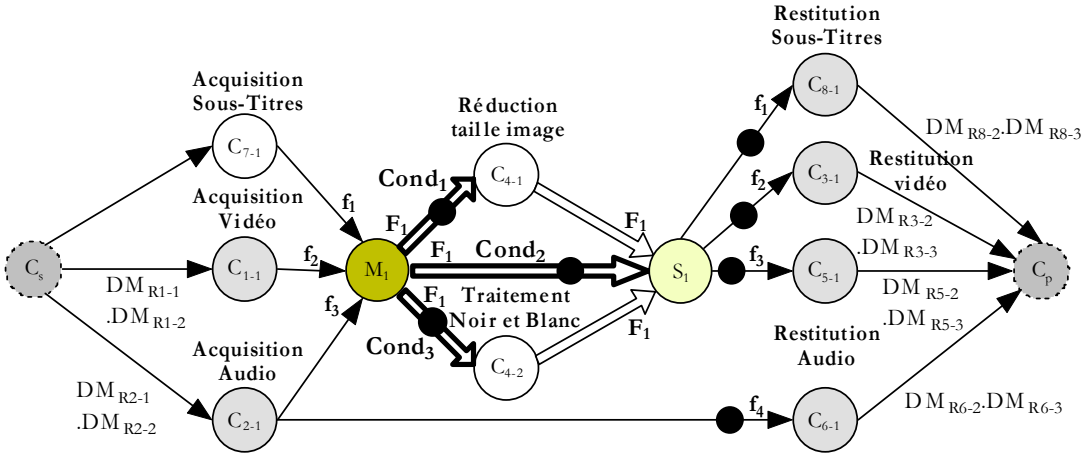


Figure 53 Graphe de Transition de l'Application de Formation à Distance

Les flux qui relient les composants sources et puits aux autres composants sont fictifs, c'est pour cela qu'ils ne sont pas désignés sur un graphe de transition. On considère que les flux sont créés par des sources localisées. Les composants $C_{7\text{-}1}$, $C_{1\text{-}1}$ et $C_{2\text{-}1}$ sont des sources localisées desquelles résultent les flux f_1, f_2 et f_3. Mis à part $C_{7\text{-}1}$, ces sources nécessitent l'utilisation de ressources matérielles. Ces spécifications sont précisées à l'aide des dépendances matérielles. Il en est de même pour les composants de restitution $C_{8\text{-}1}$, $C_{3\text{-}1}$, $C_{5\text{-}1}$ et $C_{6\text{-}1}$. La liste exhaustive des rôles matériels associés à ces dépendances est donnée dans la Table 6 du présent chapitre.

Les flux primitifs, ainsi créés, sont liés par des relations de synchronisation inter-flux. L'opérateur M_1 permet de figer ces relations et de produire le flux composé F_1.

Trois configurations sont possibles avant d'envoyer les flux vers leurs puits respectifs. Elles sont décrites par les chemins représentés par $Cond_1$, $Cond_2$ et $Cond_3$ sur le graphe (cf. Figure 53). Le chemin qui correspond à $Cond_1$ propose de réduire la taille initiale de l'image du flux vidéo. Celui décrit par $Cond_2$ n'applique aucun traitement et enfin le chemin qui correspond à $Cond_3$ permet de transformer le flux vidéo en noir et blanc.

Ensuite, un composant de séparation S_1 permet de retrouver les flux primitifs f_1, f_2 et f_3 à partir du flux composé F_1. Le but est d'envoyer chacun des flux vers son composant de restitution respectif ($C_{3\text{-}1}$, $C_{5\text{-}1}$ et $C_{8\text{-}1}$). La modélisation de l'architecture décrite peut paraître un peu étrange. En effet, on synchronise f_1, f_2 et f_3, on les traite ou pas suivant la configuration choisie et ensuite on les sépare à nouveau pour les envoyer vers leur composant de restitution respectif. En fait, pour éviter cette impression, il faut tenir compte d'un paramètre qui n'apparaît ni sur les graphes fonctionnels, ni sur les graphes de transition : le réseau. Les transferts réseaux sont définis lors du

déploiement des AMD à l'aide du modèle de composants. Il est fort probable que les configurations décrites par les arcs conditionnels introduisent le réseau après les traitements. En effet, nous rappelons que cette AMD est distribuée sur trois sites différents qui correspondent aux trois utilisateurs de l'application. Ainsi, ces arcs conditionnels permettent éventuellement d'insérer des traitements sur la vidéo si la bande passante des liaisons réseaux est amenée à diminuer. Ainsi, d'après les spécifications de l'exemple et le graphe de transition, les transferts réseaux s'opèreront probablement après les composants de traitement. Dans le chemin concrétisé par $Cond_2$, le flux composé F_1 est créé afin que les transferts réseaux ne détruisent pas les liens de synchronisation inter-flux. Ce cas de figure correspond à la deuxième source de désynchronisation que nous avions également décelée dans le chapitre 1. Il apparaît clairement sur cet exemple que les politiques de synchronisation introduites et les flux composés permettent de l'éviter.

Le flux primitif f_4 est envoyé seul vers un composant de restitution car cette AMD comporte deux utilisateurs qui vont suivre le cours à distance. Le deuxième téléspectateur ne reçoit que l'audio car il utilise un périphérique à capacités restreintes (cf. chapitre 3). Dans ce cas, le flux produit par le composant d'acquisition audio doit être envoyé vers les deux téléspectateurs (f_3 et f_4). Nous verrons lors du déploiement que des opérateurs non-fonctionnels vont être introduits afin de permettre ce dernier. Dans le cas de cette AMD, il est clair que le flux audio doit être dupliqué et envoyé vers les utilisateurs respectifs.

2.6 Synthèse

Les graphes fonctionnels nous ont permis de mettre en évidence la nécessité de disposer de deux entités pour implémenter les AMD. La première est utilisée pour l'implémentation des rôles atomiques. Les graphes de transition introduisent explicitement cette entité sous la forme de composants logiciels. La seconde est destinée à la mise en œuvre des échanges de données entre les composants logiciels. Même si elle est moins évidente sur les différents graphes, elle doit être définie afin d'assurer cette fonctionnalité en considérant la structure des données telle qu'elle a été définie dans le modèle Korrontea. De plus, ces connexions de composants doivent pouvoir être opérées entre des composants s'exécutant sur des machines différentes afin de tenir compte du caractère distribué de ces applications.

Parmi les composants logiciels, la transformation mise en œuvre permet de justifier de l'utilisation d'opérateurs de flux synchrones qui vont permettre d'assurer certaines spécifications fonctionnelles. Ces opérateurs ne sont autres que des composants

logiciels mais nous les avons nommés différemment car les aspects qu'ils adressent ne concernent pas directement les préoccupations métier d'une application. Néanmoins, ils doivent être réutilisables dans ces architectures afin d'éviter aux concepteurs d'avoir à développer les fonctionnalités qu'ils proposent. Ces opérateurs représentent les aspects non-fonctionnels des AMD.

Les composants fonctionnels permettent, quant à eux, l'implémentation des aspects métier des AMD. Nous devons définir leur fonctionnement à un niveau général. Il suffit, pour cela, d'être conscient des mécanismes que tous les composants fonctionnels devront posséder. Ces mécanismes doivent permettre de manipuler les flux synchrones. Ils doivent être capable de distinguer les données à traiter des autres ainsi que de conserver les liens de synchronisation lorsque cela est nécessaire. Nous introduisons une dimension sémantique dans ces derniers à l'aide de la possibilité de spécifier des flux prépondérants pour les traitements. Dans un traitement, un tel flux revêt une importance particulière. Cette notion introduite par les graphes de transition doit également être considérée dans la définition des composants fonctionnels.

Les dépendances matérielles permettent d'identifier les ressources matérielles à utiliser pour l'implémentation de certains composants et donc pour le déploiement des AMD. Ce sont des informations précieuses qu'il faut intégrer. En effet, du fait de leurs dépendances, ces composants ne seront probablement pas traités de la même manière par la plate-forme d'exécution. Ils ne seront, par exemple, pas mobiles pour certains et non duplicables pour tous. Ces composants sont très présents dans les AMD car ils concernent souvent des fonctionnalités d'acquisition et de restitution de médias.

Enfin, les arcs conditionnels permettent la modélisation de différents choix de configuration dans la réalisation d'un même rôle ou d'un ensemble de rôles. Pour que cette approche soit efficace, il est nécessaire que la plate-forme qui détermine les configurations à utiliser ait la connaissance de l'architecture des AMD. En conséquence, les entités qui la composent doivent pouvoir informer la plate-forme sur leurs états de fonctionnement. Elles doivent également être capables de collecter et de délivrer des informations de QdS pertinentes pour la plate-forme d'exécution.

Ce passage des graphes fonctionnels aux graphes de transition modélisé par les règles que nous venons de décrire justifie de la nécessité de disposer d'un modèle de composants répondant à l'ensemble de ces spécifications. Il doit permettre le développement des AMD en adéquation avec notre démarche de conception et ses différentes étapes. Ainsi, c'est la seule manière pour que les AMD aient le comportement que nous prévoyons. L'ensemble des spécifications déduites nous guide dans notre démarche de conception du modèle.

3 Le Modèle de Composants Osagaia

Les différentes étapes de la méthode, le modèle Korrontea et les graphes de transition permettent d'introduire les entités du modèle Osagaia. Nous avons montré la nécessité de disposer de deux entités fonctionnelles nécessaires pour la description des architectures logicielles. La première entité est utilisée pour l'implémentation des composants logiciels et la seconde est nécessaire afin d'assurer le transport des flux synchrones entre ces composants.

L'architecture préconisées est semblable à celles de type « pipes & filter » [GAN94], [GAR94]. Ainsi, chaque composant (qui correspond au concept de « filters ») possède un ensemble d'entrées et de sorties. Les composants doivent être capable de lire des flux synchrones sur leurs entrées puis de les fournir sur leurs sorties après leur avoir appliqué des transformations. Ces composants sont indépendants les uns des autres dans le sens où ils ne partagent pas d'états entre eux[56]. Le seul lien qui unit les composants d'une application est le fait qu'ils sont liés par des flux synchrones et que leur coopération œuvre dans la réalisation d'une fonctionnalité globale. Les données des flux synchrones doivent donc transiter entre les composants. Ces derniers sont connectés entre eux à l'aide de connecteurs (correspondant au concept de « pipes ») utilisés pour le transport des données. Ils permettent de relier la sortie d'un composant à l'entrée du composant suivant conformément aux arcs des graphes. Les avantages de ces architectures se révèlent intéressants pour une problématique de gestion de la QdS. Les AMD sont décomposées selon une approche de modélisation descendante qui permet d'aboutir à un ensemble de rôles atomiques. Une telle décomposition lorsqu'elle considère les différents niveaux hiérarchiques qui lui ont permis d'aboutir est intéressante pour l'obtention d'une architecture flexible. Ce type d'architecture est facilement manipulable dés lors que l'on est capable de considérer et surtout d'intervenir à ces différents niveaux. De plus, le fait que ces entités sont conscientes de leur contexte d'exécution, permet lorsqu'elles sont capables d'en informer un tiers, de mieux appréhender le comportement global d'une AMD. Ainsi, la plateforme possède des informations intéressantes pour sa tâche de gestion de la QdS.

D'après la Figure 41, nous distinguons deux catégories de composants logiciels en fonction des préoccupations qu'ils adressent. Ainsi, les opérateurs de flux synchrones traitent des aspects non-fonctionnels des AMD. Les composants fonctionnels, quant à eux, s'occupent des aspects fonctionnels en se destinant à l'implémentation

56 Cette caractéristique colle à la tentative de définition que nous avons donné dans le chapitre 2 pour les composants logiciels où nous parlions d'unité d'implémentation indépendante. En d'autres termes, un composant doit pouvoir fonctionner tout seul, il n'a pas besoin d'autres composants pour exécuter la tâche qui lui est assignée.

des rôles atomiques. Ces composants font partie du niveau hiérarchique le plus bas des AMD. Ils interviennent donc dans la fourniture d'un niveau de QdS pour une application, ce qui implique de définir des entités supervisables par la plate-forme d'exécution. Le présent modèle s'attache donc à définir ces deux types d'entités.

La connexion de l'ensemble des composants quel que soit leur type est réalisée par un connecteur chargé du transport des flux synchrones primitifs et composés. Le caractère distribué des applications que nous définissons doit être pris en compte à ce stade. En effet, les données doivent pouvoir être transférées entre des composants s'exécutant sur des machines différentes. Le connecteur doit de fait assurer le transport des flux de façon locale ou distribuée, il est en fait l'entité distribuée du modèle Osagaia. Enfin, dés lors que l'architecture d'une AMD peut être manipulée par la plate-forme, les connexions locales et distantes entre les composants doivent donc pouvoir être modifiées. Nous avons baptisé cette entité « conduit » par analogie avec les tuyaux ou conduits que l'on rencontre dans les circuits de distribution de l'eau qui ont pour finalité le transport de l'eau d'un point vers un autre. Dans ces circuits, la taille du tuyau est également un critère déterminant pour le fonctionnement de l'ensemble. Dans les AMD, la taille du tuyau dépend des propriétés des flux synchrones mais aussi de l'environnement distribué ou pas du conduit. Le fonctionnement de cette entité sera détaillé dans une deuxième partie.

Enfin, lorsque l'on définit une telle architecture il faut être sûr que les entités qui la composent vont coopérer de façon efficace afin de permettre un fonctionnement global correct de l'AMD. Il faut donc fournir les moyens nécessaires pour la coordination (que l'on peut également appelé synchronisation[57]) des exécutions des entités [MAL94] de sorte que ce qui est produit par un composant puisse être transporté le plus tôt possible par le conduit à destination d'un autre composant. Pour ce faire, il est important de connaître les instants de disponibilité des données. Cette connaissance est rendue possible à l'aide d'un mécanisme basé sur des événements qui vont permettre d'avertir les entités lors de la disponibilité des données afin que ces dernières transitent de manière efficace dans les AMD.

---

[57] Le terme synchronisation n'est pas employé ici dans le sens où nous l'employons d'habitude. La synchronisation des entités signifie que leurs exécutions doivent être coordonnées afin que les flux circulent en continu entre les entités dans un but affirmé de réaliser la fonctionnalité de l'AMD ainsi conçue.

3.1 Les Composants Logiciels

Nous commençons par aborder les principes de fonctionnement des composants logiciels du modèle Osagaia. Ces composants se distinguent en fonction des aspects traités (cf. Figure 41). Nous abordons les deux types de composants. Une première partie décrit le modèle des composants fonctionnels destiné à implémenter la partie métier des AMD. La partie suivante s'attarde à décrire le fonctionnement détaillé des opérateurs de flux synchrones.

3.1.1 Les Composants Fonctionnels

Les composants fonctionnels se destinent à l'implémentation des rôles atomiques des AMD. De la même manière que l'on distingue les composants selon leurs préoccupations, on peut séparer l'implémentation des composants fonctionnels en deux parties distinctes : les propriétés fonctionnelles et les propriétés non-fonctionnelles. Les propriétés fonctionnelles rassemblent toutes les propriétés qui concernent la mise en œuvre des aspects métier du composant c'est-à-dire le rôle atomique, la fonctionnalité pour lequel il a été conçu ainsi qu'un accès aux données. Les propriétés non-fonctionnelles décrivent les moyens dont doit disposer le composant afin d'assurer un fonctionnement correct de la partie métier. Elles concernent également les mécanismes qui vont assurer son intégration au sein d'un ensemble de composants et de conduits et sa supervision par une plate-forme. La Figure 54 met en exergue cette distinction de propriétés qui est capitale dans la définition de notre modèle. Ce type de séparation nous semble important dans la définition d'un modèle de composants afin d'assurer la réactivité et la réutilisabilité pour les développeurs. Dans le modèle Osagaia, la partie métier utilise des propriétés non-fonctionnelles afin d'assurer son interaction avec la plate-forme d'exécution et les conduits en entrée et le conduit en sortie. En effet, nous allons voir que la partie métier implémentant les niveaux hiérarchiques les plus bas ne peut fournir qu'une seule sortie.

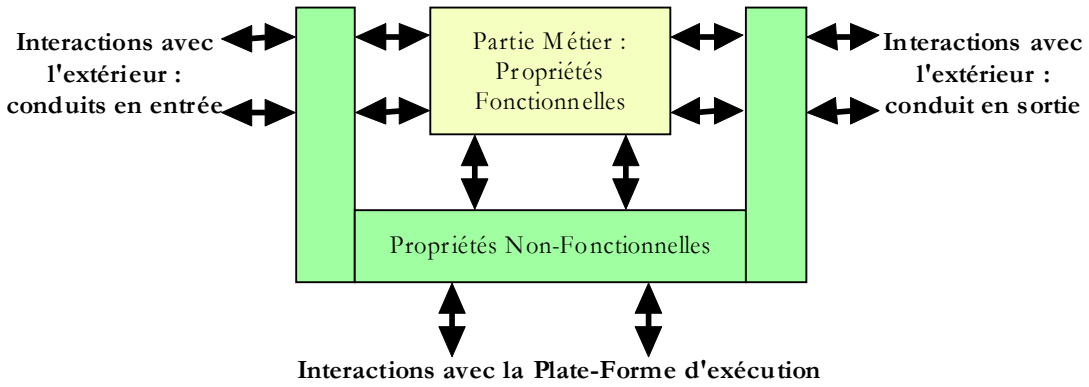


Figure 54 Principes de Fonctionnement des Composants Fonctionnels

La méthode de conception ainsi que les graphes de transition ont permis de déduire un ensemble de propriétés non-fonctionnelles que les composants fonctionnels doivent utiliser. Nous avons classé ces propriétés en fonction de la nature des préoccupations auxquelles elles répondent. Un tel classement permet de mettre en évidence la structure à utiliser. Nous avons recensé dans la Table 7 ces propriétés. Lorsque l'on examine cette table, on trouve deux types de préoccupations pour les propriétés non-fonctionnelles. Le premier type est lié à la gestion des entrées/sorties c'est-à-dire la définition des interactions avec l'extérieur décrite par la Figure 54. C'est à ce niveau qu'impacte directement le modèle de flux de données Korrontea. Le second type est plutôt lié à des préoccupations de contrôle du fonctionnement du composant dans son ensemble. Cet aspect regroupe également les interactions qu'il va avoir avec la plate-forme. Il correspond à un second type d'interactions du composant avec la plate-forme. On retrouve dans cette distinction la gestion des deux types d'interaction d'un composant (cf. Figure 54). La définition des propriétés non-fonctionnelles du composant se dessine donc par rapport aux interactions qu'un composant possède.

Table 7 Propriétés des Composants Fonctionnels

Propriétés Non-Fonctionnelles		**Propriétés Fonctionnelles**
Gestion des Entrées/Sorties	***Contrôle du cycle de vie d'un Composant***	
Gestion des connexions en entrée et en sortie	Contrôle du cycle de vie de la partie métier	Accès aux données
Intégration du composant avec les autres entités (synchronisation des entités)	Liaisons entre les différentes unités d'un composant	Implémentation métier du composant
Lecture des données provenant de l'extérieur et écriture des données vers l'extérieur	Centralise des information de QdS sur le fonctionnement du composant	
Gestion des flux synchrones (récupération des données puis transfert et traitement)	Supervision du composant par la plate-forme	
Gestion des unités d'information des flux synchrones (gestion des attributs, gestion des transferts)	A l'origine des connexions et des déconnexions du composant	

En conséquence, nous fournissons un modèle de composants fonctionnels qui offre les propriétés décrites par la Table 7. Lors de l'état de l'art (cf. chapitre 2), nous avions pris conscience de l'importance des propriétés non-fonctionnelles et nous avions également abordé plusieurs approches envisageables pour la gestion de ces propriétés. Une solution possible est de fournir un modèle unique d'entité chargée de gérer les deux types de propriétés. Cette solution, déjà pratiquée par la programmation orientée objet, montre ses limites surtout lors des tentatives de réutilisation de telles entités. La tâche du concepteur est complexe car il doit gérer de front les deux types de propriétés. Le problème mis en cause est la rigidité de ce type de structure, ce qui ne semble pas satisfaisant pour notre problématique. Actuellement, de nombreux modèles de composants ne fournissent pas cette séparation claire des aspects fonctionnels et non-fonctionnels.

Une autre solution, à notre sens plus intéressante, consiste à fournir un modèle de composants où les différents aspects sont gérés de manière indépendante [LOP95]. Ainsi, le code métier n'est pas « pollué » par l'intégration des propriétés non-fonctionnelles. Les développeurs de tels composants peuvent concentrer leurs efforts sur la partie métier sans se soucier de l'implémentation non-fonctionnelle qui de fait

est transparente pour eux. Cette approche correspond plus à notre vision des choses. En effet, nous pensons qu'elle est essentielle dans les modèles de composants si nous voulons tirer les avantages de flexibilité et de malléabilité sensés être amenés par ce paradigme. Nous pensons que plus l'indépendance des propriétés est claire, plus la réutilisabilité est satisfaisante. Nous nous orientons plutôt vers ce type de solution afin de profiter de ces avantages. Ainsi, le modèle doit fournir un ensemble de propriétés non-fonctionnelles pouvant être vues comme des services qui garantissent les fonctionnements voulus. Ce type de solution préconise l'utilisation d'entités dédiées appelées conteneur [SUN03], [BRU02], [BRU03]. Un conteneur est un environnement d'exécution pour les composants logiciels dont le but est de permettre une gestion aisée et indépendante des propriétés non-fonctionnelles. Son rôle est de proposer des services pour une exécution correcte du composant et des interactions probables avec son environnement. Ainsi, on peut réutiliser ces propriétés par réutilisation des conteneurs pour chaque composant logiciel d'une application. De plus, ces services doivent pouvoir être étendus afin de proposer des solutions à de nouveaux besoins.

Nous fournissons donc une structure permettant de gérer l'ensemble des propriétés fournies dans la Table 7. Cette structure se scinde en deux parties distinctes qui interagissent ensemble comme dans la Figure 54 :

- une partie, que nous appelons le Composant Métier (CM), est chargée d'accueillir le code correspondant aux propriétés fonctionnelles c'est-à-dire l'implémentation d'un rôle atomique et des mécanismes permettant l'accès aux données ;
- une partie, que nous nommons le Processeur Elémentaire (PE), a pour but de fournir les propriétés non-fonctionnelles nécessaires au fonctionnement du CM mais aussi à l'ensemble conduits/PE/CM d'une AMD.

Nous commençons par définir le CM et ses principales caractéristiques. Nous allons voir que les types de traitement qu'il propose ont un impact sur le fonctionnement des modules du PE.

3.1.1.1 Le Composant Métier

Le CM correspond à la partie du modèle Osagaia qui recueille le code applicatif c'est-à-dire le code qui implémente les fonctionnalités des AMD (dans [BOU05], [LAP06] et [SMI94] on trouve quelques exemples de fonctionnalités). Conformément à notre approche, une AMD est composée d'un CM par rôle atomique. Cette règle (cf. règle 1) symbolise le passage des graphes fonctionnels au graphe de transition mis en évidence par la table de correspondance (cf. Table 6). Le concept de CM couvre les

propriétés fonctionnelles du modèle Osagaia, son développement revient au concepteur d'AMD ou au concepteur de composants.

En raison des aspects adressés, nous avons appelé cette entité le composant métier par analogie avec le concept de processus métier rencontré dans l'ingénierie des systèmes d'information. Un processus métier est chargé de fournir des fonctionnalités relatives à un métier ou à un domaine particulier [HEI01]. Les CM sont dédiés à l'implémentation de fonctionnalités propres au domaine du multimédia. Elles consistent, par exemple, à des fonctionnalités d'acquisition vidéo, de mixage de bandes sonores ou de sauvegarde d'une présentation à distance dans un fichier.

3.1.1.1.1 Principes de Fonctionnement

Schématiquement, on peut dire que les CM sont connectés entre eux par des flux synchrones. Les informations sont représentées par des données contenues dans ces flux que les CM se transmettent grâce aux PE. Chaque CM a pour objectif de produire, à partir de ses entrées, des résultats à destination des autres CM. Ce traitement correspond à l'exécution d'une fonction prise ici au sens mathématique du terme c'est-à-dire qui ne dépend que de ses entrées et non de variables globales ou de tout autre information. Ainsi, chaque CM peut effectuer son traitement dés qu'il possède les données nécessaires pour le réaliser. Dans ce type d'architecture, le séquencement n'est pas explicite, la seule séquence que l'on peut rencontrer provient du fait qu'un traitement appliqué par un CM peut dépendre du résultat produit par un autre. De plus, chaque configuration débute par des CM producteurs et aboutit sur des CM consommateurs de flux. Ces types particuliers de CM correspondent respectivement à des implémentations d'acquisition/création et de restitution/stockage de flux.

La méthode de conception mise en œuvre permet de décomposer l'AMD en rôles atomiques. Les rôles atomiques tels qu'ils sont définis dans [LAP06] sont susceptibles d'accepter en entrée un ou plusieurs flux de données et/ou de produire en sortie un seul flux de données. Néanmoins, certains cas de modélisation peuvent aboutir à des rôles qui en fait ne sont pas atomiques dés lors qu'ils proposent plusieurs entrées et/ou plusieurs sorties. La décomposition fonctionnelle n'est donc pas terminée sur ce type de rôles, ce qui veut dire qu'ils peuvent à nouveau être décomposés en rôles atomiques. Afin de pallier ces défauts de conception, il est possible de les considérer à l'implémentation. Deux solutions sont envisageables afin d'implémenter un tel rôle. La première considère que tous les flux de sortie vont être rassemblés dans le même flux composé. La seconde suggère que le rôle n'est pas atomique et donc qu'il peut être décomposé à nouveau en plusieurs rôles atomiques possédant une et une seule sortie. La Figure 55 décrit ce cas de figure à travers l'exemple d'un rôle d'acquisition audio et

vidéo qui fournit en sortie deux flux de données. Soit on décrit un graphe de transition comme celui donné en haut à droite de la Figure 55 c'est-à-dire un composant qui produit en sortie un flux composé contenant l'audio et la vidéo, soit on décompose ce rôle en deux composants d'acquisition distincts ou chacun produit son flux primitif (cf. graphe de transition en bas à droite de la Figure 55). L'une et l'autre de ces solutions sont tout à fait acceptables.

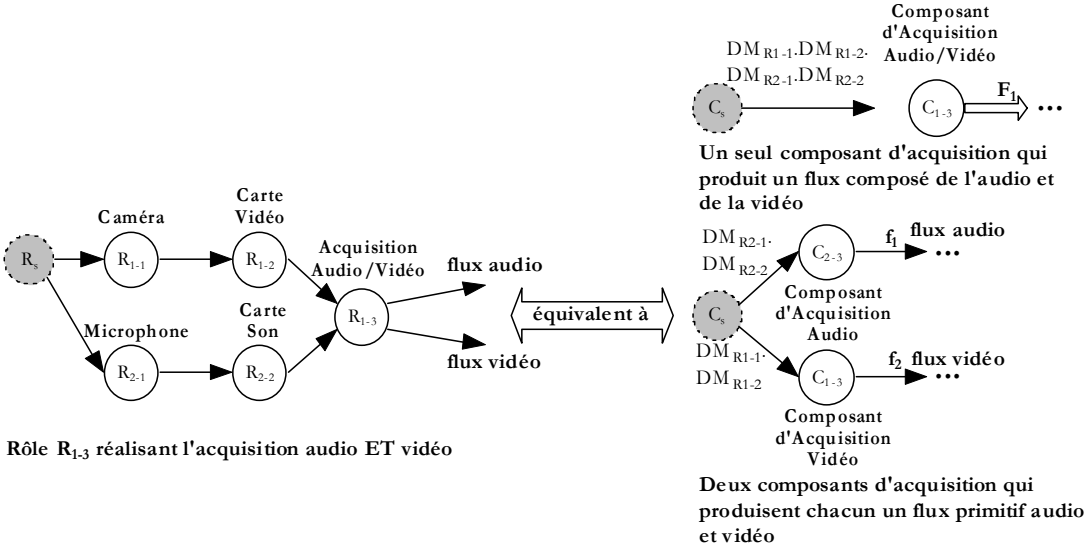


Figure 55 Représentation des rôles non atomiques

3.1.1.1.2 Accès aux Données

Le CM nécessite un environnement d'exécution qui assure sa supervision et lui fournit un accès aux données en vue de leur traitement. Pour ce faire, il est exécuté à l'intérieur d'un PE qui lui propose ses propriétés non-fonctionnelles sous la forme de services qui vont permettre de récupérer les données issues de l'extérieur mais aussi de fournir les données traitées par le CM à l'extérieur. Nous verrons dans le PE que ces communications sont réalisées à l'aide des ports d'entrée et de sortie. Ces ports sont gérés à l'intérieur du PE à l'aide de modules appelés respectivement unité d'entrée (UE) et unité de sortie (US). L'UE et l'US implémentent des interfaces qui proposent des opérations permettant respectivement de lire et d'écrire des données en entrée et en sortie du CM. Grâce à elles, chaque CM va pouvoir récupérer des données issues des flux synchrones afin de leur appliquer le traitement qu'il implémente puis fournir les flux traités en sortie. De façon générale, la tâche d'un CM se résume à des phases de lecture grâce à l'UE, à des phases de traitement des données, puis à des phases d'écriture sur les ports de sortie avec l'US. Bien entendu, le CM peut effectuer plusieurs lectures sur un ou plusieurs flux si le traitement le nécessite. Par contre, il ne fournit en sortie qu'un seul flux synchrone. Ce principe de fonctionnement est repré-

senté sur la Figure 56. Elle donne une allure générale d'implémentation des CM. Un traitement commence toujours par des phases de lecture suivies de traitement puis éventuellement de nouvelles phases de lecture et/ou des phases d'écriture. Comme on peut le voir sur la Figure 56, les phases de lecture et d'écriture peuvent être entrelacées dans le code du CM. Nous essayons de la sorte de ne pas trop contraindre l'implémentation des traitements. On peut, par exemple, disposer de CM qui commencent par traiter des données et qui en milieu du traitement ont besoin de données supplémentaires pour poursuivre.

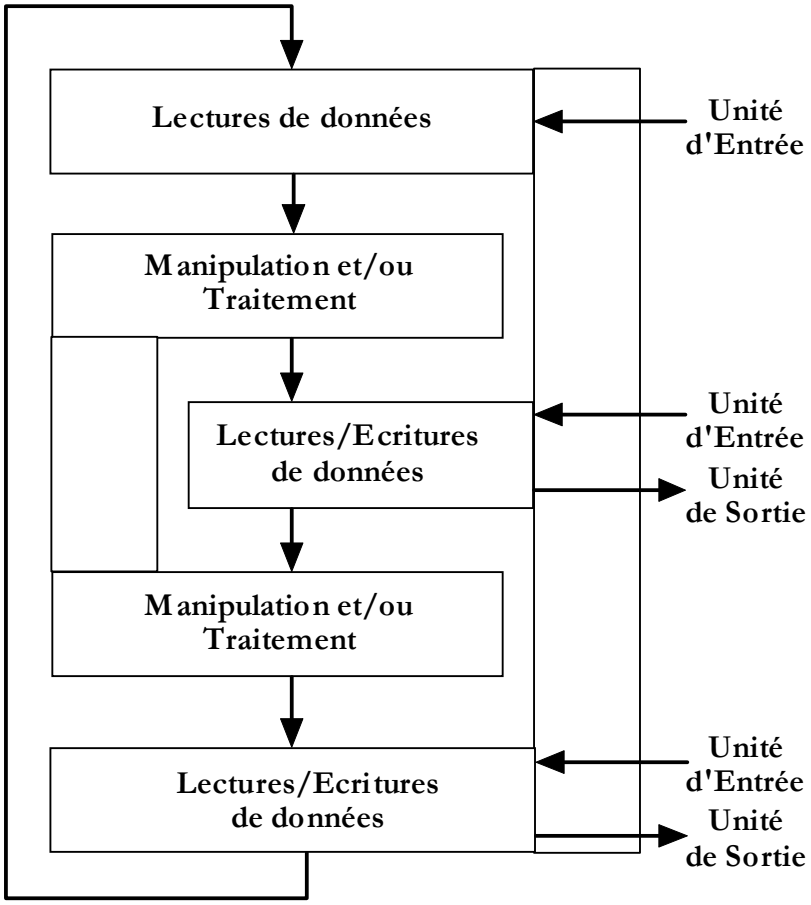


Figure 56 Principe Général de Fonctionnement du Composant Métier

Il existe cependant des cas particuliers de CM qui n'effectuent que des phases de lecture ou que des phases d'écriture. Ils sont destinés à la réalisation de fonctionnalités comme la production (acquisition/création) ou la consommation (restitution/stockage) de flux synchrones.

Un CM est capable de traiter un ou plusieurs flux de données appartenant éventuellement à un ou plusieurs flux composés. Il requiert les échantillons issus des flux de données pour leur appliquer son traitement. Les échantillons sont encapsulés dans

des unités d'information qui elles-mêmes sont encapsulées dans des tranches synchrones. Aux tranches synchrones sont associées des étiquettes temporelles qui traduisent le facteur temps des flux synchrones. Ces étiquettes sont relatives à l'horloge physique locale à un site et permettent la conservation des relations temporelles intra- et inter-flux. Chaque ensemble d'échantillons contenu dans les unités d'information possède un numéro qui permet de restituer l'ordre des échantillons dans une même tranche synchrone. Lorsqu'un composant requiert des données à traiter par l'intermédiaire des opérations de lecture, il reçoit des unités d'information associées aux étiquettes temporelles des tranches auxquelles elles appartiennent. Cet ensemble est appelé une unité temporelle. Ainsi, le CM reçoit les données qu'il doit traiter sous cette forme. En effet, certains traitements peuvent nécessiter de connaître les étiquettes temporelles et les numéros de séquence (c'est le cas par exemple des composants de restitution synchrone de flux) ou même les autres données contenus dans les unités d'information. A travers le concept d'unité temporelle, nous permettons au CM d'accéder à ces informations. En revanche, le CM ne peut pas modifier ces informations car il peut écrire en sortie uniquement les échantillons qu'il a traités. Il n'est pas du ressort du CM de gérer les attributs des unités d'information, cette tâche est entièrement confiée au PE. L'objectif est de préserver des informations cohérentes et utilisables par toutes les entités du modèle.

3.1.1.1.3 La Gestion du Cycle de Vie du Composant Métier

Le cycle de vie du CM est contrôlé à l'aide d'une interface appelée Controle-Composant qui permet d'assurer son pilotage à l'intérieur du PE. Indirectement, la plate-forme pourra contrôler son exécution par l'intermédiaire de l'UC. Cette interface est décrite sur la Figure 57, elle est requise par l'UC du PE afin d'assurer ce contrôle par la plate-forme d'exécution. Elle se compose de six opérations dont nous donnons le comportement dynamique à l'aide de contrats précisés sur la Figure 58. Ces contrats détaillent le niveau sémantique de ces interfaces par la description des intentions des opérations en s'appuyant sur des préconditions et des postconditions [BEU99], [BEU05] (cf. chapitre 2). Ainsi, pour chaque opération nous donnons les pré- et post-conditions que leur implémentation devra respecter.

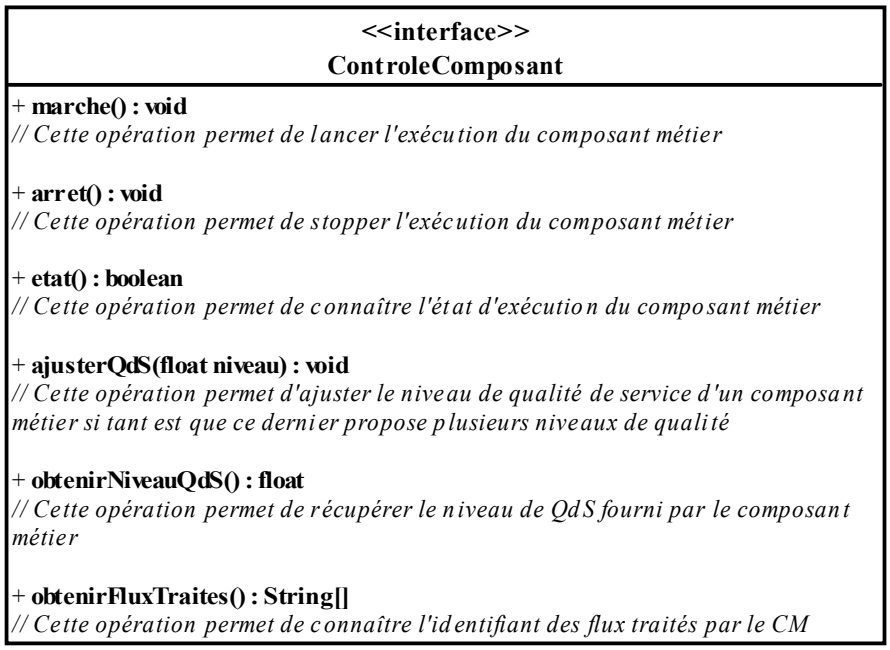


Figure 57 Interface ControleComposant

Contrat CCC1 : Mise en marche du Composant Métier

Opération : marche() : void
Préconditions :
- le composant doit être arrêté

Postconditions :
- le composant est mis en marche

Contrat CCC2 : Arrêt du Composant Métier

Opération : arret() : void
Préconditions :
- le composant doit être en état de marche

Postconditions :
- l'exécution du composant est stoppée

Contrat CCC3 : Donne l'état d'exécution du Composant Métier

Opération : etat() : boolean
Préconditions :
- ∅

Postconditions :
- retourne l'état d'exécution du Composant Métier

Contrat CCC4 : Permet d'ajuster la QdS du Composant Métier

Opération : ajusterQdS(float niveau) : void
Préconditions :
- ∅

Postconditions :
- le composant, s'il est paramètrable, devra s'exécuter avec le niveau de qualité spécifié, sinon il continuera son exécution normalement sans tenir compte de l'invocation de cette opération

Contrat CCC5 : Permet d'obtenir le niveau de qualité fourni pa le Composant Métier

Opération : obtenirNiveauQdS() : float
Préconditions :
- ∅

Postconditions :
- retourne le niveau de qualité fourni par le Composant Métier

Contrat CCC6 : Permet d'obtenir l'ensemble des flux traités par le Composant Métier

Opération : obtenirFluxTraites() : String[]
Préconditions :
- ∅

Postconditions :
- retourne l''ensemble des flux traités par le Composant Métier

Figure 58 Contrats de l'Interface ControleComposant

L'implémentation des opérations de l'interface ControleComposant, c'est-à-dire de la partie métier, est laissée à la charge du concepteur de CM afin qu'il puisse lui donner le comportement voulu tout en respectant, bien entendu, les contrats de chaque opération. marche() et arret() permettent de contrôler l'exécution du CM. Ainsi, il peut être démarré ou arrêté. Cet état d'exécution peut être connu à tout moment grâce à l'opération etat() qui donne cette information.

Enfin, l'opération obtenirFluxTraites() permet de connaître l'identifiant des flux que le CM est sensé traiter. Ces informations sont nécessaires à l'UC afin qu'elle procède à des initialisations des modules du PE lors de sa construction. Ces informations doivent être renseignées par le concepteur du CM.

3.1.1.1.4 Ajustement de la QdS des Composants Métier

Dans un souci de gestion de la QdS dans les AMD, la possibilité est laissée aux concepteurs de CM de définir et d'implémenter des composants paramétrables. Un CM paramétrable est susceptible de fournir différents niveaux de qualité pour une même fonctionnalité. L'opération ajusterQdS(float niveau) de l'interface Controle-Composant (cf. Figure 57) permet, si le CM est conçu pour, d'ajuster la QdS qu'il fournit. Le niveau de qualité désiré est spécifié par le paramètre niveau de cette opération. Nos travaux antérieurs ont permis de définir une plate-forme d'exécution ainsi qu'un modèle de QdS pour les AMD [LAP06]. Ainsi, chaque entité fonctionnelle qui compose une AMD est évaluée selon deux critères de QdS : un critère intrinsèque et un critère contextuel[58]. Le critère intrinsèque regroupe les caractéristiques de QdS d'un CM indépendamment de son contexte d'exécution. Il permet de traduire l'adéquation de ce dernier d'un point de vue fonctionnel. Il prend différentes valeurs si un CM est à même de fournir différentes qualités pour le rendu d'une même fonctionnalité. Le critère contextuel, pour sa part, regroupe les caractéristiques de QdS dépendantes du contexte d'exécution. Il traduit l'influence de ce dernier sur le fonctionnement des CM d'une AMD. Le critère contextuel est évalué par la plate-forme d'exécution tandis que le critère intrinsèque est fourni par le CM. Chaque concepteur de CM doit donc être capable de fournir une note ou un ensemble de notes traduisant le service fourni. La QdS d'un CM est représentée selon deux dimensions en octroyant une note appelée In pour le critère intrinsèque et une note appelée Co pour le critère contextuel. Ainsi, la QdS est représentée par un point P_{QdS} dont les coordonnées dans le plan des QdS sont ces deux notes : P_{QdS}=(Co, In). Le plan des QdS est donné sur la Figure 59. Chaque critère est représenté par une note comprise entre 0 et 1. Ainsi, la note 0 correspond à un niveau de qualité médiocre tandis que la note 1 correspond à un niveau de qualité idéal. Sur la Figure 59, nous avons représenté les points correspondants à la QdS idéale, à la QdS médiocre pour un CM ainsi qu'un point évalué à l'instant t_0.

---

[58] Les critères de QdS sont introduits pour caractériser les entités fonctionnelles des AMD. Ainsi, le modèle de QdS propose de caractériser les entités formées par un CM, un assemblage de CM, voire même une AMD. Nous nous intéressons ici à ces critères par rapport à l'entité fonctionnelle qu'est le CM.

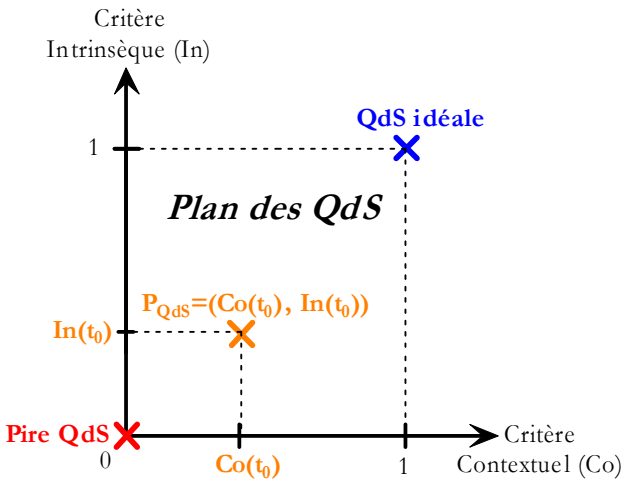


Figure 59 Représentation de la QdS [LAP06]

Dans le cas de CM paramétrables, l'opération ajusterQdS(float niveau) permet de modifier le critère intrinsèque et donc de jouer sur la QdS fournie. Ces niveaux sont choisis par la plate-forme en fonction des critères contextuels qu'elle est capable d'évaluer, l'objectif étant toujours de rapprocher la QdS fournie par les AMD de celle requise par les utilisateurs. Suivant les cas, la plate-forme choisira d'augmenter ou de dégrader la QdS intrinsèque des CM le permettant. Cette possibilité est une des formes de reconfiguration auxquelles elle peut faire appel pour gérer la QdS. Certaines des configurations possibles exprimées par les arcs conditionnels sur les graphes de transition peuvent être matérialisées par des CM paramétrables. Afin d'accompagner ce critère, les entités fonctionnelles du modèle Osagaia sont capables de délivrer des informations (états) qui vont apporter à la plate-forme une aide dans sa mission de diagnostic.

Le paramètre niveau de l'opération ajusterQdS(float niveau) est un réel qui représente ce critère intrinsèque. Nous représentons sur la Figure 60, un exemple de CM paramétrable qui fournit quatre niveaux différents de qualité. Ce CM est destiné à appliquer des traitements sur un flux vidéo. Il est capable de réduire la taille initiale de la vidéo et éventuellement de transformer le flux couleur en noir et blanc. Son utilisation peut permettre de réagir à des variations de bande passante du réseau ou encore de prendre en compte des capacités d'affichage réduites (terminaux mobiles). Sur la Figure 60, nous avons représenté un plan des QdS afin de situer chaque niveau fourni par le CM. Ainsi, la QdS globale de ce CM dépend évidemment du critère contextuel et donc chaque point peut se déplacer horizontalement sur le plan en fonction des valeurs de ce dernier. D'autres exemples de représentation de la QdS sont disponibles dans [LAP06]. Le bas de la Figure 60 détaille les différents niveaux fournis par le CM en terme de qualité et de taille de l'image. La QdS idéale correspond à l'image repré-

sentée en haut à gauche. Elle traduit un niveau intrinsèque compris entre 0.75 et 1 et peut être utilisée lorsqu'un contexte satisfaisant est disponible.

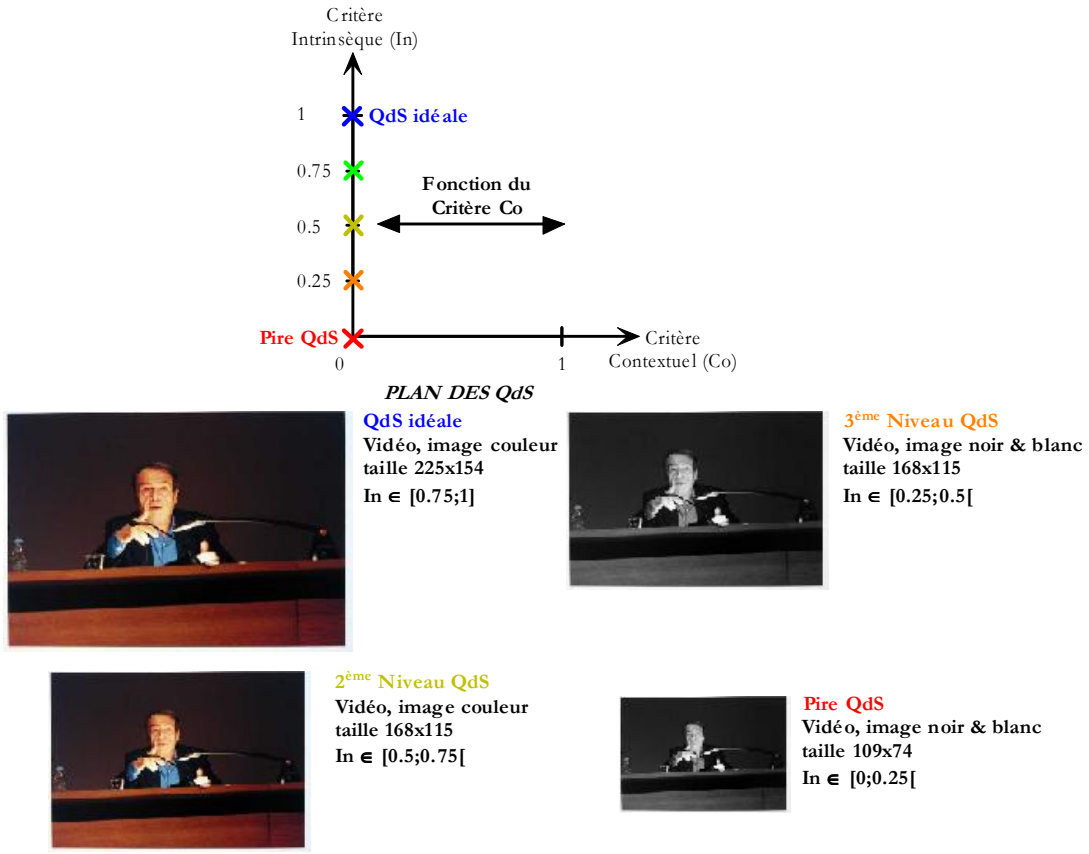


Figure 60 Différents niveaux de qualité pour un CM paramétrable

3.1.1.2 Le Processeur Elémentaire

Le PE est un conteneur de CM dont l'objectif est de proposer à ce dernier des propriétés non-fonctionnelles nécessaires à son fonctionnement et au fonctionnement global d'une AMD. La Table 7 distingue deux types de propriétés, celles qui ont trait à la gestion des entrées/sorties et celles qui relèvent du contrôle du cycle de vie du CM et de l'ensemble du PE. En conséquence, nous définissons une architecture pour le PE basée sur deux modules dont chacun est responsable d'une classe de propriétés non-fonctionnelles :

- l'unité d'échange se charge de la gestion des entrées/sorties. Elle se divise en deux unités (UE et US) qui ont la responsabilité de la gestion des connexions en entrée et en sortie du PE.
- l'unité de contrôle (UC) fournit les moyens nécessaires au contrôle de l'exécution d'un CM mais aussi de l'ensemble du PE. Elle sert également

d'interface entre le PE et la plate-forme d'exécution. Ainsi, chaque PE d'une AMD et par conséquent chaque CM peut être supervisé.

Le schéma de principe du PE est donné sur la Figure 61. Les communications entre les différents modules du PE sont réalisées à l'aide d'interfaces (cf. chapitre 2). Les PE qui composent une AMD sont connectés entre eux à l'aide de conduits chargés d'assurer l'acheminement des flux synchrones. Le conduit est l'entité fonctionnelle du modèle chargée de transporter les flux synchrones à raison d'un flux par conduit. Sur la Figure 61, le premier conduit contient deux flux de données qui en fait font partie du même flux composé. Il en est de même pour le conduit de sortie. Tout conduit connecté à un PE contient au moins un flux de données destiné à être traité par le CM. En revanche, tous les flux de données qu'ils contiennent ne sont pas nécessairement lus par le CM. Les PE supportent des CM implémentant des rôles atomiques. Par conséquent, un PE accepte un ou plusieurs conduits en entrée et un seul conduit en sortie comme sur la Figure 61 et pour les raisons évoquées dans le § 3.1.1.1.1. Les connexions entre les conduits et le PE sont réalisées à l'aide de ports d'entrée/sortie[59] qui sont gérés par l'unité d'échange et dont le fonctionnement est détaillé dans la section suivante.

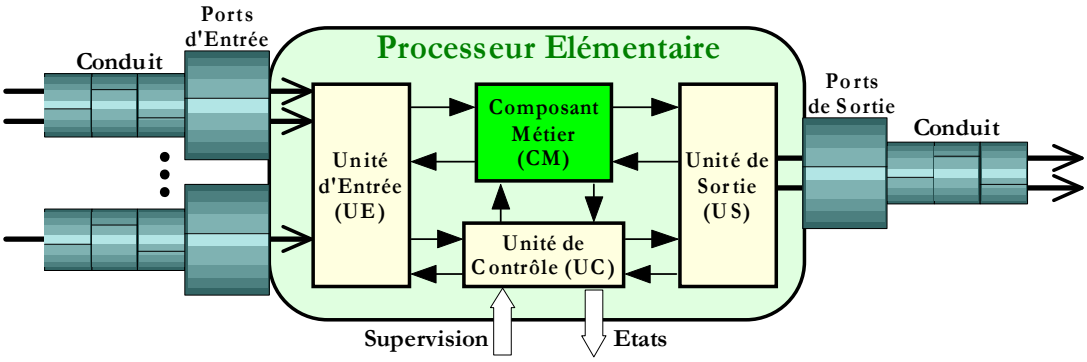


Figure 61 Architecture du Processeur Elémentaire

Comme on peut le voir sur cette figure, le nom de Processeur Elémentaire a été choisi pour ce conteneur car son architecture interne rappelle celle des entités qui composent les architectures matérielles dites à flots de données ou encore cadencées par les données[60] [CAR04], [KAP03]. Dans ces architectures, les applications sont également décrites par des graphes où chaque nœud correspond à un traitement. Les traitements sont déclenchés au fur et à mesure que les données se propagent dans une

[59] Les conduits possèdent également des ports d'entrée/sortie qui leur permettent de se connecter aux PE. Nous ne les avons pas détaillé sur la Figure 61 afin de ne pas trop surcharger le schéma.

[60] En anglais data driven.

telle architecture. Ce type de séquencement permet d'obtenir une synchronisation implicite des traitements[61] [LEE94]. Ce type d'architecture est utilisé en particulier pour le calcul parallèle distribué car il permet une réduction des temps de calcul.

Outre son rôle de conteneur pour le CM, le PE assure deux autres tâches indispensables aux AMD :

- il conserve la synchronisation des flux composés qui transitent par lui. En effet, un flux composé est constitué de flux de données liés par des relations de synchronisation inter-flux. Le rôle du CM est de traiter certains de ces flux mais, bien entendu, pas forcément la totalité d'entre eux. Par exemple, un CM faisant du traitement d'images pourra recevoir un flux composé contenant les images, le son associé et des sous-titres. Bien que le CM ne produise qu'un flux d'images en sortie, celui-ci reste synchrone des flux de son et de sous-titres auxquels il était initialement lié. Tous les flux composés parvenant à un PE sont synchrones et il appartient au PE de faire en sorte que cette synchronisation soit conservée malgré les traitements effectués par le CM qu'il contient. Nous verrons que cela suppose de savoir faire transiter dans le PE les flux non traités (sans qu'ils passent par le CM) et de reconstituer en sortie des tranches synchrones à partir de ces flux et de ceux produits par le CM.
- les AMD sont supervisées par une plate-forme dont le rôle est d'évaluer la qualité des services offerts et de proposer des réorganisations de l'application par ajout, suppression, et déplacement de composants. Pour ce faire, la plate-forme a besoin d'informations sur le fonctionnement interne de l'application et doit pouvoir transmettre des ordres aux composants qui la constituent. En tant que conteneur du CM, le PE peut évaluer son fonctionnement et en particulier la QdS liée au contexte. Ainsi, en surveillant l'écoulement des flux au travers du CM, il peut détecter des problèmes d'inadéquation du CM au contexte lorsque, par exemple, celui-ci ne parvient pas à traiter en temps réel les flux qui lui sont envoyés (ces informations peuvent être collectées en effectuant des mesures en entrée du PE). De plus, le PE connaît l'état d'exécution du CM et peut le modifier c'est-à-dire l'arrêter ou le démarrer. Enfin, il assure sa connexion et sa déconnexion aux conduits d'entrée et de sortie permet-

[61] L'architecture logicielle que nous définissons est également basée sur ces principes.

tant ainsi une réorganisation de l'architecture interne de l'application sous le contrôle de la plate-forme.

3.1.1.2.1 L'unité d'échange

L'unité d'échange est composée de l'UE et de l'US et a pour rôle la gestion des entrées et des sorties du PE. Elle est chargée de réceptionner les flux synchrones provenant du ou des conduits connectés en entrée et de fournir en sortie du PE un flux synchrone à destination du conduit connecté en sortie. Schématiquement, les flux de données destinés à être traités sont récupérés par le CM. Les autres, lorsqu'ils font partie d'un flux composé, ne font que transiter dans le PE en synchronisme avec les flux traités. En fait, l'unité d'échange est chargée de séparer les flux composés[62] et d'envoyer chaque flux de données vers sa destination à l'intérieur du PE c'est-à-dire vers l'US pour les flux qui ne font que transiter par le PE et vers le CM pour ceux qui doivent être traités. En sortie, elle reconstitue le flux composé à partir des flux de données ayant transité et de ceux produits par le CM. Elle utilise pour ce faire les politiques de synchronisation définies par le modèle Korrontea. Enfin, l'unité d'échange est chargée de mettre à jour les attributs des tranches synchrones afin, suivant les types de traitement, de conserver la cohérence des informations.

Les flux synchrones transitent des conduits vers le PE à l'aide de dispositifs appelés respectivement ports d'entrée et ports de sortie. Ces ports assurent la connexion des conduits en entrée et en sortie (cf. Figure 61). Avant de donner les fonctionnements complets de l'UE et de l'US, nous allons nous attarder sur ce concept de port.

Les ports d'entrée/sortie

La version 2 du langage de modélisation UML introduit le concept de port. Un port y est défini comme un dispositif qui fournit une fonctionnalité à partir d'une structure composite sans avoir à exposer les détails internes de sa réalisation [PIL06]. L'implémentation de ces fonctionnalités est réalisée à l'aide d'un ensemble de classes qui représentent la structure interne du port. La fonctionnalité d'ensemble, ainsi définie, est décrite sous la forme d'un port. L'avantage de réaliser une fonctionnalité au moyen d'un port est que la structure interne qu'il représente peut être utilisée par un

---

62 Pour ce qui est des flux primitifs, l'unité d'échange a pour rôle d'extraire le flux de données qu'ils contiennent et de l'envoyer au CM. Si celui-ci produit un seul flux en sortie alors le flux primitif est reconstitué. Par contre, si le CM produit plusieurs flux en sortie, un flux composé est alors constitué par application des politiques de synchronisation de Korrontea.

autre classificateur[63] se conformant à ses spécifications. Dans le modèle Osagaia, les ports sont utilisés par toutes les entités connectables c'est-à-dire les conduits, les PE et les opérateurs. On assure ainsi leur connexion mais aussi le transfert des flux entre elles. Chaque port est associé à des interfaces fournies et requises.

Les ports implémentent des zones tampons utilisées pour le stockage temporaire des tranches issues des flux synchrones. Ces zones sont unitaires c'est-à-dire qu'elles sont capables de stocker une seule tranche synchrone à la fois. Ainsi, les ports reçoivent indifféremment des flux composés ou des flux primitifs. Nous utilisons un port d'entrée par flux synchrone connecté en entrée à l'aide d'un conduit et un seul port de sortie puisque la sortie d'un PE fournit un et un seul flux synchrone (cf. § 3.1.1.1.1). Grâce aux ports d'entrée, l'UE va pouvoir lire des données afin de les transférer au CM ou bien de les faire transiter dans le PE vers l'US. De même, grâce au port de sortie, l'US va écrire les tranches qu'elle a reconstituées à l'aide des données provenant du CM et de l'UE. Le stockage unitaire impose que les ports récupèrent une à une les tranches des flux synchrones connectés en entrée et les stockent en attente de leur transfert. De la même façon, le port de sortie récupère une à une les tranches destinées à être transférées par le conduit de sortie[64].

Les ports implémentent une interface nommée AccesPort. Elle propose un ensemble d'opérations qui permet la lecture et l'écriture de tranches synchrones dans un port. Elle est décrite sur la Figure 62. Elle contient également une opération qui permet de vider un port et une opération qui permet de connaître l'état d'un port c'est-à-dire de savoir s'il est vide ou non. Les spécifications de ces opérations sont données par les contrats définis sur la Figure 63. On spécifie, par exemple, le caractère unitaire des ports. Cette caractéristique implique de disposer d'opérations d'écriture et de lecture bloquantes afin de pouvoir lire et écrire dés que les ports le permettent. Ces propriétés sont également définies par les contrats. Afin d'éviter des périodes de blocage importantes, l'interface AccesPort fournit une opération estVide() qui permet de savoir si un port est vide ou non. Elle peut donc être utilisée avant toute tentative de lecture et d'écriture sur un port. Les ports qui stockent des flux à contrainte temporelle faible sont particulièrement sensibles à ce type de problème. En effet, sur ce type de flux les tranches ne sont pas présentes de façon régulière et des temps non négli-

---

63 En UML, un classificateur est un élément de modèle qui décrit des caractéristiques structurelles et comportementales. Il constitue une généralisation de nombreux éléments d'UML comme les classes, les interfaces, les cas d'utilisation et les acteurs [OMG03].

64 Nous allons voir qu'il est possible de définir des PE sans UE ou sans US. Ces possibilités impliquent donc que ces PE n'auront pas de ports en entrée ou pas de port de sortie. Le fonctionnement est similaire sauf que dans ces cas particuliers les PE produisent ou consomment des tranches synchrones.

geables peuvent exister entre deux tranches consécutives. Il faut donc, lorsque ces cas se présentent, empêcher le CM de rester bloquer à l'attente d'une donnée sur ce type de flux. Dans ce sens, nous proposons cette alternative.

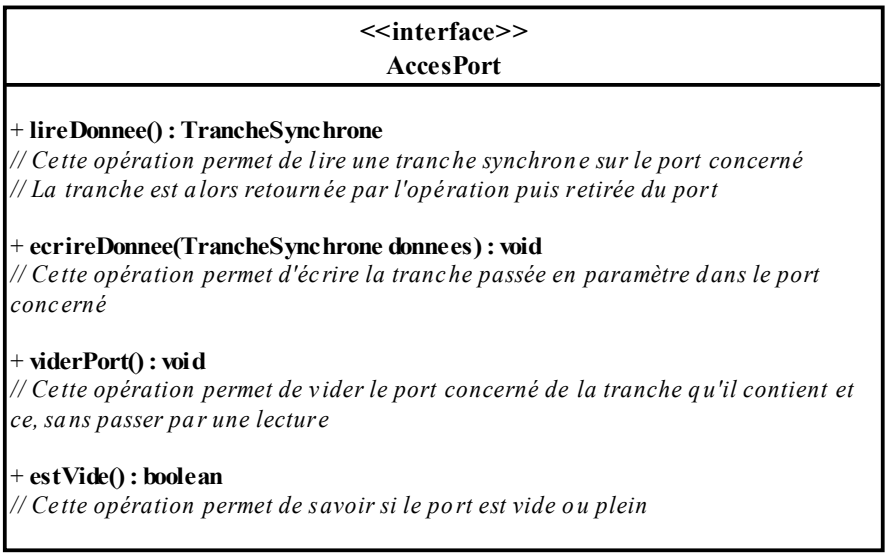


Figure 62 Interface AccesPort

Contrat CAP1 : Lire une tranche sur un port

Opération : lireDonnee() : TrancheSynchrone
Préconditions :
- ∅
Postconditions :
- l'opération se bloque jusqu'à ce que le port possède une tranche

Contrat CAP2 : Ecrire une tranche sur un port

Opération : ecrireDonnee(TrancheSynchrone donnees) : void
Préconditions :
- ∅
Postconditions :
- l'opération se bloque jusqu'à ce que le port soit vide et donc jusqu'à ce que la tranche passée en paramètre puisse être écrite

Contrat CAP3 : Vider le port

Opération : viderPort() : void
Préconditions :
- ∅
Postconditions :
- le port est vide

Contrat CAP4 : Récupérer l'état du port

Opération : estVide() : boolean
Préconditions :
- ∅
Postconditions :
- retourne VRAI ou FAUX suivant que le port est vide ou non

Figure 63 Contrats de l'Interface AccesPort

Le principe de connexions des entités du modèle à l'aide des ports est simple à comprendre (cf. Figure 61). Les ports de sortie sont connectés aux ports d'entrée et vice versa. Le transfert des tranches synchrones entre les ports ainsi connectés est réalisé à l'aide d'un mécanisme de type interface fournie et interface requise par l'intermédiaire de l'interface AccesPort (cf. Figure 62). Ainsi, chaque port de sortie fournit cette interface qui est requise par chaque port d'entrée et réciproquement. Ce principe de connexion est représenté sur la Figure 64 entre des conduits en entrée d'un PE. Les flux synchrones transitent au sein d'une AMD des ports de sortie vers les ports d'entrée de chaque entité.

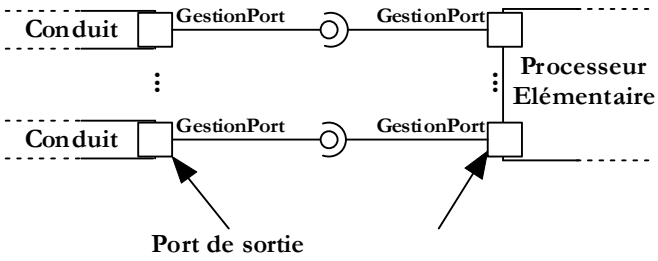


Figure 64 Connexion entre des Conduits et un Processeur Elémentaire

Dans un PE, les ports d'entrée et de sortie sont gérés respectivement par l'UE et par l'US.

L'unité d'entrée

L'UE est chargée de la gestion des connexions en entrée du PE. Sa fonction est de récupérer les données issues des flux synchrones dans les ports d'entrée afin de les aiguiller à travers le PE. Ainsi, les flux destinés à être traités par le CM sont récupérés par ce dernier tandis que les autres, s'il y en a, transitent de l'UE vers l'US. Le CM utilise une interface dédiée à la récupération des tranches sur les flux synchrones qu'il doit traiter. De plus, l'interface ControleComposant (cf. Figure 57) propose l'opération obtenirFluxTraites() qui permet de connaître l'ensemble des flux traités par un CM.

Ainsi, l'UE est capable de récupérer toutes les tranches issues des flux synchrones se trouvant dans les ports d'entrée. Pour ce faire, elle doit avoir une vision claire de tous les flux de données (à l'intérieur des flux primitifs et/ou composés) connectés en entrée du PE. Les unités d'information possèdent un attribut (idFlux) qui permet d'identifier le flux de données auxquelles elles appartiennent (cf. chapitre 5). Cet identifiant est associé aux unités d'information par les sources localisées dont le but est de créer les flux de données. Ainsi, à l'intérieur du PE, on va pouvoir désigner explicitement les flux de données et donc les manipuler. Par exemple, le CM va pouvoir désigner explicitement les flux dans lesquels il va lire des données mais aussi ceux dans lesquels il va en écrire. Une première étape dans l'identification des flux est établie par les graphes de transition.

Lorsqu'un PE reçoit des flux composés, cela signifie que le CM traite au moins un flux de données appartenant à ces flux. Les tranches reçues dans les ports adéquats sont récupérées et séparées en plusieurs flux primitifs par l'UE. Cette opération est rendue possible grâce aux identifiants des flux de données et donc à la vision que l'UE a de ces derniers. Pour ce faire, on utilise l'opérateur de séparation que nous avons introduit précédemment avec les graphes de transition et que l'on applique à chaque flux composé. Cet opérateur fournit en sortie un ensemble de flux primitifs où chacun

contient un flux de données qui rentrait dans la composition du flux composé initiale. Ces flux sont alors stockés au sein de l'UE. Après quoi, l'UE est capable de rediriger ces flux primitifs vers leur destination respective. Schématiquement, les k flux primitifs traités sont envoyés au CM, les n-k autres transitent de l'UE vers l'US avec n le nombre total de flux primitifs. Le transfert des flux à traiter vers le CM est à l'initiative de ce dernier. Le CM récupère des unités temporelles telles que nous les avons décrites dans le § 3.1.1.1.2. Ces unités sont constituées à l'aide des tranches synchrones des flux primitifs. Lorsque le CM effectue la dernière lecture de l'unité temporelle de la tranche en cours, les transferts de l'UE vers l'US sont alors déclenchés. Si le CM lit dans un flux à contrainte temporelle faible qui ne possède pas d'unités pour cette tranche alors le transfert est immédiatement réalisé vers l'US. La Figure 65 reprend ces mécanismes. Les tranches synchrones des flux composés connectés en entrée du PE sont réceptionnées par l'UE qui les sépare en flux primitifs et aiguille les tranches obtenues vers leur destination respective. Nous verrons qu'en sortie l'US est chargée de reconstruire les tranches synchrones en appliquant les politiques de synchronisation du modèle Korrontea. Sur cette figure, nous avons schématisé les flux composés en détaillant leur composition. Les lectures effectuées par le CM concernent les flux de données rentrant dans la composition des flux composés. Il est nécessaire de bien garder à l'esprit que les CM reçoivent des unités temporelles que nous n'avons pas représenté sur le schéma de principe suivant.

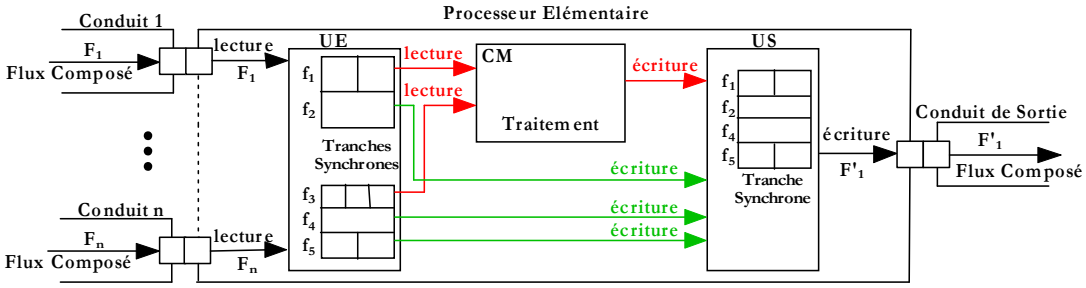


Figure 65 Principes de Fonctionnement de l'unité d'échange

Lorsqu'en entrée d'un PE, on dispose uniquement de flux primitifs, les tranches de ces derniers sont directement transmises vers leur destination respective.

Une zone de stockage temporaire est utilisée par l'UE pour stocker les tranches des flux primitifs. La zone est découpée en section à raison d'une section par flux primitif. Chaque section est désignée par l'identifiant du flux. Les flux sont rassemblés en fonction de leur destination. Cette zone de stockage est définie, en partie, lors de l'initialisation de l'UE.

Le CM requiert des données pour exécuter le traitement qu'il implémente. L'accès à ces dernières est proposé par l'UE au CM à l'aide d'une interface fournie dé-

crite sur la Figure 66. Elle se nomme LectureFlux et propose deux opérations qui permettent de lire des unités temporelles (cf. § 3.1.1.1.2) sur un flux primitif[65] stocké dans la zone dédié par l'UE en passant l'identifiant de ce dernier en paramètre. La première opération tentativeLecture(String idFlux) permet de procéder à une tentative de lecture sur un flux primitif. Cette opération n'est pas bloquante et donc se termine quoi qu'il arrive : soit en retournant l'unité temporelle, soit en retournant un résultat nul signifiant que les données ne sont pas disponibles. L'UE fournit cette possibilité afin qu'elle puisse être utilisée dans des types d'implémentation où le CM ne doit pas rester bloqué sur la lecture d'un flux. Par exemple, lorsqu'il doit lire des données sur un flux à contrainte temporelle faible. La seconde opération lectureUT(String idFlux) est, à l'inverse, bloquante c'est-à-dire que le résultat de cette dernière est l'unité temporelle requise. L'opération se bloque alors jusqu'à ce que l'unité soit disponible. Cette opération peut être utilisée sans risques lorsque l'on lit des données sur des flux à contrainte temporelle forte. De même, on peut l'utiliser si le CM lit des données sur un unique flux à contrainte temporelle faible. En effet, il n'est pas gênant dans ce cas de bloquer l'exécution du CM tant que de nouvelles données ne sont pas arrivées. Ces deux opérations permettent au concepteur du CM de gérer les lectures suivant les comportements qu'il veut donner à l'implémentation fournie. Le paramètre idFlux est une chaîne de caractères (String) qui permet de désigner le flux de données (contenu dans un flux primitif) dans lequel le CM veut aller lire une unité temporelle.

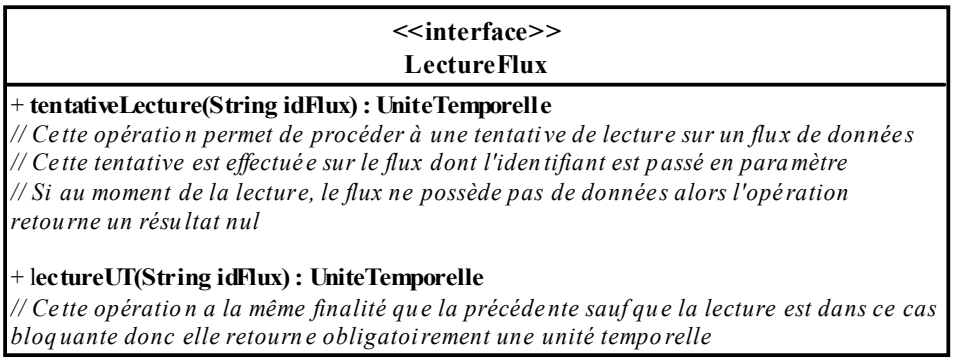


Figure 66 Interface LectureFlux

Lorsqu'un CM traite plusieurs flux ou qu'il nécessite plusieurs unités pour un même flux, il doit effectuer plusieurs opérations de lecture. La Figure 67 décrit les contrats des opérations de l'interface LectureFlux. Les préconditions pour ces deux opérations

[65] Un CM est capable de lire des données au sens large du terme uniquement sur des flux primitifs puisque les unités temporelles qu'il récupère sont composées d'une unité d'information associée à une étiquette temporelle. D'ailleurs, les opérations de lecture de l'interface LectureFlux (cf. Figure 66) ne sont définies que pour ces derniers. Nous verrons que dans l'interface proposée par l'US, le CM est capable de fournir en sortie uniquement des flux primitifs.

sont les mêmes, il est nécessaire que le flux désigné par idFlux existe. Les postconditions décrivent suivant le cas, une lecture bloquante ou pas.

Contrat CLF1 : Tentative de Lecture d'une Unité Temporelle issue d'un flux de données

Opération : tentativeLecture(String idFlux) : UniteTemporelle
Préconditions :
- le flux identifié par idFlux existe

Postconditions :
- si le flux ne possède pas d'unités à l'instant de la lecture alors l'opération retourne un résultat nul
- dans le cas contraire, l'unité temporelle est retournée par l'opération

Contrat CLF2 : Lecture d'une Unité Temporelle issue d'un flux de données

Opération : lectureUT(String idFlux) : UniteTemporelle
Préconditions :
- le flux identifié par idFlux existe

Postconditions :
- l'opération se bloque jusqu'à ce qu'une unité temporelle soit disponible sur ce flux

Figure 67 Contrats de l'Interface LectureFlux

Les flux prépondérants indiqués sur les graphes de transition sont des paramètres des composants puisqu'ils déterminent l'importance d'un flux dans un traitement. Ainsi, un PE possède le même paramètre. La présence ou l'absence de flux prépondérant pour un PE est signifiée lors de sa déclaration.

L'UE permet également de récupérer des mesures qui vont permettre d'interpréter la qualité de fonctionnement du CM qui s'exécute dans un PE. Ces métriques corrélées à d'autres vont permettre à la plate-forme d'exécution de déduire le critère contextuel de QdS du CM et donc de pouvoir tirer des conclusions sur le fonctionnement d'une configuration et d'identifier des anomalies de fonctionnement dans une AMD. La plate-forme reçoit ces mesures sous la forme d'événements envoyés par l'UC[66]. Elles constituent les états de fonctionnement du CM. On va ainsi pouvoir détecter qu'un CM « suit » ou pas ou que les données transmises sont inadaptées à la bande passante du réseau ou à la charge du processeur de la machine sur lequel s'exécute le CM. En fonction de ces informations, la plate-forme pourra ajuster la QdS fournie en ces points soit par reconfiguration de l'AMD, soit par paramétrisation des CM concernés si tant est que ces derniers fournissent cette solution.

---

[66] Nous rappelons que l'UC est un module du PE qui permet d'assurer toute la partie contrôle et supervision de l'ensemble PE et CM.

Enfin, l'UE participe également à assurer la synchronisation des entités du modèle c'est-à-dire entre les PE, les conduits et les opérateurs afin que le transfert des flux synchrones entre ces entités s'effectue correctement de bout en bout d'une AMD.

L'unité de sortie

De la même manière que l'UE gère les connexions en entrée du PE, l'US supervise les connexions en sortie. Elle a pour objectif de récupérer les données produites par le CM ainsi que celles qui ne font que transiter de l'UE vers l'US et de les proposer au conduit connecté en sortie par l'intermédiaire du port de sortie. Une autre fonction consiste à initialiser ou à mettre à jour les attributs des unités d'information dans lesquelles seront encapsulées les données traitées ou produites par le CM. De plus, lorsque l'on trouve plusieurs flux primitifs en sortie, cela signifie qu'ils font partie du même flux composé et dans ce cas, l'US est chargée de constituer les tranches synchrones de ce flux en appliquant les politiques de synchronisation du modèle Korrontea selon les types de flux (contrainte forte, contrainte faible) puis de les écrire dans le port de sortie. Lorsqu'un seul flux primitif est disponible, l'US reconstitue ses tranches et les écrit une à une dans le port de sortie.

L'US gère le port de sortie qui est le point de sortie d'un flux synchrone sur le PE puisque nous avons vu qu'un PE ne possédait en sortie qu'au plus un flux synchrone primitif ou composé. Afin d'établir la communication avec le port de sortie, l'US utilise l'interface AccesPort (cf. Figure 62). De la même manière que l'UE, l'US a une vision de tous les flux de données en sortie du PE grâce à l'identifiant des flux attribué à leurs unités d'information.

L'US gère les unités d'information des flux de données de deux façons différentes selon qu'il existe ou non un flux prépondérant en entrée du PE. Ces comportements sont intimement liés aux traitements que le CM implémente. En particulier, lorsque le CM consomme plusieurs flux il est indispensable pour conserver les relations de synchronisation inter-flux de savoir lequel de ces flux constitue la référence temporelle. La notion de flux prépondérant va également jouer un rôle dans la définition de cette référence. En effet, nous choisissons d'adopter le flux prépondérant comme référence temporelle en raison de son importance dans le traitement correspondant. La présence d'un flux prépondérant en entrée d'un PE est définie lors de la création de ce dernier.

Le premier comportement correspond au cas où le PE considéré ne possède pas de flux prépondérant en entrée ou ne possède pas d'entrées[67]. On se trouve alors dans une configuration où le CM produit un ou plusieurs flux primitifs soit en fonction de flux d'entrée, soit sans aucune entrée. Ce type de CM constitue une source localisée. Dans ce cas l'US est chargée de créer le ou les nouveaux flux en initialisant les attributs des unités d'information et des tranches synchrones et ce à chaque fois que le CM produit un ensemble d'échantillons c'est-à-dire qu'il effectue une écriture. Ces phases d'écriture sont réalisées à l'aide d'une opération proposée par une interface que nous détaillons dans la suite de cette section. L'US crée donc en sortie un ou plusieurs flux primitifs. Lorsqu'elle en crée plusieurs, ils sont rassemblés par cette dernière dans un flux composé. Chaque flux primitif est créé en constituant des unités d'information qui composeront des tranches synchrones. Une unité d'information est composée des références de la source localisée (identifiant du CM qui a créé le flux et du site où se trouve ce dernier), d'un numéro de séquence et de l'identifiant du flux de données. Ces informations sont associées à l'ensemble des échantillons produits par le flux. L'unité d'information ainsi créée est encapsulée dans une tranche synchrone qui est l'unité de transport des données dans une AMD. L'US rassemble dans la même tranche les unités d'information qui sont créées au même moment puis elle associe à la tranche une étiquette temporelle délivrée par l'horloge physique locale du site. L'US définit également le type de contrainte temporelle du flux ainsi créé. Nous allons voir que la contrainte est spécifiée à l'aide de l'opération d'écriture. Si plusieurs flux primitifs sont créés par le CM, l'US applique ensuite les politiques de synchronisation à ces flux afin de définir un flux composé.

Le second comportement de l'US est mis en œuvre lorsque le PE possède en entrée un flux prépondérant qui peut être primitif ou composé. Un tel PE exécute un CM qui traite le flux prépondérant (ou un de ses flux de données dans le cas d'un flux composé) éventuellement à l'aide d'autres flux de données contenus dans d'autres flux primitifs ou composés. Lorsque le CM effectue une lecture sur le flux prépondérant, l'UE lui délivre, si des données sont disponibles, une unité temporelle. Certaines informations contenues dans cette unité (en particulier l'identifiant du site de la source localisée et l'étiquette temporelle) sont sauvegardées au sein de l'US. Lorsque le CM écrit des échantillons qu'il vient de traiter, ces échantillons sont associés aux informations sauvegardées correspondant à la dernière unité temporelle lue sur le flux prépondérant. Ainsi, l'US reconstitue en sortie un ou plusieurs flux primitifs. Pour cha-

---

67 En effet, certains traitements ne possèdent pas d'entrées et donc pas de flux prépondérants : c'est le cas d'un CM de création de données. Dans ce cas de figure, le PE qui accueillera ce type de CM ne possèdera pas d'UE. De la même manière, lorsqu'un CM ne possède pas de sortie alors le PE qui l'accueillera ne proposera pas d'US.

que tranche, elle récupère l'étiquette temporelle sauvegardée et pour les unités d'information l'identifiant du site puis elle met à jour l'identifiant du flux et le numéro de séquence en fonction du nombre d'unités dans la tranche. On est de la sorte sûr que toutes ces informations sont cohérentes. Les étiquettes attribuées aux tranches synchrones formeront une suite a priori infinie et croissante c'est-à-dire un ordre strict. Cette méthode est intéressante car elle peut permettre de créer des relations de synchronisation inter-flux artificiels entre flux de données qui ne proviennent pas forcément de la même source localisée. Par exemple, si on veut incruster la vidéo d'un journaliste sur la vidéo d'une rue de la ville dont il parle alors on pourra utiliser cette méthode de synchronisation artificielle par les traitements. La vidéo du journaliste sera définie comme flux prépondérant, elle sera incrustée sur la vidéo du paysage. On obtiendra de fait en sortie la vidéo incrustée et les deux bandes sonores dont celles du paysage qui aura été resynchronisée par affectation des étiquettes temporelles du flux prépondérant avec l'ajout de l'identifiant du site du flux prépondérant. Ainsi, on produira un flux composé avec ces trois flux qui seront désormais considérés comme synchrones entre eux. Dans ce genre de manipulation, une synchronisation stricte des flux n'est pas nécessaire puisque ces flux à l'origine ne possédaient pas de relations de sémantique entre eux. On crée artificiellement ces relations grâce au traitement infligé par le CM.

L'US utilise une zone de stockage afin de stocker temporairement les tranches synchrones des flux primitifs qui transitent de l'UE vers l'US mais aussi celles qui viennent d'être créées lors de l'écriture de données par le CM. Cette zone se divise en sections à raison d'une section par flux primitif. Chaque section est identifiée par l'identifiant du flux de données contenu dans le flux primitif. De plus, dans cette zone, on stocke l'étiquette temporelle et l'identifiant du site qui correspondent aux informations de la dernière unité lue sur le flux prépondérant (si flux prépondérant il y a). Ces informations sont mises à jour à chaque nouvelle lecture sur ce flux. Si cette zone comporte une seule section, cela veut dire qu'un seul flux primitif transite par le PE et qui plus est par le CM. Les tranches stockées sont immédiatement écrites une à une dans le port de sortie du PE. Lorsque cette zone comporte plusieurs sections, cela veut dire que tous les flux primitifs représentés doivent rentrer dans la composition d'un flux composé. C'est pourquoi on applique en sortie de cette zone les politiques de synchronisation de Korrontea. On utilise en fait l'opérateur de fusion introduit avec les graphes de transition. Les tranches créées par cet opérateur sont ensuite écrites une par une dans le port de sortie du PE. Cette zone est créée, en partie, lors de l'initialisation de l'US.

L'US implémente une interface EcritureFlux composée de deux opérations. Elle est décrite sur la Figure 68. Cette interface est requise par le CM et par l'UE. La première opération permet au CM de produire l'ensemble d'échantillons qu'il vient de traiter.

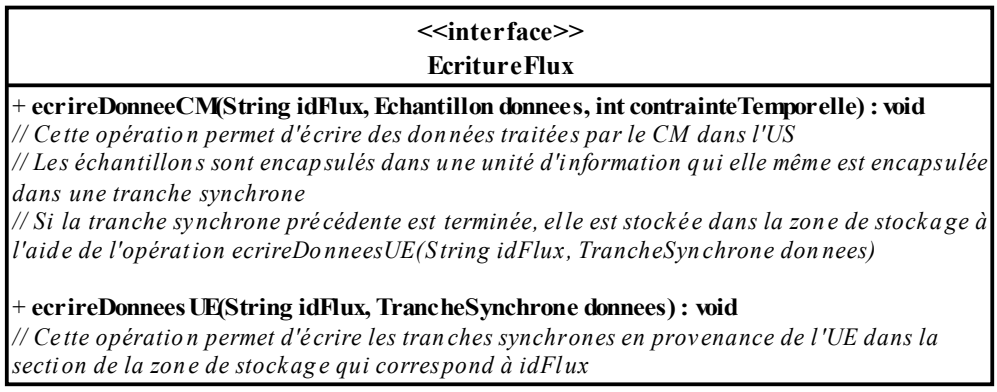


Figure 68 Interface EcritureFlux

Cette opération prépare l'unité d'information qui va encapsuler les échantillons produits par le CM. Pour former un flux primitif, l'unité doit être intégrée dans une tranche synchrone. Le paramètre idFlux désigne le flux de données que l'on va intégrer dans un flux primitif. Le paramètre contrainteTemporelle est un entier qui permet de définir la contrainte du flux (0 pour les flux à contrainte forte et 1 pour les flux à contrainte faible). Les constitutions des unités et des tranches sont réalisées en fonction des comportements décrits précédemment. Lorsque l'écriture concerne un flux à contrainte temporelle faible, l'invocation de l'opération encapsule les données dans une unité qui est intégrée dans une tranche. Cette dernière est directement écrite dans la zone de stockage de l'US. Dans le cas contraire, avant de former une tranche, on va comparer l'étiquette temporelle courante à l'étiquette de la tranche constituée lors de la précédente invocation de l'opération qui est mise en attente car elle est susceptible de contenir plusieurs unités d'information. Si ces étiquettes sont égales, l'unité d'information formée est intégrée à la tranche précédente. Sinon, l'unité est intégrée à une nouvelle tranche qui prend l'étiquette temporelle courante comme attribut. Dans ce cas, la tranche précédente est terminée, elle est écrite dans la zone de stockage de l'US dans la section qui correspond à idFlux. Dans ce sens, la création d'une tranche synchrone d'un flux primitif à contrainte temporelle forte déclenchée par une écriture provenant du CM n'est pas définitive. En effet, il faut toujours attendre l'écriture suivante. La seconde opération permet d'écrire les tranches synchrones provenant de l'UE dans la section de la zone de stockage qui correspond à idFlux. Elle est également utilisée par la première opération afin de stocker une tranche dans cette zone lorsque sa constitution est terminée. La Figure 69 donne les contrats associés à ces opérations. Les préconditions de ces contrats spécifient que le paramètre idFlux existe

afin de permettre le stockage des tranches synchrones dans la zone de l'US. Les post-conditions décrivent les cas d'utilisation décrit précédemment.

Contrat CEF1 : Ecrire des données provenant du CM

Opération : ecrireDonneeCM(String idFlux, Echantillon donnees, int contrainteTemporelle) : void
Préconditions :
- idFlux existe

Postconditions :
- les données sont encapsulées dans une unité d'information qui elle même est intégrée soit dans la tranche constituée lors de la précédente invocation, soit dans une nouvelle tranche
- les tranches ainsi définies sont écrites dans la zone de stockage de l'US dans la section qui correspond à idFlux lorsque leur constitution est terminée (à l'aide de la seconde opération)

Contrat CEF2 : Ecrire des tranches synchrones provenant de l'UE dans la zone de stockage de l'US

Opération : ecrireDonneeUE(String idFlux, TrancheSynchrone donnees) : void
Préconditions :
- idFlux existe

Postconditions :
- les tranches synchrones sont écrites dans la zone de stockage de l'US dans la section qui correspond à idFlux

Figure 69 Contrats de l'Interface EcritureFlux

Enfin, l'US participe également à assurer la synchronisation des entités du modèle c'est-à-dire entre les PE, les conduits et les opérateurs afin que le transfert des flux synchrones entre ces entités s'effectue correctement de bout en bout d'une AMD.

3.1.1.2.2 L'unité de contrôle

L'unité de contrôle (UC) est chargée de fournir les moyens nécessaires à la supervision du PE et donc indirectement du CM par la plate-forme d'exécution. Elle est, à ce titre, capable de contrôler le cycle de vie de ce dernier. La plate-forme va également recevoir de l'UC des informations de QdS traduisant le contexte de fonctionnement du CM. L'UC communique avec la plate-forme d'exécution à l'aide d'événements et d'opérations spécifiques qui traduisent respectivement les états et les commandes qui vont permettre l'initialisation et la manipulation du PE (cf. Figure 61).

Initialisation du Processeur Elémentaire

La construction d'un PE est à l'initiative de la plate-forme d'exécution. Un PE est construit à l'aide d'un CM et à l'aide des conduits qui vont permettre sa connexion au reste de l'AMD. L'éventuelle présence d'un flux prépondérant en entrée du PE est également définie à ce stade. En fonction des ces informations de connexion, l'unité d'échange va être initialisée. Si un PE est construit sans conduits d'entrée, l'UE ne sera pas créée. De la même manière, si le PE est construit sans conduit de sortie, l'US ne sera pas construite. Dans le cas général, un PE possède un ensemble de conduits d'entrée et un seul conduit de sortie. Chacun des conduits d'entrée est destiné à être connecté avec des ports d'entrée. L'UE sera donc construite avec le nombre de ports

nécessaires et l'US par défaut possèdera un port de sortie (cf. 3.1.1.1.1). Les conduits pourront ensuite être connectés en entrée et en sortie du PE.

Le CM est défini à l'aide des noms des flux qu'il doit traiter. Cette information peut être obtenue grâce à l'interface ControleComposant (cf. Figure 57) à l'aide de l'opération obtenirFluxTraites(). Cette interface est fournie par le CM à l'UC. Ainsi, l'UC va donner à l'UE les noms des flux que le CM va traiter. Cette dernière peut commencer à initialiser sa zone de stockage en fonction de ces informations. La fin de ces initialisations est réalisée une fois que le CM est exécuté.

Lorsqu'un CM paramétrable doit être exécuté, il est nécessaire de l'initialiser pour définir le niveau de qualité qu'il doit fournir lors de son exécution. Cette information est rendue possible par la plate-forme à l'aide de l'interface ControlePE fournie par l'UC. Suite à quoi, l'initialisation est réalisée par l'UC à l'aide de l'interface ControleComposant fournie par le CM et de l'opération ajusterQdS(float niveau) (cf. Figure 57).

L'UE est susceptible de fournir des informations sur l'état de fonctionnement du CM qui s'exécute dans un PE. Ces informations sont délivrées à l'UC à l'aide d'événements. Lorsqu'une UE est construite dans un PE, l'UC de ce dernier s'abonne aux événements fournis par l'UE afin de pouvoir être informée de ces informations de QdS. L'UC peut se désabonner de ce service à tout moment. Elle va récupérer ces événements et elle-même générer des événements afin d'informer la plate-forme de ces informations qui traduisent l'état de fonctionnement du CM.

L'UC prépare l'exécution des unités du PE. L'UC démarre les unités et le CM à l'aide de méthodes spécifiques de l'interface controlePE.

Arrêt du Processeur Elémentaire

L'arrêt du processeur élémentaire est également à l'initiative de la plate-forme d'exécution. Cet arrêt est réalisé à l'aide de l'interface ControlePE fournie par l'UC. Le CM est arrêté en premier par l'UC (cf. interface ControleComposant Figure 57). Toutes les tâches en cours sont stoppées ainsi que l'UE et l'US. Les données contenues dans le PE au moment de l'arrêt seront perdues. En effet, lors de l'arrêt d'un PE les traitements effectués par le CM ne sont pas forcément terminés. On considère que les données stockées dans le PE à ce moment là vont de pair avec les données en cours de traitement et donc n'ont de significations que si elles sont couplées ensemble. En conséquence, nous choisissons de ne pas les conserver.

Supervision du Processeur Elémentaire

Les AMD que nous définissons sont supervisables par la plate-forme d'exécution pour pouvoir être reconfigurées dans un but de gestion de la QdS. Cette particularité implique un contrôle total de la plate-forme sur les entités qui composent une AMD. Nous définissons donc des entités entièrement supervisables dont le PE fait partie. Pour assurer cette supervision, l'UC fournit une interface appelée ControlePE. Elle est décrite sur la Figure 70 et permet de contrôler le cycle de vie du PE et de ses différents modules.

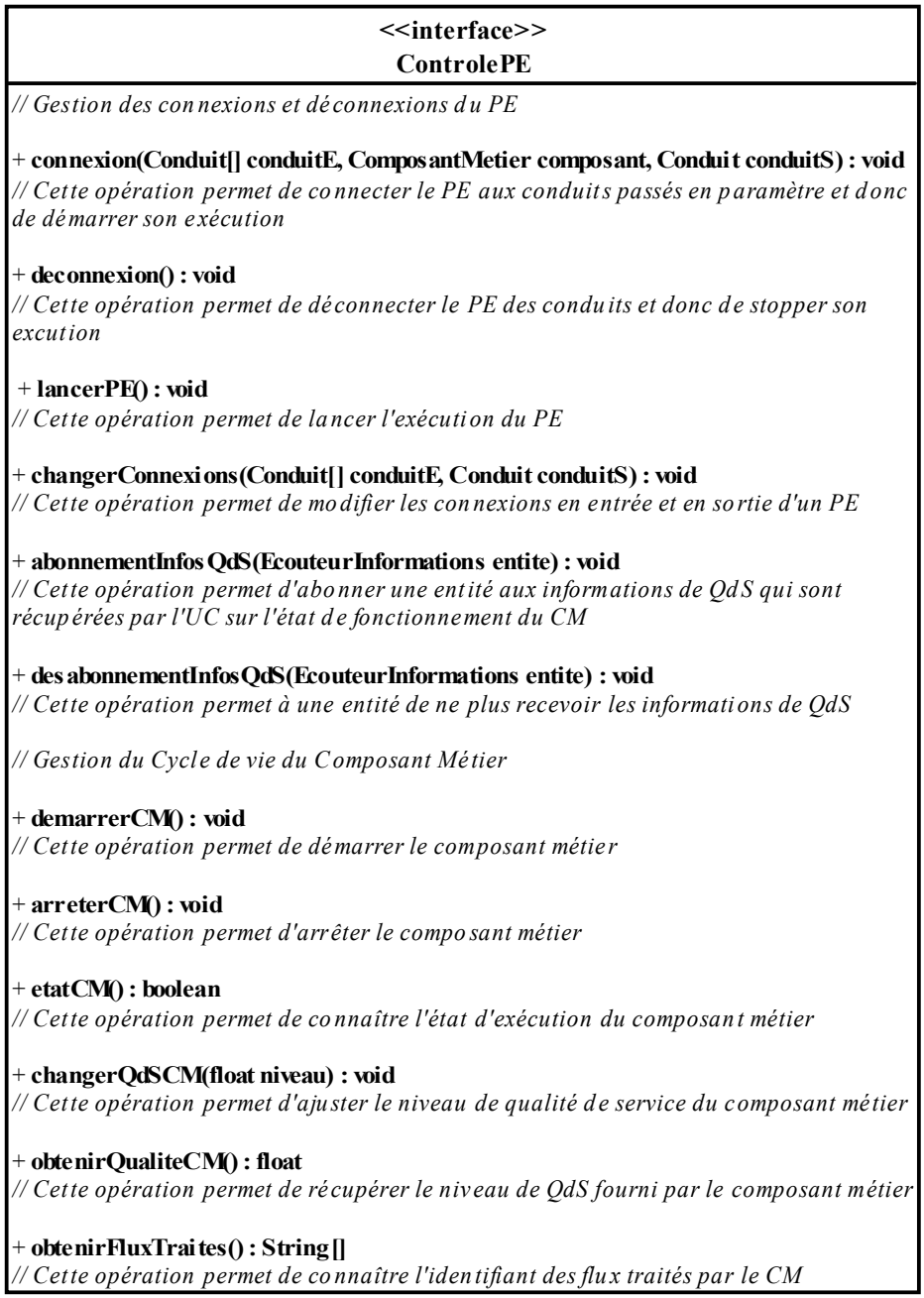


Figure 70 Interface ControlePE

Cette interface est composée de deux parties. La première concerne les aspects relatifs à la connexion du PE tandis que la seconde est liée à la gestion du cycle de vie du CM. Nous ne détaillons pas à nouveau cette seconde partie car elle se base sur l'interface ControleComposant que nous avons définie (cf. Figure 57). Les opérations de cette interface possèdent donc les mêmes contrats que celles de l'interface ControleComposant. Elles permettent à la plate-forme de piloter le CM par l'intermédiaire de l'UC.

La Figure 71 donne les contrats associés à la connexion du PE.

Contrat CCP1 : Connexion du PE et des conduits d'entrée/sortie

Opération : connexion(Conduit[] conduitE, ComposantMetier composant, Conduit[] conduitS) : void
Préconditions :
- les conduits doivent être dédarés au préalable
- le CM passé en paramètre doit être dédaré
- les conduits et le CM ne doivent pas se trouver en état de fonctionnement

Postconditions :
- le PE est entièrement défini en fonction des paramètres passés à cette opération
- il est prêt à fonctionner

Contrat CCP2 : Déconnexion du PE et des conduits d'entrée/sortie

Opération : deconnexion() : void
Préconditions :
- le PE est en train de fonctionner

Postconditions :
- les unités du PE sont stoppées
- le CM est arrêté
- les conduits d'entrée et de sortie sont déconnectés
- les entités abonnées au service d'événements sont automatiquement retirées de la liste et désabonnées de droit, un événement leur est envoyé afin de les informer

Contrat CCP3 : Lancer l'exécution du PE

Opération : lancerPE() : void
Préconditions :
- le PE est à l'arrêt

Postconditions :
- le PE est exécuté

Contrat CCP4 : Changer les connexions du PE

Opération : changerConnexions(Conduit[] conduitE, Conduit conduitS) : void
Préconditions :
- les conduits passés en paramètre doivent exister
- la modification des connexions ne peut en aucun cas changer le nombre de conduits en entrée d'un PE

Postconditions :
- les connexions sont mises à jour en fonction des paramètres de l'opération

Contrat CCP5 : Abonnement aux informations de QdS

Opération : abonnementInfosQdS(EcouteurInformations entite) : void
Préconditions :
- l'entité ne doit pas déjà être abonnée à ce service

Postconditions :
- l'entité est ajoutée à la liste des abonnés, elle recevra désormais les événements liés à ces informations

Contrat CCP6 : Désabonnement aux informations de QdS

Opération : desabonnementInfosQdS(EcouteurInformations entite) : void
Préconditions :
- l'entité doit être abonnée à ce service

Postconditions :
- l'entité est retirée de la liste des abonnés, elle ne recevra plus les événements liés à ces informations

Figure 71 Contrats de l'Interface ControlePE

L'opération de connexion permet de définir le PE à l'aide d'un CM et de conduits d'entrée/sortie. Une fois le PE déclaré, il doit être mis en état de marche à l'aide de l'opération lancerPE(). L'opération de déconnexion permet de stopper le fonctionnement du PE et de supprimer ces connexions. L'invocation de cette opération provoque l'arrêt du CM. Une opération changerConnexion(Conduit[] conduitE, Conduit conduitS) est définie afin de pouvoir changer dynamiquement les connexions que pos-

sède un PE. Cette opération ne change en aucun cas le nombre de conduits connectés à un PE. Elle ne permet donc pas l'ajout ou le retrait de conduit sur un PE. Elle permet seulement de remplacer un ou plusieurs conduits connectés au PE. Enfin, un système d'abonnement est proposé afin que la plate-forme puisse à loisir s'abonner ou se désabonner de la réception des informations de QdS qu'un PE est susceptible de fournir.

La plate-forme est susceptible de reconfigurer les AMD afin d'adapter leur fonctionnement à la QdS requise par les utilisateurs. Pour ce faire, elle peut effectuer des opérations de reconfiguration. Elle est capable d'ajouter, de retirer, de remplacer ou de déplacer un CM d'un site vers un autre et enfin modifier le niveau de qualité fourni par les CM paramétrables. La Table 8 donne le détail des opérations de reconfiguration de la plate-forme. Cette dernière utilise l'interface ControlePE afin de mener à bien ces opérations. Nous les avons nommé ajout, retrait, remplacement, déplacement et modification dans cette table. L'opération de remplacement d'un CM par un autre est en fait l'exécution d'une opération de retrait suivie d'une opération d'ajout. L'opération de déplacement d'un CM d'un site vers un autre consiste à effectuer une opération de remplacement distribuée c'est-à-dire que l'on va retirer un CM d'un site et l'ajouter sur un autre. Dans ce cas les conduits d'entrée, qui étaient connectés au PE qui contenaient le CM que l'on veut déplacer, doivent être changés en conduits distribués. Le conduit de sortie, qui était alors un conduit distribué, doit être changé pour un conduit local. Il faut noter que ces opérations sont réalisées par les parties locales à chaque site de la plate-forme. La Figure 72 donne le principe de cette opération.

Lorsque la plate-forme décide de changer un CM pour, par exemple, le remplacer par un autre, le PE qui contient le CM à remplacer (celui en cours de fonctionnement) doit être déconnecté de l'application (opération de deconnexion() du PE). Ainsi, le CM à ajouter est encapsulé dans un nouveau PE que la plate-forme se charge de construire.

Table 8 Commandes de Reconfiguration de la Plate-Forme

Commandes de la Plate-Forme d'exécution	*Opérations invoquées*
AJOUT	connexion(Conduit[] conduitE, ComposantMetier composant, Conduit conduitS) demarrerCM() lancerPE() abonnementInfosQdS(EcouteurInformations entite)
RETRAIT	desabonnementInfosQdS(EcouteurInformations entite) deconnexion()
REMPLACEMENT (RETRAIT + AJOUT)	desabonnementInfosQdS(EcouteurInformations entite) deconnexion() connexion(Conduit[] conduitE, ComposantMetier composant, Conduit conduitS) demarrerCM() lancerPE() abonnementInfosQdS(EcouteurInformations entite)
DEPLACEMENT (RETRAIT + AJOUT distribué)	***Sur un site*** desabonnementInfosQdS(EcouteurInformations entite) deconnexion() remplacement des conduits d'entrée par des conduits distribués ***Sur l'autre site*** remplacement du conduit de sortie distribué par un conduit local connexion(Conduit[] conduitE, ComposantMetier composant, Conduit conduitS) demarrerCM() lancerPE() abonnementInfosQdS(EcouteurInformations entite)
MODIFICATION	obtenirQdSCM() changerQdSCM(float niveau)

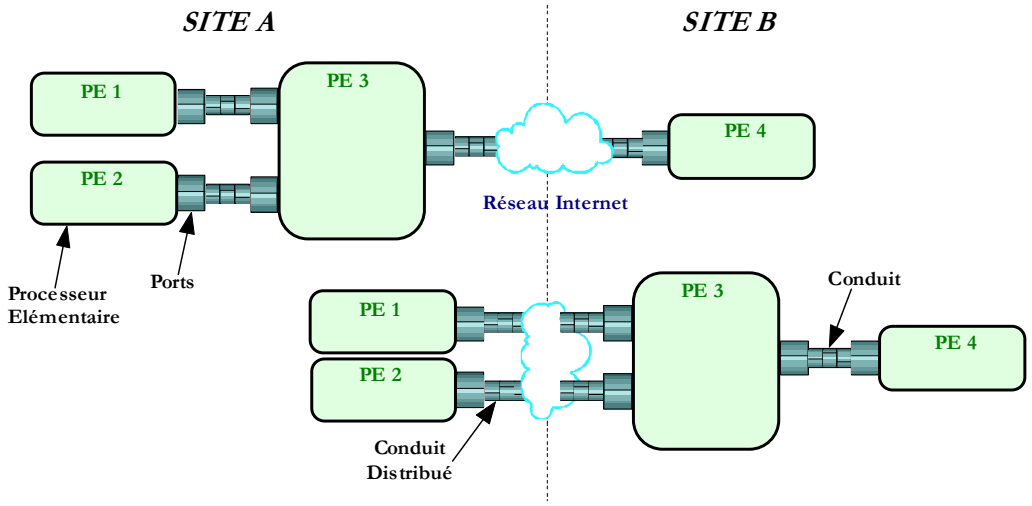


Figure 72 Déplacement du PE 3 du site A vers le site B

Gestion des événements

Le PE est capable de générer des événements à destination de la plate-forme d'exécution afin de l'informer de l'état d'exécution du CM qu'il exécute. Ces événements traduisent des informations de QdS et sont récupérés par le gestionnaire d'événements de la plate-forme (cf. chapitre 3). Le gestionnaire prend la décision de traiter ou pas les événements ainsi reçus.

Ces événements apparaissent lorsqu'une anomalie de QdS se produit ou va se produire. Dans le cas du PE, ils sont produits par l'UE et sont récupérés par l'UC. Ces événements sont émis avec une périodicité T et ceci jusqu'à ce que leur cause disparaisse [LAP06]. Il est donc inutile de les acquitter puisque leur cause disparaît dés qu'ils sont traités avec succès.

Ces événements sont classés en deux types. Les événements notés E_1 indiquent que le CM en cours d'exécution ne remplit plus correctement son rôle. Ce type d'événement avertit la plate-forme que la QdS est en train de se dégrader. Ces événements indiquent donc qu'il faut reconfigurer car la configuration actuelle n'est plus adaptée au contexte. Les événements notés E_2 indiquent que le CM en cours d'exécution est sous-utilisé. Ils préviennent donc la plate-forme que la QdS peut être améliorée. Ils indiquent donc qu'il est possible de reconfigurer car la configuration actuelle n'est pas optimale pour le contexte actuel.

Une partie de ce chapitre présente les informations que l'UE va obtenir sur le fonctionnement du PE.

3.1.1.2.3 Communications entre les modules du Processeur Elémentaire

Enfin, pour finir cette présentation du processeur élémentaire, nous résumons son architecture à l'aide d'un diagramme basé sur les diagrammes de composants UML [OMG03]. Ce diagramme est représenté sur la Figure 73. Il décrit les principales articulations des modules du PE en précisant les modes d'interactions choisis et présentés tout au long de cette section sur les composants fonctionnels. Le PE est représenté à l'aide d'un paquetage (habituellement utilisé pour modéliser les systèmes ou les sous-systèmes). Nous avons représenté chaque module du PE à l'aide de composants qui requièrent et fournissent des interfaces pour assurer leur communication. Nous retrouvons donc les interfaces que nous avons présentées jusqu'à maintenant. La seule implémentation qui change dans le PE est celle du CM. Les communications avec la plate-forme sont réalisées à l'aide d'interfaces et d'événements. L'interface ControlePE permet d'exécuter des commandes sur le PE. Les événements NotificationQds permettent de donner les états de fonctionnement du PE à cette dernière. Les communications entre le PE et les conduits sont réalisées à l'aide de l'interface AccesPort.

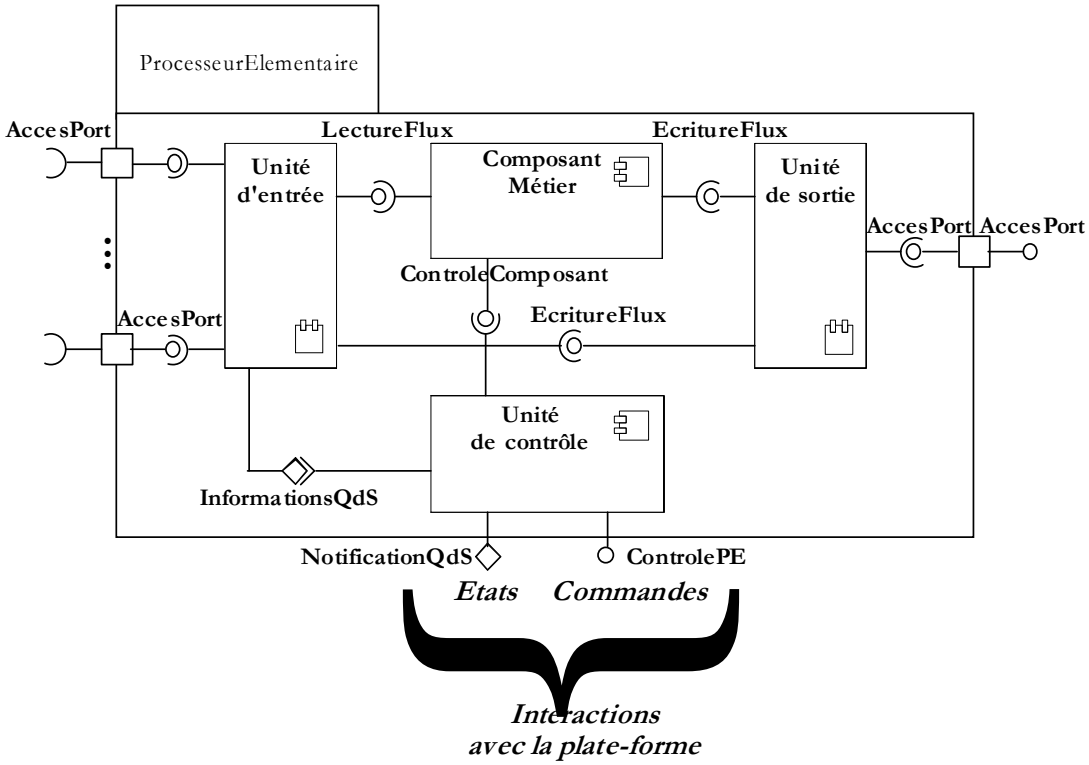


Figure 73 Communication entre les modules du PE

3.1.2 Les Opérateurs de Flux Synchrones

La deuxième classe d'entités que le modèle définit concerne des opérateurs de flux synchrones. Ces opérateurs relèvent des aspects non-fonctionnels des AMD. L'objectif est d'assurer et de permettre les spécifications fonctionnelles proposées par la méthode de conception. Leur nécessité est apparue lors du passage des graphes fonctionnels aux graphes de transition. On en distingue quatre qui sont : la fusion, la séparation, la disjonction et la conjonction. Nous avons vu que les opérateurs de fusion et de séparation sont utilisés dans le processeur élémentaire. L'opérateur de fusion permet de créer un flux composé à partir d'un ensemble de flux synchrones provenant tous du même site. L'opérateur de séparation réalise la fonction inverse en séparant un flux composé qu'il reçoit en entrée en un ensemble de flux primitifs. L'opérateur de disjonction permet l'implémentation des disjonctions assurant la définition de chemins parallèles sur les graphes de transition. Il permet de récupérer, à partir d'un flux composé, les flux primitifs qui doivent être traités en parallèle dans des chemins de disjonction. Enfin, l'opérateur de conjonction permet de réaliser la fonction inverse lorsque les traitements parallèles des flux primitifs sont terminés. Il récupère ces flux primitifs et forment le flux composé tel qu'il était initialement (avant la disjonction). En effet, l'opérateur de conjonction est forcément couplé à un opérateur de disjonction en amont (cf. § 2.4).

3.1.2.1 Implémentation des Opérateurs

Les politiques de synchronisation du modèle Korrontea définissent une opération qui permet de produire des flux composés à partir d'un ensemble de flux synchrones (cf. chapitre 5). L'opération inverse n'est pas explicitement définie mais semble tout à fait possible et facile à implémenter. Elle permettrait alors de séparer un flux composé en un ensemble de flux primitifs. Ces deux opérations vont nous permettre d'implémenter les opérateurs de fusion[68] et de séparation.

La fonction de l'opérateur de disjonction est proche de celle de l'opérateur de séparation dans le sens où on doit, à partir d'un flux composé, fournir un ensemble de flux primitifs afin que chacun puisse être traité par des composants de traitement différents appliqués en parallèle. L'opérateur de disjonction peut donc être implémenté grâce à l'opération de séparation des flux composés. Lorsque les traitements parallèles de plusieurs flux sont terminés, il est nécessaire de reconstituer le flux composé tel qu'il existait avant la disjonction. La fonction de l'opérateur de conjonction est de fournir ce flux composé. De la même manière, cette fonction peut être réalisée grâce aux politiques de synchronisation dont l'objectif est de fabriquer des flux composés.

On dispose donc de deux opérations, l'une qui permet de créer des flux composés et l'autre qui permet de diviser ces flux composés en flux primitifs. A partir de ces deux opérations, on va pouvoir définir les quatre opérateurs. Nous résumons ceci à l'aide de la Table 9.

---

68 De plus, lorsque l'opérateur de fusion a été défini, il a clairement été expliqué que ce dernier serait réalisé à l'aide des politiques de synchronisation du modèle Korrontea.

Table 9 Opérateurs de Flux Synchrones

Opérations / Opérateurs	*Politiques de Synchronisation*	*Division d'un flux composé en flux primitifs*
Fusion		
Séparation		
Disjonction		
Conjonction		

Dans la suite, nous présentons uniquement les fonctionnements de l'opérateur de fusion et de séparation puisque les deux autres sont basés sur les mêmes principes.

3.1.2.1.1 L'Opérateur de Fusion

L'opérateur de fusion utilise les politiques de synchronisation. Il accepte en entrée un ensemble de flux primitifs et il produit en sortie le flux correspondant. La première tâche est d'évaluer cet ensemble de flux en étudiant les contraintes temporelles de chacun. Ainsi, on va pouvoir déterminer la politique à utiliser. L'algorithme de cet opérateur est donné sur la Figure 74. Nous ne détaillons pas les algorithmes de chaque politique, ils sont donnés dans le chapitre 5.

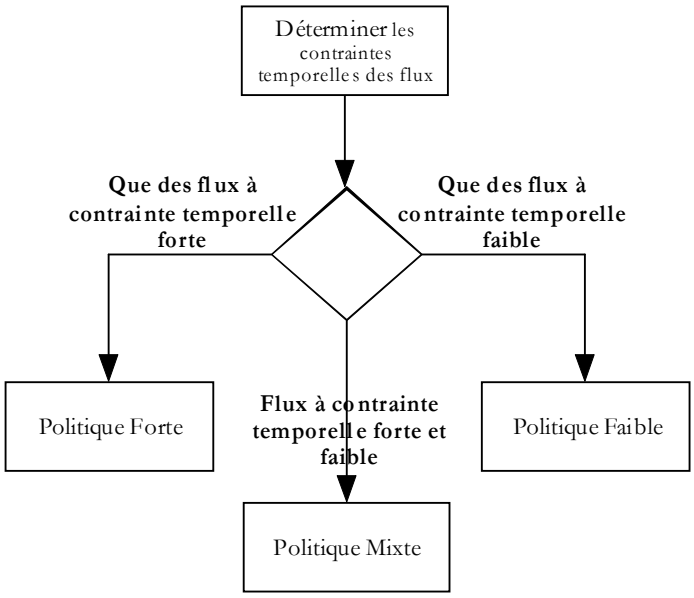


Figure 74 Algorithme de l'opérateur de Fusion

3.1.2.1.2 L'Opérateur de Séparation

L'opérateur de séparation permet de produire, à partir d'un flux composé, l'ensemble des flux de données contenus dans ce dernier sous la forme de flux primitifs. Pour cela, il détermine le nombre de flux de données contenus dans une tranche synchrone. Cette information est facile à obtenir dés lors que l'on dispose de la première tranche du flux composé : $\forall$ TS $\in$ F_n où F_n est le flux composé qui doit être séparé, le nombre de flux est égal à |flux(TS)|. Ensuite, pour chaque tranche qui se présente en entrée de l'opérateur, on regarde dans le flux n (n varie de 1 à nombre de flux de données) si des unités d'information sont présentes. Si c'est le cas, on crée une tranche à laquelle on associe l'étiquette temporelle de la tranche en cours de séparation et les unités d'information du flux de données n. Cette tranche est envoyée en sortie de l'opérateur afin de la rendre disponible à l'extérieur. Si ce n'est pas le cas, aucune tranche n'est créée pour ce flux de données et on passe au suivant. On traite de cette manière toutes les tranches du flux composé qui se présentent en entrée de l'opérateur. L'algorithme de cet opérateur est donné sur la Figure 75. Dans cet algorithme la contrainte temporelle du flux composé importe peu puisque l'on traite les tranches seulement lorsque celles-ci sont disponibles en entrée.

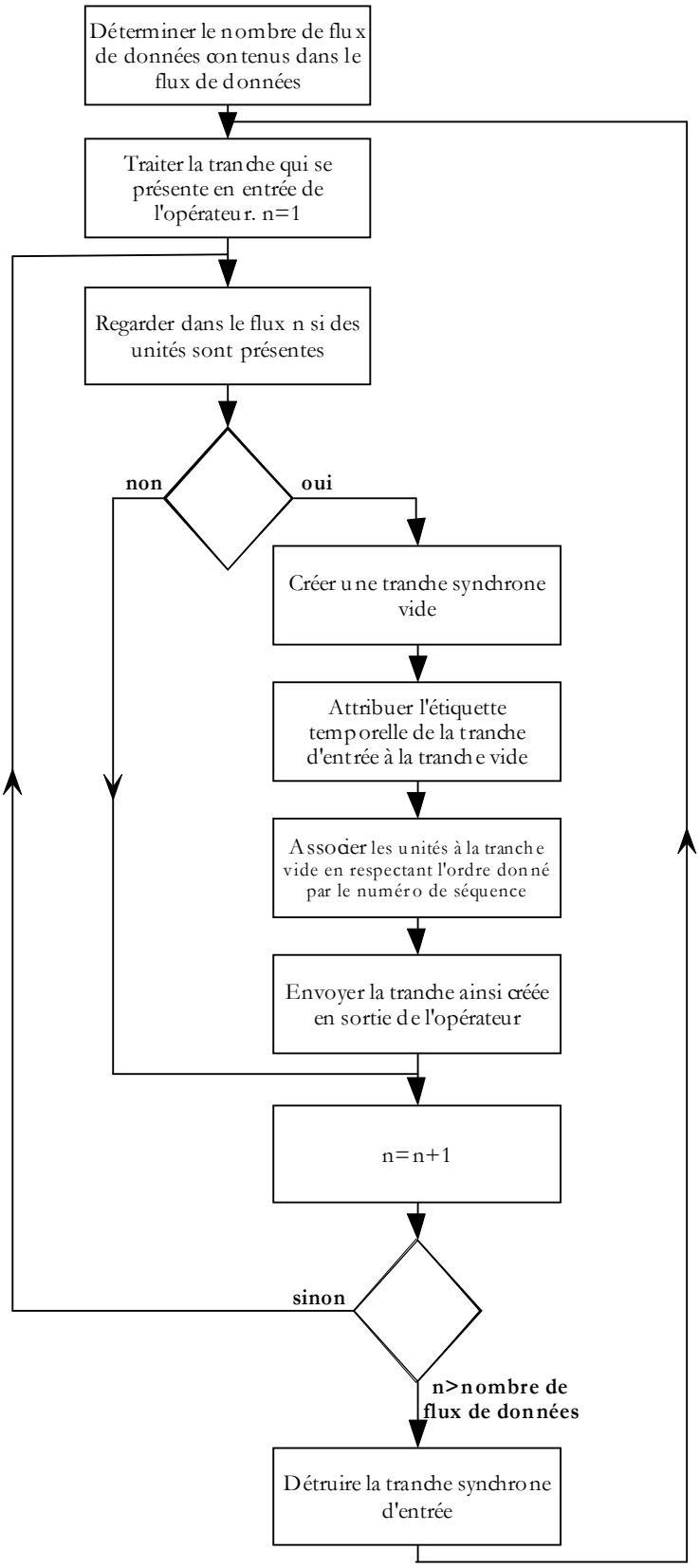


Figure 75 Algorithme de l'opérateur de Séparation

3.1.2.1.3 Réalisation de Structures Complexes par Assemblage d'Opérateurs

Lorsque nous avons présenté les opérateurs avec les règles de transformation, nous les avons introduit avec les cas d'utilisation les plus courants et les plus simples à mettre en œuvre. Néanmoins, on peut essayer de résoudre des structures de graphes fonctionnels plus complexes à l'aide de ces quatre opérateurs. L'objectif de cette section est de montrer qu'en réalisant des assemblages d'opérateurs, on arrive à résoudre de telles structures. Nous prenons un exemple de graphes fonctionnels afin d'appuyer notre argumentation. Considérons donc une partie d'un graphe fonctionnel représenté sur la gauche de la Figure 76. Ce graphe possède trois flux en entrée qui doivent rester synchrones pendant leur transfert dans l'AMD. Les deux premiers flux sont de la vidéo et de l'audio, le dernier est un flux de sous-titres. Les deux premiers flux doivent subir un prétraitement audio/vidéo synchrone c'est-à-dire que ce dernier ne peut être appliqué que si les deux flux sont synchrones. Ensuite, chacun des deux flux subit son traitement respectif et leur transport synchrone (tous les deux) n'est plus nécessaire. Néanmoins, ils doivent rester liés au flux de sous-titres qui subit également un traitement. Le graphe de transition que nous proposons pour réaliser ce graphe fonctionnel est donné sur la partie droite de la Figure 76. Les trois flux sont donc synchronisés par un opérateur de fusion M_1. Ces flux doivent être traités en parallèle. D'un côté, on doit effectuer un prétraitement synchrone sur un flux composé de l'audio et de la vidéo et de l'autre un flux primitif de sous-titres. Pour ce faire, on utilise l'opérateur de disjonction D_1 qui fournit en sortie f_1, f_2 et f_3. f_1 et f_2 sont alors rassemblés par M_2 dans un flux composé F_{11}. Ainsi, l'opérateur de disjonction couplé à un opérateur de fusion permet de proposer ce genre de possibilités. Après cet assemblage, on aura bien un flux composé F_{11} contenant l'audio et la vidéo et un flux primitif f_3 contenant les sous-titres. Sur le reste du graphe, on pourrait utiliser un opérateur de conjonction entre f_1 et f_2 mais il n'est pas nécessaire dans ce cas là puisque Co_1 va se charger de recréer le flux composé F_1 à partir de f_1, f_2 et f_3. Ainsi, le couplage de ces opérateurs nous permet d'implémenter les propriétés fonctionnelles spécifiées.

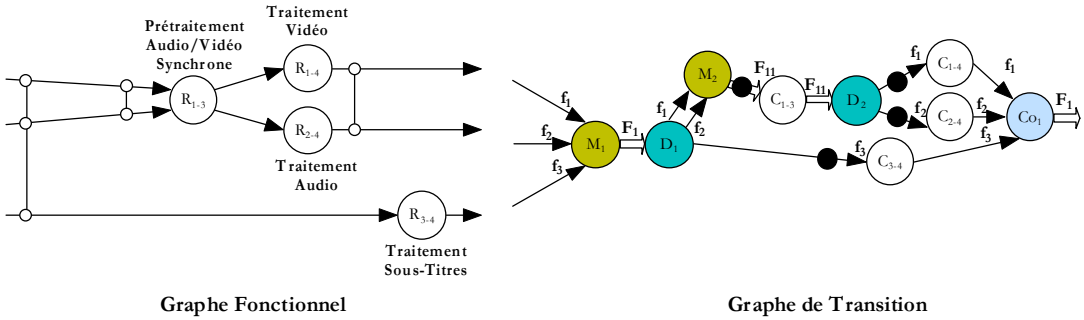


Figure 76 Exemple de Spécifications Fonctionnelles Complexes

3.1.2.2 Architecture générale des Opérateurs

Les opérateurs adressent les propriétés non-fonctionnelles des AMD. Leur structure est donc différente de celle des PE puisqu'ils justifient cette dernière pour assurer l'implémentation de la partie métier. Ils forment une unité unique qui implémente une des deux opérations non-fonctionnelles identifiées précédemment. Les opérateurs sont dotés de ports d'entrée et de sortie afin qu'ils puissent être connectés aux entités fonctionnelles. Comme pour ces dernières, ils récupèrent les tranches issues des flux synchrones par l'intermédiaire des ports d'entrée et fournissent les tranches créées sur les ports de sortie. Les ports possèdent le même fonctionnement que ceux utilisés dans les entités fonctionnelles (cf. § 3.1.1.2.1). Ils fournissent donc également une interface AccesPort (cf. Figure 62 et Figure 63).

Les opérateurs sont également supervisables par la plate-forme d'exécution et fournissent pour ce faire une interface unique qui s'appelle ControleOperateur. Cette interface est décrite sur la Figure 77. Elle contient trois opérations liées à leur connexion et trois opérations liées au contrôle de leur cycle de vie. Les contrats de cette interface sont donnés sur la Figure 78. Un point important de ces contrats est la spécification des modalités de connexion suivant l'opérateur que l'on utilise. En effet, tous les opérateurs ne sont pas connectés de la même manière. Ainsi, les opérateurs de fusion et de conjonction possèdent n flux d'entrée pour un conduit de sortie. Les opérateurs de séparation et de disjonction acceptent un flux en entrée et produisent n flux en sortie.

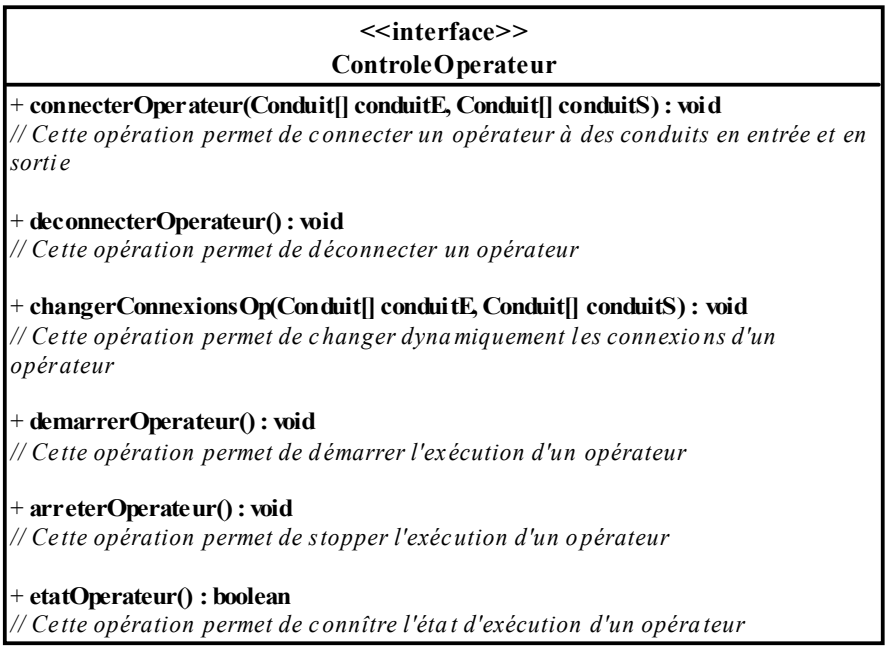


Figure 77 Interface ControleOperateur

Contrat CCO1 : Connexion d'un opérateur avec des conduits d'entrée et de sortie
Opération : connecterOperateur(Conduit[] conduitE, Conduit[] conduitS) : void
Préconditions :
- les conduits doivent être dédarés au préalable
- suivant les opérateurs utilisés, les entrées/sorties sont limitées :
  * les opérateurs de fusion et de conjonction possèdent n conduits en entrée et un seul en sortie
  * les opérateurs de séparation et de disjonction possèdent 1 conduit en entrée et n en sortie

Postconditions :
- l'opérateur est connecté aux conduits spécifiés en paramètres, il est prêt à être démarré

Contrat CCO2 : Déconnexion d'un opérateur

Opération : deconnecterOperateur() : void
Préconditions :
- l'opérateur doit être arrêté au préalable

Postconditions :
- les conduits d'entrée et de sortie sont automatiquement déconnectés

Contrat CCO3 : Changer les connexions d'un opérateur

Opération : changerConnexionsOp(Conduit[] conduitE, Conduit[] conduitS) : void
Préconditions :
- les conduits passés en paramètre doivent être dédarés au préalable
- la modification des connexions ne peut en aucun cas changer le nombre de conduits d'entrée et de sortie de l'opérateur

Postconditions :
- les connexions sont réinitialisées en fonction des conduits passés en paramètre

Contrat CCO4 : Démarrer l'opérateur

Opération : demarrerOperateur() : void
Préconditions :
- l'opérateur doit être à l'arrêt

Postconditions :
- l'opérateur est mis en marche

Contrat CCO5 : Stopper l'opérateur

Opération : arreterOperateur() : void
Préconditions :
- l'opérateur doit être en état de marche

Postconditions :
- l'opérateur est arrêté

Figure 78 Contrats de l'Interface ControleOperateur

Ces opérateurs sont constitués de buffers en entrée et de buffers en sortie. Le nombre de buffers varie en fonction du nombre de conduits connectés en entrée et en sortie de l'opérateur tout en respectant les modalités de connexion données par le contrat CCO1 (cf. Figure 78). En d'autres termes, on retrouve un buffer d'entrée par flux connecté en entrée et un buffer de sortie par flux connecté en sortie. L'opération implémentée par un opérateur (cf. Figure 74 et Figure 75) va chercher des données à traiter sous la forme de tranches synchrones dans les buffers d'entrée, ces données sont traitées et produites dans les buffers de sortie et ainsi de suite. Les ports d'entrée et les ports de sortie s'occupent respectivement d'alimenter et de vider ces buffers.

A la différence des entités fonctionnelles, les opérateurs ne fournissent pas d'informations de QdS à la plate-forme d'exécution. Les fonctionnalités fournies sont des mécanismes qui vont permettre d'assurer une qualité de service comme la synchronisation des flux ou la possibilité d'effectuer plusieurs traitements en parallèle. De plus, ils sont destinés à être utilisés dans des endroits précis de l'AMD conformément aux spécifications fonctionnelles. Les phases de reconfiguration ne peuvent pas cibler ces opérateurs car le fait de les supprimer ou de les déplacer pourrait influer de façon négative sur la QdS perçue par les utilisateurs des AMD. En conséquence, ces unités ne participent pas directement à la gestion de la QdS par adapatation de l'architecture logicielle des AMD.

3.2 Les Connecteurs de Composants Logiciels

La partie précédente a permis d'introduire les composants logiciels qui sont les entités chargées d'implémenter les préoccupations fonctionnelles et les préoccupations non-fonctionnelles des AMD que nous avons identifiées. Il est maintenant nécessaire de définir un moyen pour permettre la connexion de ces composants afin de réaliser des assemblages. Ce moyen fait l'objet de la seconde entité fonctionnelle dont nous avons justifié la nécessité à l'issue de la présentation de la méthode de conception. Cette entité est dédiée à la connexion des composants et plus précisément au transport des flux synchrones entre eux. On a également vu que l'on utilisait des concepts similaires aux architectures « pipes & filters » [GAN94], [GAR94] que l'on va retrouver au niveau de la définition de cette entité que nous avons baptisée le conduit.

L'objectif de cette section est de définir le conduit qui se charge du transport des flux synchrones dans les AMD. Ce transport doit être synchrone afin de répondre aux spécifications fonctionnelles. Il doit pouvoir être réalisé de façon locale ou distribuée entre les composants. Nous avons vu dans le chapitre 1 que le transport distribué constitue une seconde source de désynchronisation des données. Par conséquent, l'entité définie doit permettre d'éviter cet inconvénient.

Le conduit participe à la fourniture du service requis. Cependant, son fonctionnement est aussi dépendant du contexte d'exécution surtout dans le cas des conduits distribués. Cette entité doit donc être supervisable par la plate-forme d'exécution afin que cette dernière puisse gérer les liens entre les composants. De plus, les conduits doivent pouvoir fournir des informations de QdS qui vont permettre de déceler des anomalies de QdS mais aussi des possibilités d'amélioration de la QdS.

Dans ce type d'architecture, les traitements sont déclenchés par les données qui transitent de façon continue. Cette caractéristique est due à la structure basée flux de

données que nous avons choisie. Elle permet de fait un séquencement implicite des entités qui composent une configuration. Néanmoins, afin d'être efficace une telle architecture doit répondre à un certain nombre de règles. Nous nous proposons dans un premier temps d'étudier ces règles à travers la coordination entre les entités du modèle. Cette étude est bien évidemment centrée sur le conduit.

3.2.1 Etude de la coordination Composant-Conduit-Composant

Une AMD est composée d'un ensemble de composants logiciels (CM/PE et opérateurs de flux) interconnectés par des conduits chargés du transport synchrone des flux. Cette entité est un point central de l'architecture qui doit assurer la coordination des composants logiciels c'est-à-dire leur permettre de fonctionner efficacement ensemble.

L'entité responsable du transport des données doit assurer la récupération des données traitées par un composant afin de les transférer vers le traitement suivant et ainsi de suite. Cette séquence doit être respectée et conforme aux spécifications fonctionnelles. Par exemple, si on utilise un composant de compression de données avant de les transmettre sur le réseau, il faut s'assurer que ce traitement est correctement établi afin que le composant de décompression situé à l'autre bout du réseau puisse correctement décompresser les données. Ce type de fonctionnement se base sur le modèle producteur/consommateur. Afin de bien comprendre, on peut utiliser l'exemple du pipe utilisé par le système d'exploitation UNIX. Dans ce système, le pipe permet de rediriger la sortie standard d'une commande vers l'entrée standard d'une autre permettant ainsi de répondre à des besoins spécifiques. Ainsi, la seconde commande ne peut débuter son exécution qu'une fois que la première a terminé la sienne. Si nous transposons cet exemple à nos préoccupations, on se rend compte que les mécanismes sont similaires et qu'une donnée traitée par un CM ne peut être transportée par le conduit que lorsque l'on est sûr que le composant a terminé son traitement.

Plusieurs auteurs ont étudié et proposé des modèles de coordination. Malone et Crowston [MAL94] définissent un ensemble de contraintes qu'ils jugent nécessaires afin de définir une coordination basée sur le modèle producteur/consommateur efficace. Ils proposent d'utiliser le terme de dépendances entre un producteur et un consommateur. Le schéma que nous utilisons pour faire le lien avec nos travaux est celui d'un composant producteur de données, transportées par un conduit vers un composant consommateur. Ils spécifient alors trois types de contraintes afin de réaliser ce type de coopération.

La première est appelée contrainte préalable. Elle spécifie le fait que le composant consommateur de données ne peut appliquer son traitement que si le composant

producteur a terminé le sien. Cette contrainte signifie que le composant consommateur par l'intermédiaire du conduit doit être sûr que les données qu'il va consommer ont été traitées correctement en amont. Afin de considérer cette contrainte dans notre modèle, nous proposons que les données soient écrites en sortie une fois que le traitement du composant est terminé. Ainsi, un système de notification par événements entre les composants et les conduits permet de signifier aux conduits et aux composants que les données peuvent être transportées ou consommées.

La contrainte de transfert est la seconde introduite. Nous avons anticipé son existence depuis le début de cette section mais il est quand même nécessaire de la présenter. Pour que des données produites puissent être consommées par un autre composant, il est nécessaire de les transférer vers ce consommateur. Nous justifions donc l'utilisation d'une entité dédiée à cette tâche. Le pipe UNIX abordé tout à l'heure répond à cette contrainte dés lors qu'il permet de fournir la sortie d'une commande en entrée d'une autre. En ce qui concerne nos travaux, nous ajoutons même la notion de transfert synchrone importante dans les applications qui manipulent des médias.

Enfin, la dernière contrainte est dite d'utilisabilité. Elle permet de spécifier une autre caractéristique importante qui réside dans le fait que les données produites par un composant producteur doivent pouvoir être utilisables par le composant consommateur afin que la coordination mise en place ait un sens. Cette contrainte est beaucoup liée à la sémantique de la coordination et à des notions de compatibilité. Dans nos travaux, cette sémantique est apportée en amont de la définition du conduit puisqu'elle doit être assurée par les différentes étapes de la méthode de conception. De plus avec la définition du modèle Korrontea, nous faisons un pas de plus vers cette contrainte dés lors que nous rendons unique la manipulation des données. L'exemple du pipe UNIX permet également de justifier cette contrainte.

Nous présentons donc par la suite, le conduit que nous avons spécifié afin qu'il permette ce genre de coordination entre les composants logiciels d'une AMD.

3.2.2 Architecture Interne du Conduit

L'architecture interne du conduit est similaire à celle du PE en termes d'unités qui le composent. Cependant, ces unités possèdent des comportements différents de celles définies dans le PE. Avant de débuter le fonctionnement précis du conduit, nous tenons à préciser que le conduit sert à transporter un unique flux synchrone qu'il soit primitif ou composé.

Le conduit se dote donc de deux unités qui sont l'unité d'échange et l'unité de contrôle. Comme pour le PE, l'unité d'échange est dédiée à la gestion des connexions

en entrée et en sortie du conduit. Elle se compose de deux unités chacune chargée de gérer les connexions de chaque extrémité. Ainsi, l'UE est chargée des connexions en entrée de celui-ci et l'US des connexions de sortie. L'UC quant à elle va permettre la supervision de cette entité par la plate-forme d'exécution. La justification de ces entités peut être amenée par une table similaire à la Table 7. Nous ne la redéfinissons pas car on va se rendre compte que l'on se trouve confronté au même type de spécifications. Cependant, les finalités de ces deux entités sont différentes, les unités introduites ne possèderont pas les mêmes comportements. L'architecture du conduit est donnée sur la Figure 79. Le processus client/serveur est chargé de récupérer les données provenant de l'UE et de les transférer vers l'US. Le conduit représenté sur cette figure est un conduit local. Un conduit distribué se structure de la même manière à la différence que le réseau sépare l'UE et l'US. Ainsi, le processus client/serveur et l'UC se retrouvent donc distribués sur deux machines. L'UC peut donc être supervisée par la plate-forme sur ces deux endroits. De plus, les états qu'elle est sensée envoyer à la plate-forme peuvent également l'être des deux côtés du conduit distribué.

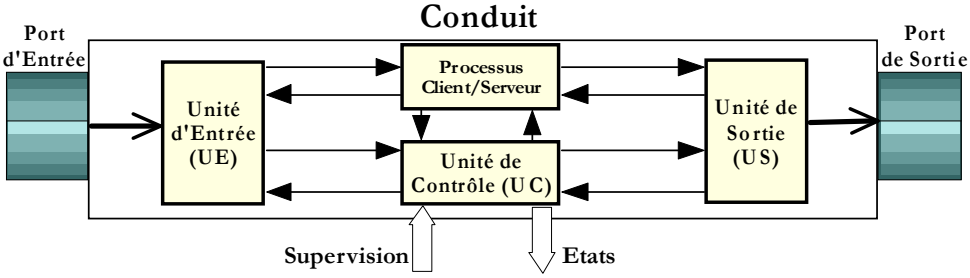


Figure 79 Architecture interne du Conduit

3.2.2.1 L'unité d'échange

L'unité d'échange se scinde en deux parties distinctes, à savoir l'UE et l'US. Ces unités se justifient par le fait qu'elles sont chargées des connexions respectives en entrée et en sortie du conduit mais aussi par le fait que le conduit peut être dans certains cas distribué et donc qu'il aspire à être réparti sur deux machines différentes, une qui contiendra l'UE et l'autre l'US. Chacune de ces unités se dote d'un port d'entrée et d'un port de sortie afin de rendre compatible les connexions de ce dernier et des composants logiciels. On se réfèrera au § 3.1.1.2.1 pour les caractéristiques des ports d'entrée/sortie.

L'UE et l'US du conduit ont des comportements très simples et très minimalistes. Le but du conduit est de transférer les données du composant producteur vers le consommateur. Ces données sont transférées sous la forme de flux synchrones composés ou primitifs. Dés qu'une tranche synchrone est disponible dans le port d'entrée

du conduit, l'UE la récupère et la stocke dans un buffer type file d'attente[69]. L'utilisation d'un buffer est justifiée par le conduit distribué. En effet, lors des communications réseaux il est nécessaire de bufferiser les tranches synchrones avant de les envoyer. Ainsi, les tranches reçues du composant producteur sont stockées avant d'être envoyées. A l'aide de celui-ci, on prend en compte les temps de transmission du réseau. De la même manière, avant d'être écrites dans les ports de sortie, les tranches synchrones sont stockées dans un buffer se trouvant au sein de l'US du conduit. Cette seconde bufferisation se justifie également par le fait qu'elle permet de compenser la gigue introduite par la transmission à travers le réseau. Ce type de structure de données est souvent utilisé [STO93] afin de résoudre ce problème.

Pour lire et écrire dans les ports, l'UE et l'US utilisent l'interface fournie par ces derniers qui se nomme AccesPort. Elle est décrite sur la Figure 62. Ainsi, le processus client/serveur est chargé de récupérer les tranches stockées dans le buffer de l'UE puis de les transmettre en local ou de manière distribuée vers le buffer de l'US.

Pour assurer les communications entre ces différentes unités, les mécanismes utilisés sont également basés sur des interfaces fournies et requises. L'UE fournit une interface qui permet au processus client/serveur de venir lire des données dans le buffer de l'unité d'entrée. Cette interface se nomme RecupererTranche et est décrite sur la Figure 80. Elle contient deux opérations. La première permet de lire la tranche synchrone prête à sortir du buffer d'entrée de l'UE. Cette lecture est bloquante. Si le buffer ne possède pas de tranches alors le processus se bloque jusqu'à ce qu'il y en ait une qui arrive. La deuxième opération permet de vider le buffer. Elle peut être invoquée en cas de dépassement de capacité par exemple. Les contrats qui spécifient ces opérations sont donnés sur la Figure 81.

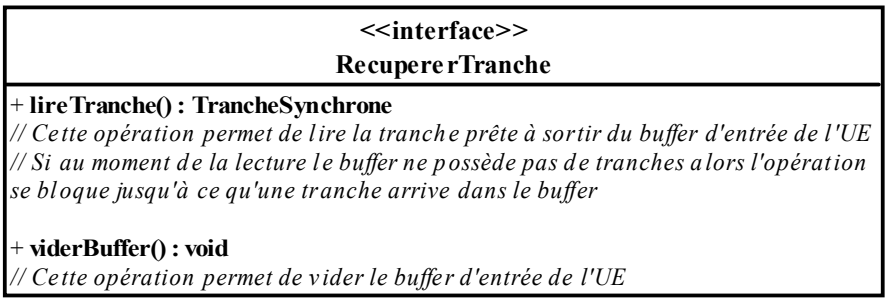


Figure 80 Interface RecupererTranche

[69] Cette structure de données est très intéressante dans nos travaux car elle permet de conserver la séquence des flux synchrones.

Contrat CRT1 : Récupérer une tranche sur le buffer de l'UE

Opération : lireTranche() : TrancheSynchrone
Préconditions :
- le buffer doit contenir au minimum une tranche

Postconditions :
- la tranche requise est retournée puis vidée du buffer

Contrat CRT2 : Vider le buffer de l'UE

Opération : viderBuffer() : void
Préconditions :
- Ø

Postconditions :
- le buffer est vide

Figure 81 Contrats de l'Interface RecupererTranche

L'US fournit une interface FournirTranche qui permet au processus client/serveur d'écrire dans le buffer de l'US la tranche qu'il vient de récupérer du buffer de l'UE. La tranche est alors écrite en fin de buffer. Cette interface est décrite sur la Figure 82. Elle fournit également l'opération viderBuffer() qui permet de vider le buffer de l'US. Les dépassements de capacité des buffers d'entrée et de sortie sont détectés par l'UC qui en informe la plate-forme. La plate-forme peut résoudre ce type de problème en donnant l'ordre de vider le buffer par exemple. Les contrats de ces opérations sont décrits sur la Figure 83. Les contrats de l'opération viderBuffer() sont le mêmes. En ce qui concerne l'opération écrireTranche(TrancheSynchrone tranche), l'écriture est toujours possible. Elle ne possède pas de préconditions et augmente la capacité du buffer d'une unité en tant que postcondition.

<<interface>> **FournirTranche**
+ **ecrireTranche(TrancheSynchrone tranche) : void** *// Cette opération permet d'écrire une tranche synchrone à la fin du buffer de sortie de l'US* + **viderBuffer() : void** *// Cette opération permet de vider le buffer de l'US.*

Figure 82 Interface FournirTranche

Contrat CFT1 : Ecrire une tranche dans le buffer de l'US

Opération : ecrireTranche(TrancheSynchrone tranche) : void
Préconditions :
- Ø
Postconditions :
- le buffer contient une tranche supplémentaire

Contrat CRT2 : Vider le buffer de l'UE

Opération : viderBuffer() : void
Préconditions :
- Ø
Postconditions :
- le buffer est vide

Figure 83 Contrats de l'Interface FournirTranche

3.2.2.2 Le Processus Client/Serveur

Les interfaces fournies par l'unité d'échange permettent respectivement de lire et d'écrire des tranches dans les buffers d'entrée et de sortie. Le processus client/serveur requiert ces interfaces afin d'invoquer ces opérations dans le but de récupérer des données en entrée et de les transmettre en sortie du conduit.

Lorsque le conduit est distribué, le processus client/serveur l'est également. Il récupère les tranches synchrones dans le buffer d'entrée et les transmet vers le buffer de sortie grâce à une communication réseau. Cette communication est réalisée à l'aide des protocoles de communication adéquats et dont les interfaces sont fournies avec le modèle. Nous ne détaillons pas ces interfaces car notre objectif n'est pas de s'attarder sur ces aspects réseaux. On considère que les manières de faire sont connues. Le paramètre réseau intervient également dans l'évaluation de la QdS. Nous essayerons plus tard de déduire des mesures qui permettent de tirer des conclusions sur l'état du réseau. Nous pensons qu'il est nécessaire de prévoir l'utilisation de plusieurs protocoles de transport et même laisser ouvert cet aspect afin qu'il puisse être étendu. Certains travaux préconisent l'utilisation de différents protocoles selon la QdS requise [CHE95], le type des données transmises et la bande passante de la liaison utilisée. Nous retenons cette approche qui nous semble très intéressante dans un contexte de gestion de la QdS des AMD.

Ce processus fournit une interface afin qu'il puisse être supervisé par l'UC et indirectement par la plate-forme d'exécution. Cette interface s'appelle ControleProcessus et propose deux opérations qui permettent de démarrer et d'arrêter le processus (cf. Figure 84). Les contrats de l'interface sont détaillés sur la Figure 85.

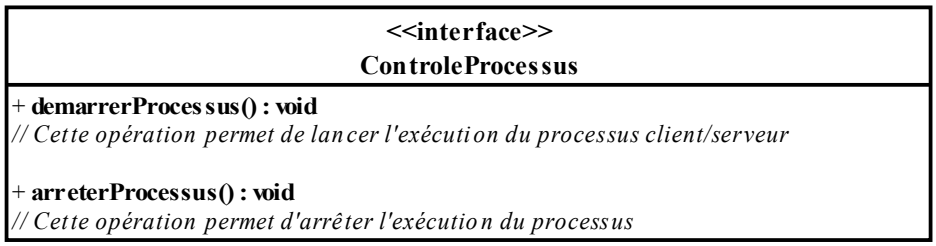


Figure 84 Interface ControleProcessus

Contrat CCPR1 : Démarrer le processus client/serveur

Opération : demarrerProcessus() : void
Préconditions :
- le processus doit être arrêté

Postconditions :
- le processus est démarré

Contrat CCPR2 : Arrêter le processus client/serveur

Opération : arreterProcessus() : void
Préconditions :
- le processus doit être en marche

Postconditions :
- le processus est arrêté

Figure 85 Contrats de l'Interface ControleProcessus

3.2.2.3 L'unité de contrôle

L'UC est chargée de la supervision du conduit par la plate-forme. Comme pour le PE, l'exécution du conduit peut être commandée par la plate-forme et ce dernier est susceptible de lui délivrer des événements qui traduisent son état de fonctionnement. Pour ce faire, l'UC fournit une interface ControleConduit qui permet ces échanges d'information. Cette interface est décrite sur la Figure 86. Elle est composée de six opérations. Les deux premières permettent de démarrer et de stopper l'exécution du conduit. Les deux suivantes permettent respectivement de vider les buffers d'entrée et de sortie du conduit. Ainsi, dans des cas critiques la plate-forme pourra exécuter ces commandes en attendant de trouver une solution. Les deux dernières permettent d'abonner et de désabonner la plate-forme d'exécution à la réception d'événements qui traduisent des informations de QdS sur l'exécution du conduit. Ce système d'événements est exactement le même que celui employé par le PE (cf. 3.1.1.2.2). L'UC du conduit peut recevoir des informations de QdS de la part de l'UE et de l'US. Ces informations sont envoyées sous la forme d'événements par chacune des deux unités. Les contrats de ces opérations sont donnés sur la Figure 87.

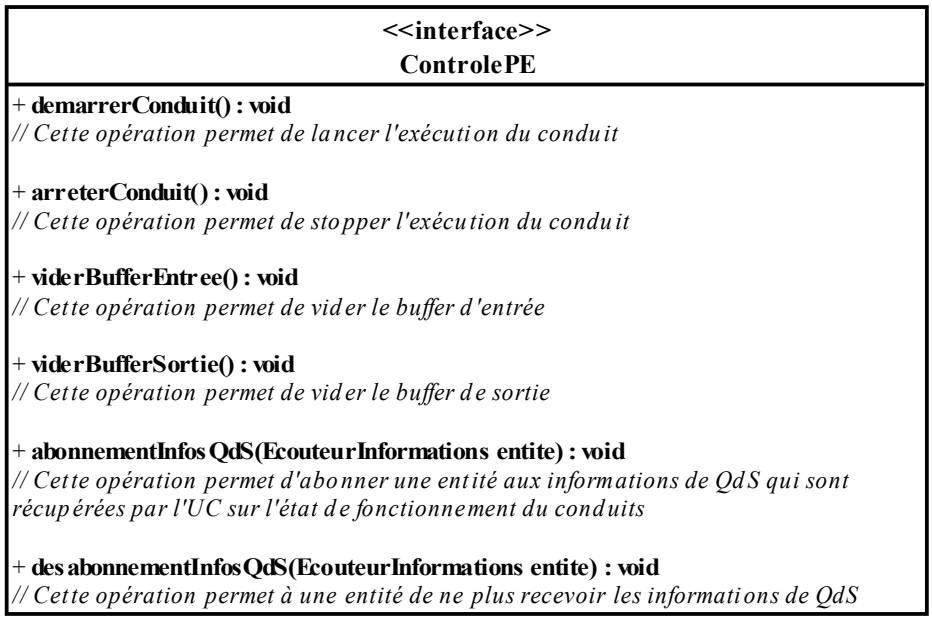


Figure 86 Interface ControleConduit

Contrat CCD1 : Lancer l'exécution du conduit

Opération : demarrerConduit() : void
Préconditions :
- le conduit doit être à l'arrêt

Postconditions :
- le conduit est mis en état de marche

Contrat CCD2 : Stopper l'exécution du conduit

Opération : arreterConduit() : void
Préconditions :
- le conduit doit être en état de marche

Postconditions :
- le conduit est arrêté

Contrat CCD3 : Vider le buffer d'entrée du conduit

Opération : viderBufferEntree() : void
Préconditions :
- Ø

Postconditions :
- le buffer d'entrée est vide

Contrat CCD4 : Vider le buffer de sortie du conduit

Opération : viderBufferSortie() : void
Préconditions :
- Ø

Postconditions :
- le buffer de sortie est vide

Contrat CCD5 : Abonnement aux informations de QdS

Opération : abonnementInfosQdS(EcouteurInformations entite) : void
Préconditions :
- l'entité ne doit pas déjà être abonnée à ce service

Postconditions :
- l'entité est ajoutée à la liste des abonnés, elle recevra désormais des événements liés à ces informations

Contrat CCD6 : Désabonnement aux informations de QdS

Opération : desabonnementInfosQdS(EcouteurInformations entite) : void
Préconditions :
- l'entité doit être abonnée à ce service

Postconditions :
- l'entité est retirée de la liste des abonnés, elle ne recevra plus les événements liés à ces informations

Figure 87 Contrats de l'Interface ControleConduit

Dans le cas d'un conduit distribué, l'UC est distribué sur deux sites et le processus client/serveur se divise en deux processus (un client et un serveur). Pour que le conduit soit utilisable, les deux parties du conduit doivent être arrêtées ou démarrées par les plates-formes locales aux sites où elles se trouvent. En ce qui concerne les événements traduisant des informations de QdS, ils sont envoyés à la plate-forme locale du site ou le morceau de conduit se trouve. Par exemple, lorsque l'événement est envoyé par l'UE, il est donc reçu par l'UC qui se trouve de ce côté et est transmis à la plate-forme locale à ce site. En fait, lorsque le conduit est distribué, chaque partie de ce dernier possède une UC locale qui permet de le commander et de recevoir son état de fonctionnement. Les méthodes de vidage des buffers doivent être invoquées sur les

UC locales respectives du conduit distribué. Ainsi, vidageBufferEntree() doit être invoquée sur la partie du conduit qui possède l'UE. De la même manière, vidageBufferSortie() doit être invoquée sur la partie du conduit qui possède l'US.

3.2.3 Communication entre les modules du Conduit

Nous résumons l'architecture du conduit par un diagramme basé sur les diagrammes de composants UML [OMG03]. Ce diagramme est représenté sur la Figure 88.

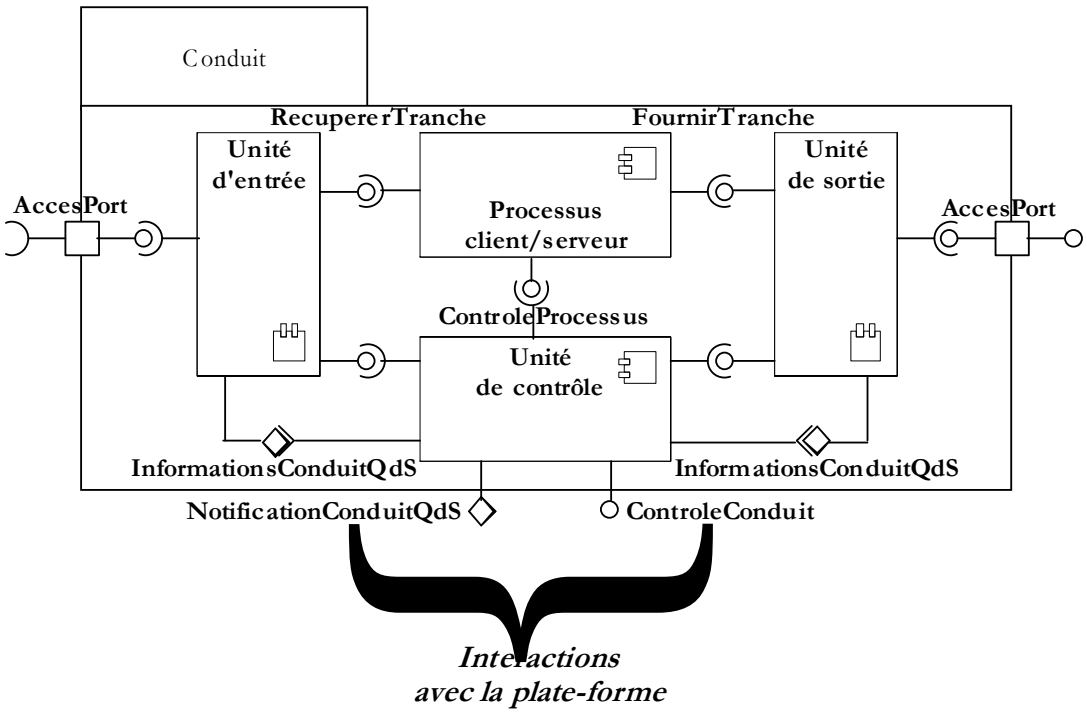


Figure 88 Communication entre les unités du Conduit

Il décrit les modes de communication entre les différentes unités du conduit. Il récapitule les modes d'interactions choisis et présentés dans cette partie. Ainsi, les unités du conduit communiquent à l'aide d'interfaces fournies/requises et de sources/puits d'événements[70]. Les événements sont utilisés pour notifier des informations tandis que les interfaces permettent l'exécution d'opérations.

Le conduit est représenté à l'aide d'un paquetage. Les ports sont représentés par des carrés qui se trouvent aux deux extrémités du conduit. Cette entité possède un port d'entrée et un port de sortie car elle est capable de transporter seulement un flux synchrone. Les communications entre les conduits, les PE/CM et les opérateurs sont

70 La notation utilisée pour les sources/puits d'événements est empruntée au modèle de composants CORBA [MAR99].

réalisées grâce aux ports d'entrée et de sortie et à leur interface AccesPort. Les unités sont représentées à l'aide de composants.

Le conduit représente un modèle générique pour le transport des données. Ce transport peut s'effectuer, selon les besoins, de façon locale ou distribuée. L'implémentation du conduit est toujours la même, son unique rôle est de transporter les tranches des flux synchrones entre un composant producteur et un composant consommateur tout en étant supervisable par la plate-forme d'exécution. La communication avec cette dernière est réalisée à l'aide d'interfaces et d'événements. Ainsi, l'interface ControleConduit permet d'exécuter des commandes sur le conduit et les événements NotificationConduitQdS permettent de notifier les états de fonctionnement du conduit à cette dernière.

3.3 Synchronisation des entités

Lorsque l'on étudie la composition de composants logiciels, il est nécessaire de s'intéresser à la coordination des entités qui sont composées. Nous avons déjà abordé un premier volet de cette étude avec la coordination des composants à l'aide du conduit (cf. § 3.2.1). Le second volet que nous proposons dans cette section est l'étude de la coordination des exécutions des entités. En effet, on doit être certain que lorsque des données sont disponibles, elles puissent être transportées le plus tôt possible vers le composant consommateur.

Lorsqu'un CM termine son traitement, les données traitées sont écrites en sortie du PE qui l'exécute. Elles sont rassemblées dans des tranches synchrones par l'US puis ces dernières sont écrites une à une dans le port de sortie du PE à destination du conduit connecté en sortie. Pour assurer un fonctionnement correct de l'ensemble, il est nécessaire que ce dernier sache quand des données sont disponibles dans le port de sortie du PE. Ainsi, il peut les récupérer et les transmettre vers le prochain PE. De la même manière, lorsque le conduit écrit dans son port de sortie une tranche synchrone à destination d'un PE, il faut que le PE sache quand ces données sont disponibles dans le port de sortie du conduit. Ainsi, il peut les récupérer et délivrer au CM qu'il contient celles qui sont destinées à être traitées.

Notre considération des contraintes temporelles des flux nous amène à penser que les entités du modèle défini doivent être synchronisées. En effet, la possibilité de transporter des flux à contrainte faible signifie que le temps qui sépare deux tranches est a priori inconnu et imprévisible. Il est de fait difficile de connaître à l'avance les instants où les tranches seront disponibles dans les ports de sortie des entités. Nous devons donc fournir un moyen qui va permettre d'obtenir cette connaissance.

L'architecture logicielle proposée étant guidée par les données, la solution à ce problème est donc de connaître les moments où des données sont disponibles dans les ports de sortie. Nous proposons d'utiliser une solution basée sur une communication par événements que nous décrivons à l'aide du diagramme de séquence de la Figure 89. Ce diagramme est établi entre un conduit connecté en entrée d'un PE. Il peut facilement être adapté à un conduit connecté en sortie d'un PE mais aussi entre un conduit connecté en entrée d'un opérateur et ainsi de suite. La seule différence dans ce dernier cas d'utilisation est que les opérateurs n'utilisent pas d'unités d'entrée mais le principe reste le même. Il faut préciser que la séquence décrite sur la Figure 89 se répète pour toutes les tranches écrites une à une dans les ports de sortie des conduits connectés en entrée d'un PE.

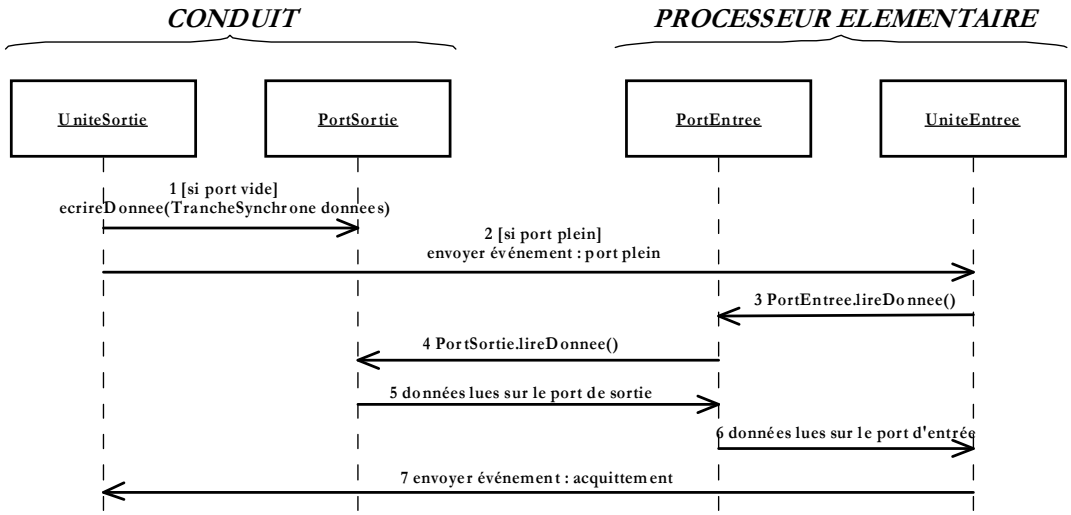


Figure 89 Diagramme de Séquence

Les buffers de sortie du conduit sont vidés par l'US afin d'écrire les tranches successivement dans les ports de sortie et donc de les rendre disponibles pour le PE auquel il est connecté. Lorsque l'US retire du buffer la tranche prête à sortir, elle écrit cette dernière dans le port de sortie à condition que ce dernier soit vide (1). En effet, l'opération d'écriture dans un port est bloquée jusqu'à ce qu'il soit vidé. A la fin de cette opération, le port est à nouveau plein. L'unité de sortie envoie un événement à destination de l'unité d'entrée du PE pour lui notifier qu'une tranche synchrone est disponible dans le port de sortie du conduit (2). A la réception de cet événement par l'unité d'entrée, cette dernière procède à une opération de lecture sur le port d'entrée auquel est connecté le conduit (3). Cette opération est bloquée jusqu'à ce qu'une tranche soit disponible dans le port d'entrée. Cette lecture déclenche une autre opération de lecture à l'initiative du port d'entrée sur le port de sortie du conduit afin de récupérer la tranche qu'il contient (4). Cette tranche est ainsi retournée par l'opération lire-

Donnee() puis écrite dans le port d'entrée du PE (5). Cette séquence débloque l'opération de lecture en attente depuis l'étape 3. La tranche est ainsi récupérée par l'UE du PE (6). Suite à quoi, cette dernière informe l'unité de sortie du conduit que la tranche a bien été récupérée à l'aide d'un événement d'acquittement (7). Cette notification permet de s'assurer que l'événement envoyé lors de l'étape 2 ne peut être manqué et donc on évite le risque d'interbloquer la propagation des flux synchrones dans une AMD. Cet événement de notification indique à l'US du conduit qu'elle peut à nouveau transférer une tranche du buffer de sortie vers le port de sortie.

Puisque plusieurs conduits peuvent être connectés en entrée d'un PE ou même d'un opérateur, nous utilisons dans ces entités (dans l'UE pour le PE) une file d'événement qui va permettre de traiter séquentiellement les événements reçus. Ainsi, dans ces cas de connexion on est certain de traiter tous les événements sans risquer d'en manquer. Nous allons voir dans la partie suivante que la taille de cette file d'événements va constituer une information intéressante sur l'état de fonctionnement du CM utilisé dans un PE.

Ce type d'interactions entre un conduit et un composant de l'application nous permet de compléter le mode de communication décrit par la Figure 64. Désormais, une communication bidirectionnelle à l'aide d'événements est établie entre l'US de chaque conduit connecté en entrée d'un PE et l'UE de ce dernier. De la même manière, entre l'US d'un PE et l'UE du conduit connecté en sortie et aussi entre les opérateurs et les conduits.

Lors de la création d'un PE à l'aide des conduits connectés en entrée et en sortie, un système d'abonnement est élaboré entre les UE et les US de chaque entité afin de permettre cette communication. Lorsqu'un conduit est déconnecté en entrée ou en sortie, il se désabonne de ce service d'événements auprès de l'unité correspondante qui ne traitera plus les événements de ce conduit. Les nouveaux conduits connectés devront s'abonner auprès des unités correspondantes. Quand un PE est déconnecté pour, par exemple, être remplacé par un autre ce dernier envoie aux conduits connectés en entrée les acquittements non encore envoyés afin de ne pas bloquer l'exécution de l'AMD. Après quoi le PE se désabonne de ce service.

3.4 Gestion des Informations de QdS

Avant de refermer ce chapitre, nous allons nous intéresser aux informations de QdS que les entités fonctionnelles sont capables de délivrer à la plate-forme. A l'aide de ces dernières, nous allons voir que la plate-forme peut déduire des états de fonctionnement. Ces informations sont liées à l'occupation mémoire de ces entités. On va

obtenir ces informations à l'aide de mesures effectuées sur la taille des différents buffers contenus dans les entités du modèle.

En ce qui concerne le PE, on se propose de mesurer la taille de la file d'attente des événements se trouvant dans l'UE. Cette taille traduit la propagation des données entre les conduits d'entrée et le PE. Si la taille de cette file croît, on peut alors déduire que le CM n'arrive pas à suivre puisqu'il ne consomme pas suffisamment de tranches en entrée. Si cet événement se répète, on peut aboutir à une situation de surcharge de la mémoire ce qui peut avoir pour effet au niveau de l'AMD une augmentation du temps de latence de bout en bout. On peut donc déduire que le CM ne suit pas et donc qu'il faut le remplacer par un autre capable de réaliser la même tâche, quitte à devoir perdre en qualité. Nous pouvons également interpréter le cas inverse c'est à dire le cas où cette file d'attente se trouve dans une situation de famine sans aucune donnée à transférer. Dans ce cas là, on peut déduire que le CM consomme très rapidement les tranches dans les ports et donc que le niveau de qualité qu'il fournit est peut être sous-évalué. Ceci voudrait dire qu'il pourrait fonctionner avec un niveau de qualité supérieur et donc qu'éventuellement la plate-forme pourrait augmenter la qualité du service fourni. Un CM fonctionnant correctement avec un niveau de qualité adéquat devrait posséder une taille de file d'attente qui finit par se stabiliser.

La connaissance de ces seules informations n'est évidemment pas suffisante pour que la plate-forme puisse prendre des décisions quant à la reconfiguration possible de l'AMD. Par conséquent, nous proposons d'effectuer des mesures également sur la taille des buffers d'entrée et de sortie du conduit dans un contexte local mais aussi dans un contexte distribué. Ces mesures supplémentaires vont nous permettre de déduire les états de fonctionnement des CM d'une AMD. Ainsi, si le buffer de sortie d'un conduit est en état de famine, on peut déduire que le CM contenu dans le PE suivant arrive à suivre puisque aucune donnée ne reste stockée dans ce buffer. Il est même peut-être sous-évalué si sa file d'attente se retrouve également dans le même état. Par contre, si ce même buffer croît rapidement, la situation est plus inquiétante car cela signifie que le CM suivant n'arrive pas à suivre et qu'il est probablement surévalué.

Dans un contexte distribué, ce type de mesures peut également se révéler fort intéressant. En effet, si on mesure la taille du buffer d'entrée du conduit c'est-à-dire celui qui est chargé de stocker les tranches synchrones juste avant de les transmettre, on peut déduire des informations intéressantes sur la bande passante de la liaison réseau. Dans ce cas là, une famine du buffer d'entrée se traduira par des transmissions réseaux performantes puisque les données ne s'accumulent pas en entrée du conduit. Dans le cas contraire, un buffer qui croît rapidement indiquerait une bande passante

insuffisante de la liaison réseau. Ces informations sont intéressantes car elles permettent d'ajuster la qualité des données transmises.

Nous montrons donc les différentes mesures que l'on peut effectuer sur les buffers des entités fonctionnelles. Cependant, ces mesures ne sont jamais utilisées seules car elles ne peuvent pas permettre de conclure tout le temps. Par contre, si on les corrèle entre elles alors dans ce cas on constitue des informations de QdS intéressantes pour la plate-forme.

4 Synthèse

Ce chapitre a introduit le modèle de composants que nous avons spécifié afin de répondre à la problématique que nous nous sommes fixée à savoir l'implémentation d'AMD reconfigurable dynamiquement. Ces reconfigurations sont nécessaires pour la gestion de la QdS [LAP06]. Ainsi, le modèle de composants Osagaia est spécialement conçu pour la spécification et l'implémentation des AMD. Les entités qui le composent ainsi que les règles que nous avons fixé aux architectures logicielles ainsi obtenues se justifient par le passage des graphes fonctionnels aux graphes de transition ainsi que par l'introduction du modèle Korrontea. Cette transformation permet de dériver les spécifications fonctionnelles en des spécifications plus proches de l'implémentation. Ces points essentiels de notre approche mettent en évidence un ensemble de spécifications que l'utilisation du modèle Osagaia devra assurer.

Nous avons choisi comme style une approche basée sur les composants logiciels [SZY02] car ce paradigme amène de la flexibilité pour la manipulation des architectures. En ce sens, nous pensons qu'ils constituent des entités intéressantes pour la reconfiguration dynamique des applications. L'étude d'un grand nombre de travaux sur ce thème nous a permis de valider ce choix [ACC02], [ALD02], [BEU05], [BRU02], [CHE03], [CRN02], [DIO95], [HEI01], [KON00], [LAY04], [MED00], [OCC03], [SEG02]. Nous nous sommes également appuyés sur quelques grands principes de la programmation par aspects [BOU01], [DUC02], [KIC97], [LOP95] qui semblent intéressants afin de favoriser la réutilisation des composants logiciels. Ainsi, nous avons étudié les propriétés fonctionnelles et non-fonctionnelles que notre modèle se doit de proposer. Nous avons, de la sorte, mis en évidence la nécessité de disposer de deux types de composants : les composants fonctionnels et les opérateurs. Les opérateurs sont introduits afin de répondre à un certain nombre de spécifications fonctionnelles telles que la conservation de la synchronisation entre plusieurs flux (synchronisation inter-flux). Ces opérateurs sont destinés à être appliqués sur les flux synchrones du modèle Korrontea. Les composants fonctionnels adressent, quant à eux, les parties

métier des AMD c'est-à-dire l'ensemble des fonctionnalités requises par ces dernières. Ces fonctionnalités sont identifiées par la phase de spécification des AMD préalable à notre approche.

L'implémentation et l'exécution de ces entités fonctionnelles nécessitent également la mise en place de propriétés non-fonctionnelles. De fait, nous avons trouvé intéressant d'introduire le PE qui n'est autre qu'un conteneur de composants logiciels. Ainsi, il est censé implémenter un ensemble de services nécessaires à l'exécution des composants fonctionnels et plus généralement de l'ensemble de l'AMD. Les spécifications fonctionnelles nous ont également convaincus dans l'utilisation d'un connecteur dont le seul but est de connecter les composants et les opérateurs entre eux et de transporter les flux synchrones au sein des AMD. Nous avons nommé cette entité le conduit et nous l'avons défini comme l'entité distribuée du modèle dés lors qu'il autorise indifféremment le transport local ou distribué des flux. Chacune de ces deux entités permet de proposer des mécanismes qui évitent les sources de désynchronisation identifiées préalablement [BOU05], [BOX05]. Pour ce faire, nous avons intégré le modèle Korrontea et ses politiques de synchronisation au modèle de composants Osagaia. De plus, ce modèle autorise une implémentation des AMD conformes aux spécifications fonctionnelles introduites par les graphes fonctionnels.

De plus, les entités du modèle Osagaia sont entièrement supervisables par la plate-forme d'exécution. Ainsi, l'architecture d'une AMD peut être modifiée afin d'approcher le plus possible la QdS requise. Les opérateurs, les conduits et les PE fournissent des interfaces à la plate-forme d'exécution afin que cette dernière puisse exécuter des commandes sur ces entités. En fait, ces interfaces permettent de contrôler leur cycle de vie. Ces entités fournissent également à la plate-forme des états d'exécution afin de traduire leur mode de fonctionnement. Ces informations sont précieuses pour la plate-forme car elles sont susceptibles de l'aider dans sa tâche de diagnostic et de plan d'action pour la gestion de la QdS. Dans des cas critiques de fonctionnement, ces informations sont notifiées à la plate-forme sous la forme d'événements. Ainsi, la communication entre les entités d'une AMD et la plate-forme est réalisée à l'aide d'interfaces et d'événements.

La communication entre les entités du modèle est également réalisée à l'aide d'interfaces et d'événements. Les interfaces permettent de propager les flux synchrones à l'intérieur des entités du modèle. Les événements, quant à eux, permettent de synchroniser l'exécution des entités et donc d'éviter les risques d'interblocage des flux synchrones. Ainsi, les entités sont coordonnées entre elles selon le modèle producteur/consommateur [MAL94].

La dernière tâche qui nous incombe est de définir une implantation détaillée des AMD à l'aide des entités du modèle Osagaia. En effet, maintenant que nous connaissons les entités ainsi que leurs règles d'utilisation, nous pouvons définir précisément l'architecture des AMD. Cette implantation va également nous permettre d'introduire explicitement la localisation de ces entités sur les différents sites d'une application. Ainsi, nous introduisons la dimension réseau. Nous présentons cette implantation et ses différentes caractéristiques dans la prochaine partie de ce mémoire.

Partie 4 – Implémentation des Applications Multimédias Distribuées

Nous abordons dans cette dernière partie l'implémentation proprement dite des AMD à partir du modèle Osagaia. Grâce à ce modèle, nous pouvons décrire précisément l'architecture des AMD en termes de CM, de PE, d'opérateurs et de conduits. L'objectif est d'obtenir une vue directement implémentable des AMD dans laquelle soit introduite de manière explicite la dimension réseau. Nous définissons donc la répartition des différentes entités entre les sites sur lesquels l'AMD doit être déployée. Les conduits distribués du modèle Osagaia permettront ensuite d'établir ces connexions distantes. L'introduction de cette caractéristique va mettre en évidence la nécessité de proposer un nouvel opérateur de flux synchrones afin de rendre possible cette implémentation. Tous ces aspects sont définis à l'aide des graphes que nous appelons graphes d'implantation et que nous présentons dans le chapitre 7.

Le dernier chapitre de cette partie donne un rapide aperçu du prototype que nous avons développé afin de valider les différents modèles et leurs principales caractéristiques. Le langage Java s'est imposé en raison de sa portabilité notoire. Ce choix se justifie par la forte hétérogénéité du réseau Internet auquel nous nous adressons qui impose de disposer d'implémentations portables. Nous avons utilisé un exemple simple d'AMD afin de tester les reconfigurations dynamiques, la synchronisation multimédia mais aussi l'efficacité de ce type d'implémentation et des modèles définis.

Chapitre 7 – Graphes d'Implantation

« Mais dès que je perçois ou que je présuppose que cela implique un rapport à un état précédent d'où la représentation dérive d'une règle, alors je me représente quelque chose […], c'est-à-dire que je reconnais un objet que je dois poser dans le temps à une certaine place déterminée et qui ne peut être autrement en raison de l'état précédent. »

Critique de la raison pure, Emmanuel Kant, 1787

La partie précédente a permis d'introduire les modèles nécessaires à la description des flux de données et de l'architecture logicielle pour les AMD. Cette étape étant proche de l'implémentation, nous devons pouvoir en dériver les modèles nécessaires pour leur implémentation. La dernière étape consiste donc à donner une description précise de l'architecture des AMD à l'aide des entités d'implémentation que nous avons introduites avec Osagaia. Cette description est faite à l'aide de graphes dont la structure est justifiée par les modèles Korrontea et Osagaia. Ainsi, les nœuds seront matérialisés par des PE qui contiendront chacun un CM implémentant une fonctionnalité basique de l'AMD ou par des opérateurs de flux synchrones. Les arcs, quant à eux, seront matérialisés par des conduits chargés du transport des flux synchrones entre les PE ou les opérateurs. Ces graphes préciseront également les différents sites d'une AMD en identifiant les localisations des différentes entités.

Chaque entité du modèle Osagaia étant supervisable, la plate-forme pourra alors adapter la structure des AMD en fonction des exigences de QdS à respecter. Les graphes d'implantation présentés dans ce chapitre contiendront alors toutes les configurations possibles d'une AMD. Ces graphes s'obtiennent à partir des graphes de transition à l'aide du modèle de composants Osagaia. La dimension réseau fait également son apparition à ce stade. L'architecture décrite par ces graphes est de type « pipes & filters » [GAN94], [GAR94].

1 Introduction

Le modèle Osagaia propose de décrire les AMD comme une interconnexion de PE, d'opérateurs de flux synchrones et de conduits (cf. chapitre 6). Nous avons défini à l'aide de ce modèle une composition horizontale [ROM06] de ces différentes entités afin de permettre une implémentation des AMD. Ce modèle est basé sur des commu-

nications à l'aide d'interfaces et d'événements afin de coordonner l'ensemble de ces entités (cf. chapitre 6).

L'objectif de ce chapitre est de proposer des graphes qui décrivent de manière précise l'architecture distribuée de ces applications. L'obtention de ces graphes se base sur les graphes de transition qui donnent une vue de l'application en termes de composants logiciels, d'opérateurs et de flux synchrones. Les entités qu'ils décrivent peuvent maintenant être encapsulées dans celles du modèle Osagaia puisqu'il a été développé à partir de ces informations. Enfin, les informations concernant la distribution de l'application à travers différents sites seront transcrites sur ces graphes afin de définir les points de l'architecture où vont être utilisées les entités spécialisées dans cette tâche à savoir les conduits distribués. Nous définissons des règles de transformation qui vont nous permettre d'obtenir les graphes d'implantation. Nous allons voir que ces transformations vont imposer de définir un opérateur de flux synchrones supplémentaire.

Ces graphes considèrent tant les aspects fonctionnels que non-fonctionnels des AMD. Les premiers sont implémentés par les PE et les CM ainsi que par les conduits. Les seconds sont implémentés par des opérateurs de flux synchrones. Ainsi, l'ensemble des spécifications des AMD définies en amont pourra être obtenu.

2 Les Graphes d'Implantation

Il s'agit d'une transformation des graphes de transition ajoutant la dimension réseau des AMD et les entités fournies par le modèle Osagaia. Un ensemble de règles permet d'obtenir les graphes d'implantation et ainsi de disposer d'une vue des AMD directement déployable et implémentable.

Les PE encapsulent des CM qui implémentent les fonctionnalités des AMD devant être réalisées de manière logicielle. Les fonctionnalités fournies de façon matérielle sont définies sur ces graphes à l'aide des dépendances matérielles introduites sur les graphes de transition (cf. chapitre 6). Ces dépendances sont alors localisées sur des sites précis de l'application et signifient que ces sites doivent se doter du matériel désigné afin de permettre l'exécution de l'AMD.

La Figure 90 résume le principe de la transformation présentée dans ce chapitre.

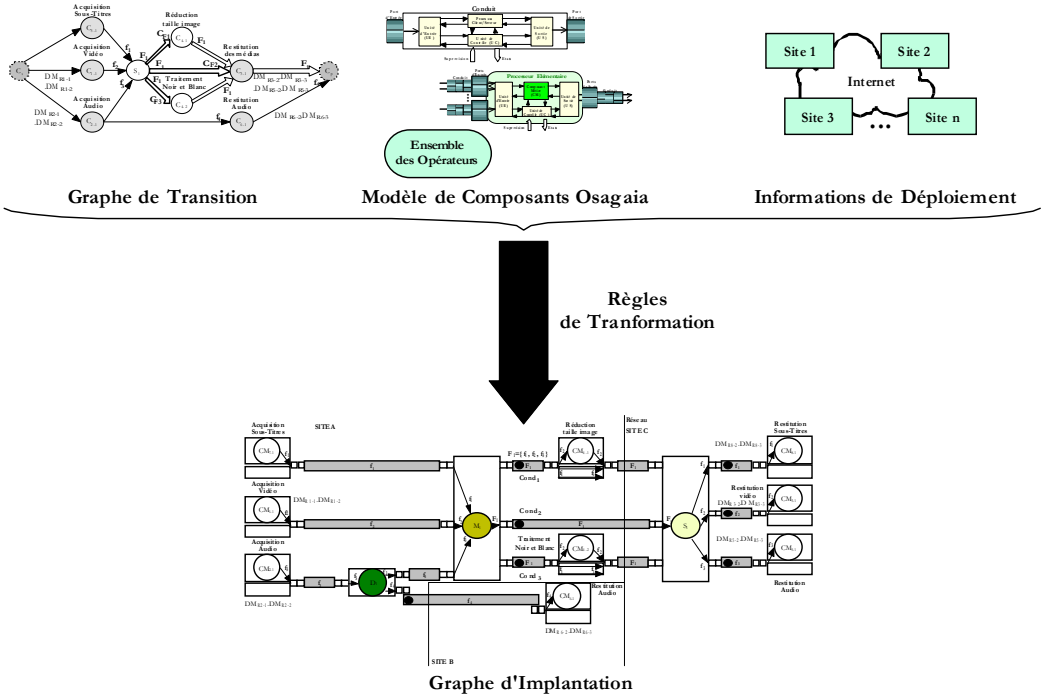


Figure 90 Définition des Graphes d'Implantation

Les informations apportées par les graphes de transition, le modèle Osagaia et les sites de déploiement permettent de déduire ces graphes.

2.1 Représentation des Graphes d'Implantation

Les graphes d'implantation fournissent une vue des AMD directement implémentable à l'aide des entités du modèle Osagaia. Sur ces graphes, une AMD est décrite à l'aide de PE (encapsulant chacun un CM), d'opérateurs de flux synchrones et de conduits. Les graphes d'implantation, notés GI(CL, C) possèdent les mêmes propriétés que les graphes de transition. L'ensemble CL désigne les composants logiciels (PE/CM et opérateurs) qui constituent un tel graphe tandis que l'ensemble C désigne les conduits utilisés.

Ces graphes se centrent sur la notion de CM utilisée pour implémenter chaque fonctionnalité atomique d'une AMD.

2.1.1 Les Nœuds

Chaque composant logiciel C_{i-j} identifié sur les graphes de transition décrit une fonctionnalité atomique d'une AMD (c'est une fonctionnalité de base d'une AMD, cf. chapitre 3 et 4). Chacun de ces composants sera implémenté à l'aide d'un CM puis encapsulé dans un PE chargé d'assurer son exécution. La structure des graphes de transi-

tion est conservée c'est-à-dire que les PE sont assemblés en parallèle ou en séquence. Ces CM représentent la partie fonctionnelle de l'AMD liée à ses aspects métier, ils sont notés CM_{i-j} et appartiennent à l'ensemble CL.

Chaque opérateur de flux identifié sur les graphes de transition apparaît également sur les graphes d'implantation. Ils constituent des unités d'implémentation particulières chargées de mettre en œuvre des spécifications fonctionnelles de l'AMD. Ces opérateurs sont également assemblés en parallèle ou en séquence avec les PE. Ils représentent la partie non-fonctionnelle de l'AMD, ils sont notés de la même manière que sur les graphes de transition et appartiennent à l'ensemble CL. Ainsi, nous distinguons les différents aspects d'une AMD [BOU01], [KIC97], [LOP95].

Les nœuds des graphes d'implantation décrivent les CM et les opérateurs. Ces graphes débutent par des CM producteurs de données et aboutissent à des CM consommateurs. Ces CM particuliers représentent les origines et destinations des flux synchrones. Les flux synchrones sont transportés dans des conduits représentés par les arcs du graphe.

2.1.2 Les Arcs

Les arcs des graphes d'implantation représentent les conduits du modèle Osagaia chargés de transporter les flux synchrones dans les AMD. Ils appartiennent à l'ensemble C qui représente l'ensemble des conduits d'un graphe.

Les graphes de transition introduisent plusieurs configurations différentes pour une même fonctionnalité à l'aide des arcs conditionnels. Ces derniers sont supprimés des graphes d'implantation. Cependant, les différents choix de configuration sont quand même représentés sur ces graphes. Elles sont modélisées en conservant les conditions associées à ces arcs qui permettent de définir les contraintes à respecter dans le choix d'une configuration plutôt qu'une autre. Elles sont exclusives et notées $Cond_i$. Les contraintes dans l'utilisation de certains composants sont conservées également et notées O_{CL}, où CL représente l'ensemble des nœuds du graphe. Elles traduisent des obligations dans l'utilisation du composant spécifié en amont.

Les caractéristiques d'orientation et de cycle sont les mêmes pour les graphes d'implantation (cf. § 2.4 du chapitre 4). Nous ne revenons donc pas sur ce point.

Nous donnons sur la Figure 91, les représentations des arcs et des nœuds utilisées par les graphes d'implantation. Chaque représentation est détaillée dans le paragraphe suivant. Les dépendances matérielles sont notées sous la même forme que sur les graphes de transition. Nous verrons que sur les graphes d'implantation, elles sont localisées sur les sites concernés.

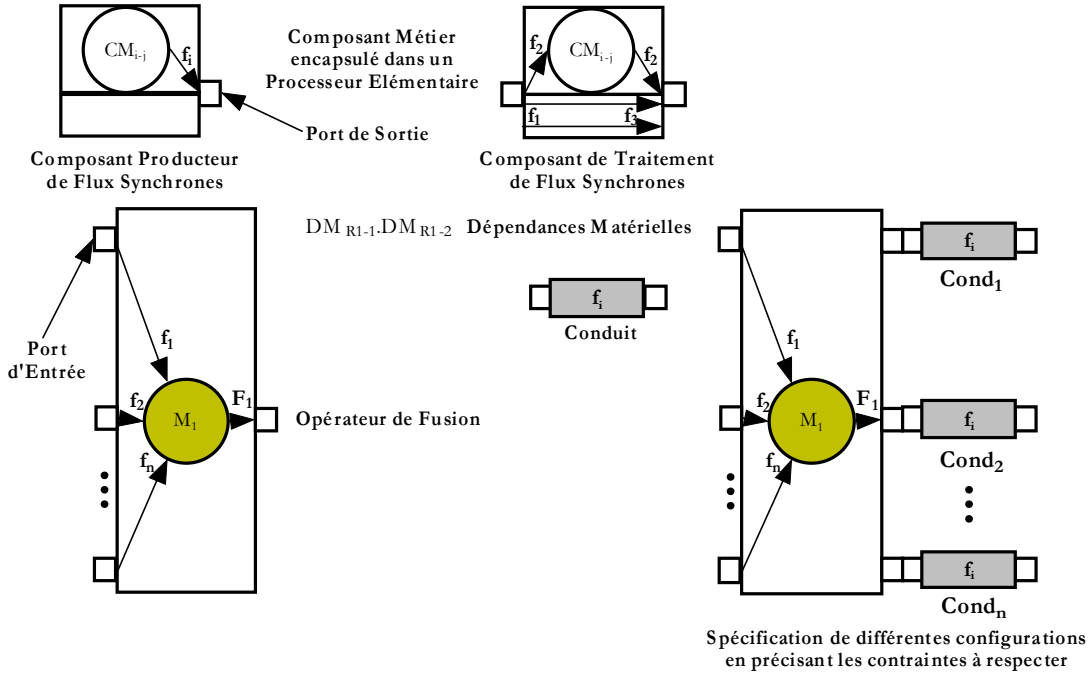


Figure 91 Représentation des éléments des Graphes d'Implantation

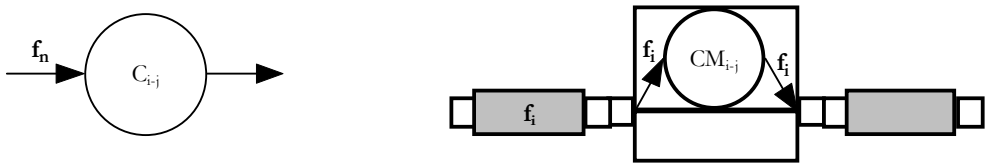
Il faut noter que sur cette figure, nous donnons un exemple de représentation d'un opérateur. L'opérateur utilisé est celui de fusion. La représentation des autres opérateurs est donnée dans les règles de transformation qui permettent de les introduire.

2.2 Obtention des Graphes d'Implantation

Les graphes d'implantation sont obtenus par transformation des graphes de transition auxquels on intègre les entités définies par le modèle Osagaia. Ils utilisent également des informations de distribution des AMD. Nous présentons dans cette section les règles qui permettent de passer d'une représentation à l'autre. Chaque règle est décrite par des représentations possibles équivalentes.

Règle 1 *Les Composants Métier*

Chaque composant logiciel de type fonctionnel identifié sur les graphes de transition est encapsulé dans un PE. La Figure 92 décrit cette représentation. Les flux entrants et sortants des PE peuvent être primitifs ou composés. La propagation des flux à l'intérieur du PE est réalisée à l'aide des ports d'entrée/sortie représentés de chaque côté du PE par des petits carrés.



Composant C_{i-j} **Processeur Elémentaire**

Figure 92 Représentation des CM et des PE

Les flux prépondérants sont identifiés en entrée du PE en désignant les conduits qui les transportent.

Règle 2 *Identification des Flux Prépondérants*

Un flux prépondérant est un flux primitif ou composé désigné en entrée d'un PE. Ce flux est désigné en apposant une pastille noire sur le conduit qui contient ce flux en entrée du PE. Un PE possède en entrée au plus un conduit transportant un flux prépondérant. Lorsqu'aucun flux prépondérant n'est indiqué en entrée d'un PE, ce dernier constitue une source localisée qui crée un nouveau flux en sortie. La Figure 93 donne une représentation d'un PE ayant un flux prépondérant en entrée.

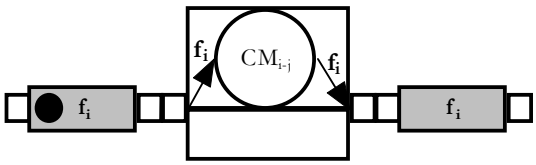


Figure 93 Flux prépondérant en entrée d'un PE

Un flux prépondérant conserve son identité en sortie du PE. Ce dernier ne change pas cette information, il applique seulement un traitement sur les données qui constituent ce flux. Lorsqu'un PE ne possède pas de flux prépondérant en entrée, on a affaire à une source localisée.

Règle 3 *Les Sources Localisées*

Une source localisée est un CM dont le but est de créer ou de capturer des flux synchrones. Un tel CM se distingue par le fait que le PE ne possède pas de flux prépondérant en entrée. La Figure 94 donne la représentation d'une source localisée. Les informations liées aux flux produits sont donc initialisées ou modifiées par cette dernière. Le ou les flux sortants d'une source localisée sont différents des flux entrants. Les graphes d'implantation ont pour origine des sources localisées.

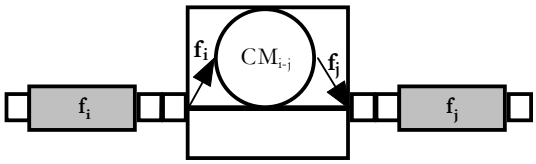


Figure 94 Représentation des Sources Localisées

De plus, ces sources ont pour particularité d'être localisées en un site particulier de l'application. Nous allons voir que cette information prend tout son sens avec les graphes d'implantation puisque les différents sites d'une AMD sont identifiés.

Les dépendances matérielles identifiées sur les graphes de transition sont conservées sur les graphes d'implantation. Elles sont associées aux sites de chaque AMD et ainsi listent de manière exhaustive le matériel requis par chacun d'eux afin d'assurer un fonctionnement correct.

Les autres types de composants représentés sur ces graphes sont les opérateurs de flux synchrones.

Règle 4 *Opérateur de Fusion*

L'opérateur de fusion sur les graphes d'implantation est représenté sur la Figure 95. A partir de plusieurs flux synchrones de même site, il produit un flux composé en sortie noté F_1. On notera sur ces graphes la constitution du flux composé de la façon suivante : $F_1=\{f_1, f_2, \ldots, f_n\}$.

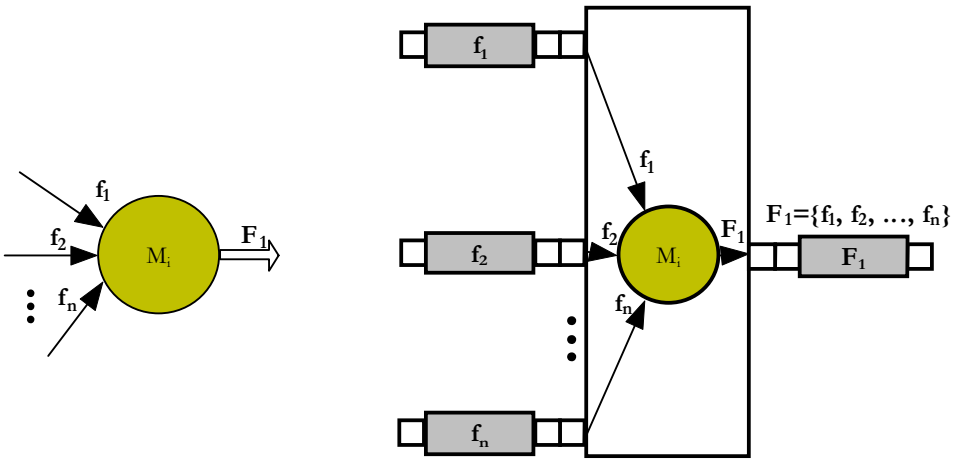


Figure 95 Opérateur de Fusion

L'opérateur de fusion permet de rassembler dans un même flux composé plusieurs flux synchrones reliés entre eux par des relations de synchronisation inter-flux. On définit de la même manière l'opérateur inverse de séparation afin d'éclater un flux composé en plusieurs flux primitifs.

Règle 5 *Opérateur de Séparation*

L'opérateur de séparation sur les graphes d'implantation est représenté sur la Figure 96. Il permet de séparer le flux composé d'entrée en plusieurs flux primitifs.

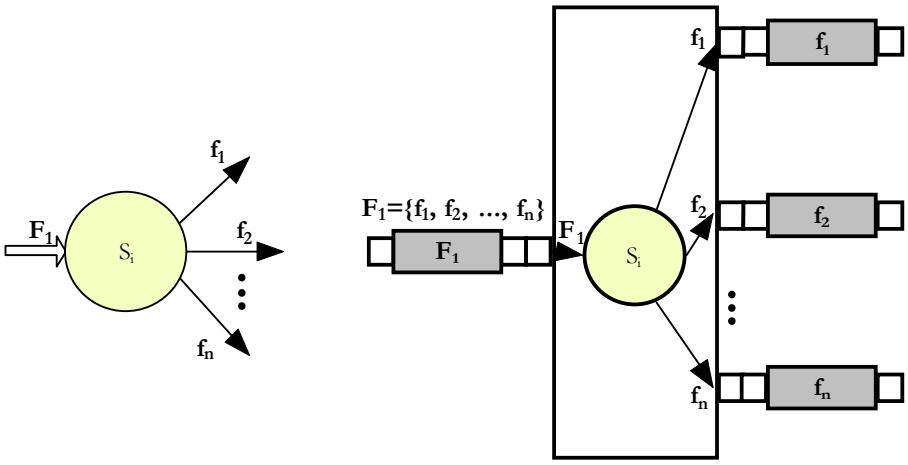


Figure 96 Opérateur de Séparation

Ces deux opérateurs concernent la synchronisation inter-flux. L'opérateur de fusion permet d'obtenir des flux composés. Nous allons maintenant voir la représentation du traitement des flux composés dans les PE.

Règle 6 *Traitement des Flux Composés*

Un PE est susceptible d'accepter en entrée un flux composé même si le CM qu'il contient ne traite que certains flux primitifs. Cette solution évite de perdre les relations de synchronisation inter-flux. La Figure 97 représente le traitement des flux composés. L'intérieur du processeur élémentaire détaille les flux traités par le CM (f_2 sur la figure) et les flux en transit (f_1 et f_3). Les politiques de synchronisation du modèle Korrontea permettent de recréer en sortie le flux F_1. Bien entendu le CM peut traiter plusieurs flux appartenant à un ou plusieurs flux composés. De même, un PE peut accepter plusieurs conduits en entrée mais n'en produit qu'un seul en sortie (cf. chapitre 6). Un flux peut également être produit en sortie du CM à partir de plusieurs flux d'entrée. Les graphes d'implantation donnent un niveau de détail qui permet de donner des spécifications d'implémentation précises en termes de flux traités et de flux en transit dans les PE mais aussi dans les opérateurs (cf. règles 4 et 5 et règles suivantes).

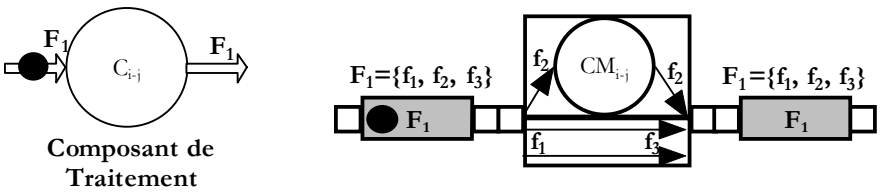


Figure 97 Traitement des Flux Composés

Lorsque plusieurs CM se partagent en parallèle le traitement des flux primitifs constituant un flux composé, il est fait appel à l'opérateur de disjonction. Un opérateur de conjonction permettra ensuite de fusionner à nouveau les flux ainsi traités pour reconstituer un flux composé sans perte des liens de synchronisation inter-flux.

Règle 7 *Opérateur de Disjonction*

Cette règle est en fait une généralisation de la règle 6 puisque avec cet opérateur, on permet le traitement parallèle de plusieurs flux primitifs. La Figure 98 donne une représentation de cet opérateur sur les graphes d'implantation. Les flux sortant d'un tel opérateur sont forcément prépondérants afin de pouvoir reconstituer, après traitement, le flux composé tel qu'il existait en amont.

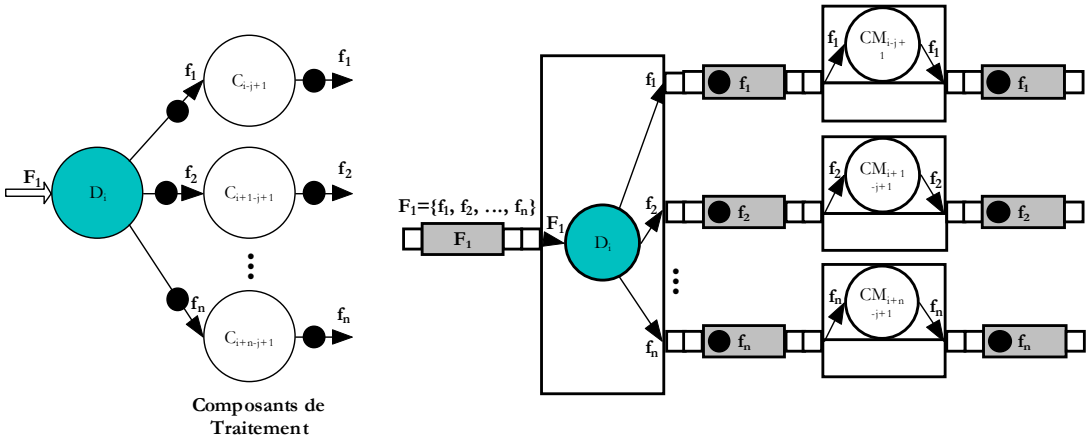


Figure 98 Opérateur de Disjonction

La fin d'une disjonction est signalée par une conjonction c'est-à-dire que les traitements parallèles sont terminés et les flux ainsi traités se rejoignent en un nœud du graphe. Nous avons introduit un opérateur de conjonction qui réalise la fonction inverse de l'opérateur de disjonction. Il permet donc de reconstituer le flux composé d'origine.

Règle 8 *Opérateur de Conjonction*

A la fin des traitements parallèles de plusieurs flux synchrones, il est nécessaire de reconstituer le flux composé tel qu'il existait avant la disjonction. Cette opération est réalisée grâce à un opérateur de conjonction. Il est représenté sur la Figure 99. Cet opérateur n'est utilisé que si l'on a utilisé en amont celui de disjonction. Cette condition est spécifiée à l'aide de la contrainte d'utilisation O_{Di}.

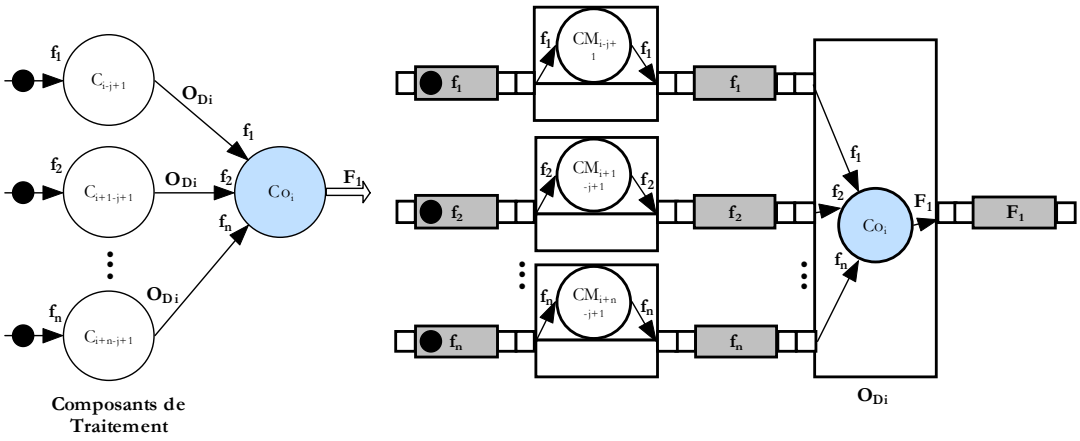


Figure 99 Opérateur de Conjonction

La règle de transformation suivante concerne l'introduction d'un nouvel opérateur de flux synchrones. Cet opérateur fait son apparition à ce stade car les graphes d'implantation ont pour but de donner une vision précise de l'implémentation des AMD. C'est un opérateur de duplication utilisé pour dupliquer les flux synchrones dans les AMD.

Règle 9 *Opérateur de Duplication*

Certains flux synchrones sont destinés à être envoyés vers plusieurs CM pouvant être distribués sur plusieurs sites d'une AMD. Il est donc nécessaire de pouvoir dupliquer ces flux afin de les envoyer vers leurs destinations respectives. Pour introduire cette possibilité, nous introduisons un opérateur de flux supplémentaire noté Du_i et appelé duplication. Il accepte un flux synchrone primitif ou composé en entrée et produit en sortie n flux synchrones qui sont des duplications exactes du flux d'entrée, seul leur nom change afin de pouvoir les différencier dans l'AMD. Cette transformation est représentée par la Figure 100. Afin d'éviter toute ambigüité, rappelons qu'un composant ne peut pas produire deux flux de sortie, de sorte que les flux sortants de C_{i-j} sur le dessin ci-dessous représentent bien le même flux destiné à être envoyé vers différents composants. L'opérateur de duplication permet de dupliquer le flux f_i et de l'envoyer avec sa copie vers leurs destinataires respectifs. Ainsi, nous utilisons un opérateur de duplication à chaque fois qu'un flux doit être envoyé vers plusieurs CM ou vers plusieurs opérateurs.

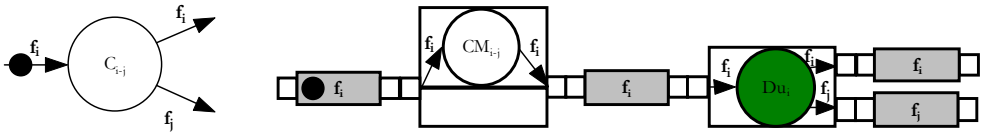


Figure 100 Opérateur de Duplication

Les arcs conditionnels des graphes de transition permettent de modéliser plusieurs configurations. Ces différentes configurations sont également représentées sur les graphes d'implantation. Elles permettent de décrire différents niveaux de QdS.

Règle 10 *Les Arcs Conditionnels*

Les différentes configurations que représentent les arcs conditionnels sont modélisées sur les graphes d'implantation à l'aide de conditions notées Cond$_i$. Ces conditions permettent de déterminer quelle configuration sera choisie et sont exclusives. La Figure 101 décrit un exemple de ce type de représentation sur les graphes d'implantation. On y trouve, en sortie du composant CM$_{i-j}$ trois configurations possibles. Ce type de modélisation ne signifie évidemment pas que ce CM possède trois sorties mais seulement que, selon les conditions associées, on choisira l'une des configurations de sortie. De même, le composant CM$_{i+2-j}$ ne possède pas trois entrées mais seulement une qui sera définie par la configuration choisie. Le mot Fin Condition permet d'indiquer la fin d'un choix de configurations.

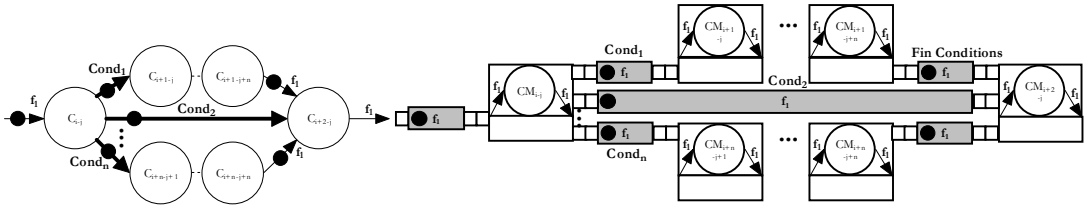


Figure 101 Modélisation des Arcs Conditionnels

Enfin, la dernière information apportée par les graphes d'implantation est la distribution des entités sur les différents sites d'une AMD. Cette information de distribution permet de localiser précisément les entités sur chacun des sites. De plus, on va notifier pour chaque site les dépendances matérielles qui doivent être respectées pour un fonctionnement correct de l'application. Elles permettent de traduire les rôles matériels nécessaires à l'exécution de l'AMD.

Règle 11 *Localisation des entités*

Les graphes d'implantation précisent la notion de distribution. On y trouve les différents sites de l'application en spécifiant les entités qui vont s'exécuter sur chacun d'entre eux. Grâce à cette information, on va pouvoir identifier les conduits distribués à utiliser pour répartir l'AMD en fonction des points où les flux seront transmis à travers le réseau. Cette information n'était pas explicite sur les graphes de transition, elle le devient sur les graphes d'implantation. La Figure 102 donne un exemple de répartition. Le réseau est décrit à l'aide d'une ligne verticale. Ainsi, on peut identifier de chaque côté de cette ligne les sites de l'AMD. Le conduit entre les deux PE traversé par ce trait sera un conduit distribué lors de l'implémentation dont l'objectif sera de transmettre le flux f$_1$ du site A vers le site B.

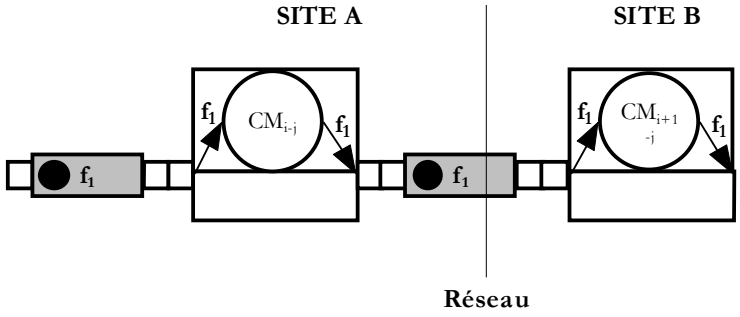


Figure 102 Localisation des entités

Avant de refermer ce chapitre, nous donnons dans la section suivante des exemples de transformation ainsi que les graphes d'implantation correspondants.

3 Exemples de Graphe d'Implantation

L'exemple que nous présentons est celui que nous avons utilisé tout au long de ce mémoire. Il est présenté plus en détail dans le chapitre 3. La Figure 104 donne le graphe d'implantation obtenu après un choix de localisation des différents composants par application des règles de transformation que nous avons décrites précédemment. Nous rappelons sur la Figure 103 le graphe de transition correspondant qui a permis d'obtenir le graphe d'implantation.

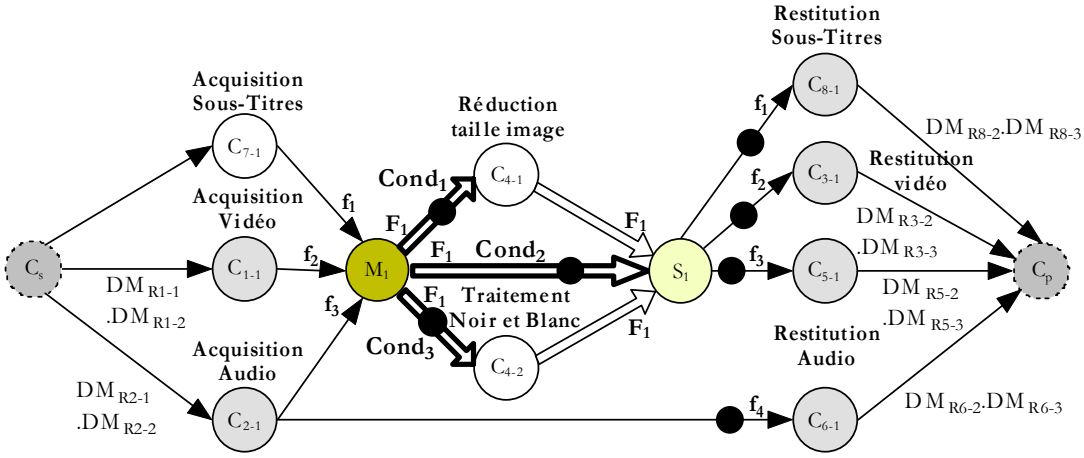


Figure 103 Graphe de Transition de l'Application de Formation à Distance

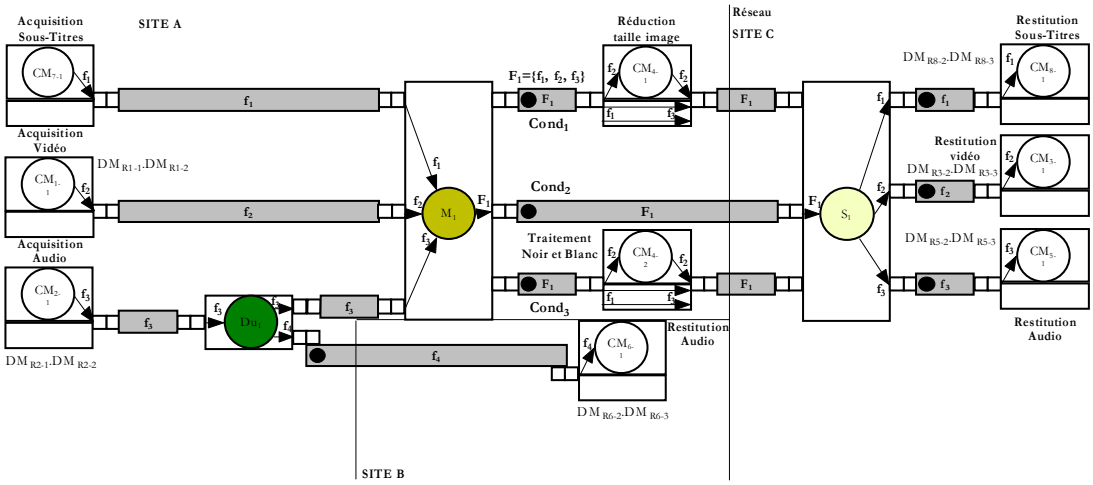


Figure 104 Graphe d'Implantation de l'Application de Formation à Distance

Le site A correspond au site du locuteur. Il se dote des composants d'acquisition de flux nécessaires afin de disposer des flux audio, vidéo et des sous-titres. Ces flux sont évidemment liés par des relations de synchronisation inter-flux. C'est pourquoi nous utilisons, sur ce site, un opérateur de fusion M_1. Avant de transmettre ces flux vers leur destination plusieurs configurations sont possibles afin d'éventuellement pouvoir traiter les flux si la liaison réseau est insuffisante. Ceci est spécifié en sortie de l'opérateur de fusion. Ainsi, nous pourrons soit transmettre le flux synchrone en l'état ($Cond_2$), soit réduire la taille de l'image ($Cond_1$), soit convertir la vidéo en noir et blanc ($Cond_3$).

Les sites B et C sont respectivement ceux des deux téléspectateurs. Le site C reçoit un flux composé de l'audio de la vidéo et des sous-titres. Le site B reçoit uniquement l'audio car le périphérique qu'il utilise possède des capacités restreintes. Comme le flux audio doit être transmis sur le site B et sur le site C, nous utilisons un opérateur de duplication pour réaliser cette tâche. Sur le site B, un opérateur de séparation permet de diviser le flux composé et d'envoyer chaque flux primitif qu'il contient vers le composant de restitution adéquat.

Sur chaque site de l'AMD, nous avons indiqué les dépendances matérielles telles qu'elles ont été introduites par les graphes de transition.

Cette vue permet d'identifier les points de transmission de données et donc les conduits distribués qu'il faudra utiliser lors du déploiement de l'AMD.

Avant de conclure, nous décrivons également un exemple de conjonction et de disjonction dans un graphe de transition. Cet exemple traite d'un flux composé d'un flux audio et d'un flux vidéo. Ces deux flux primitifs doivent être compressés avant d'être transmis sur le réseau. La partie du graphe de transition qui correspond est donnée sur la Figure 105. Comme nous pouvons le voir, nous utilisons les opérateurs

de disjonction et de conjonction afin de réaliser ces traitements en parallèle. Le flux composé F_1 est éclaté en deux flux primitifs f_1 et f_2. Chacun des flux est ensuite traité par les composants adéquats. Puis l'opérateur de conjonction permet de reconstituer le flux composé F_1 initial. L'implantation de ce graphe de transition à l'aide du modèle Osagaia est donnée par le graphe de la Figure 106.

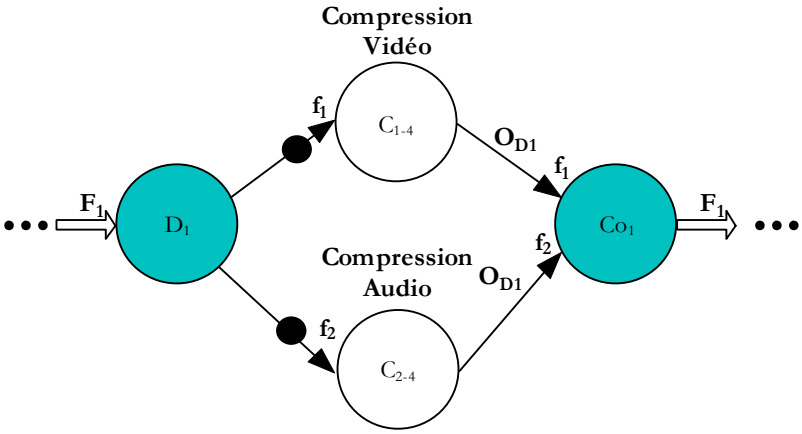


Figure 105 Exemple de Conjonction et de Disjonction

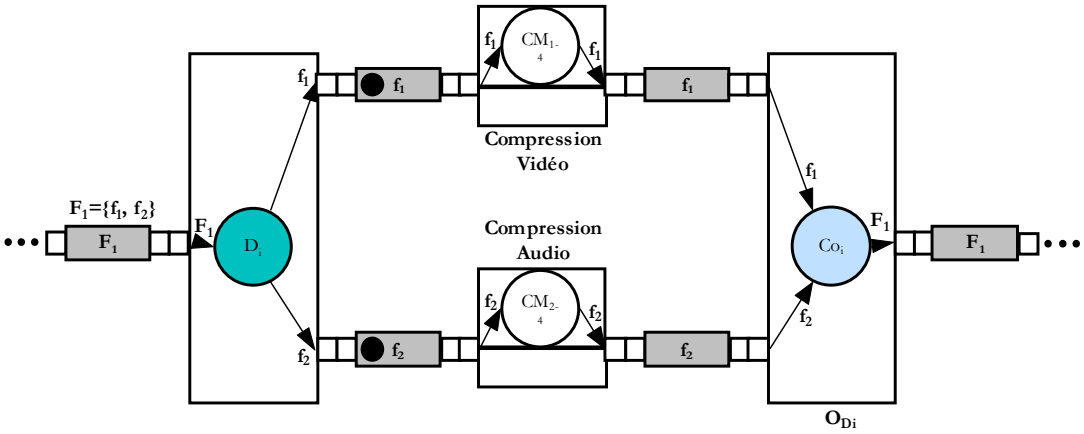


Figure 106 Graphe d'implantation avec Conjonction et Disjonction

4 Synthèse

Nous avons décrit dans ce chapitre une nouvelle transformation dont le but est d'aboutir à une vue des AMD directement implémentable. Dans la précédente partie, nous avons défini des unités d'implémentation afin d'implémenter ces dernières. Ces unités font partie intégrante du modèle Osagaia. La transformation introduite ici utilise donc ces unités afin de donner cette vue des AMD. Les graphes d'implantation obtenus à l'issue de cette transformation sont définis comme une interconnexion de ces unités. L'intérêt de ces graphes réside dans le fait qu'ils fournissent une topologie

des AMD qui pourra être réalisée en connectant les entités du modèle Osagaia. La méthode que nous avons introduite dans la partie précédente nous a permis de transformer les spécifications fonctionnelles et d'obtenir une implémentation fidèle à ces dernières.

Cette transformation aborde une caractéristique importante des AMD que nous avions laissée de côté jusqu'à maintenant. Elle concerne la distribution de ces applications à travers différents sites. En effet, nous concentrons notre attention sur les applications multimédias constituées d'un ensemble de périphériques reliés à l'aide du réseau Internet. Les graphes d'implantation permettent donc de décrire le déploiement des AMD sur ces périphériques en décrivant les entités à implémenter sur chacun d'eux. Ainsi, nous concrétisons les points de transmission des flux synchrones à travers les différents périphériques qui constituent une AMD. Nous avions, bien entendu, anticipé ceci en introduisant dans le modèle Osagaia une entité distribuée permettant de procéder à de telles transmissions. A l'aide des graphes d'implantation, nous connaissons désormais les endroits de l'AMD ou les conduits distribués devront être utilisés.

Cette transformation nous a également permis d'introduire un opérateur de flux synchrones supplémentaire. Nous avons vu sur les graphes de transition que certains flux synchrones sont susceptibles d'être envoyés vers plusieurs CM/PE ou opérateurs. Il est donc nécessaire de fournir un moyen de réaliser cette possibilité. Ce moyen fait partie de l'implémentation non-fonctionnelle des AMD et fait l'objet d'un opérateur de duplication. Cet opérateur permet de dupliquer le flux synchrone qu'il reçoit en entrée en n flux synchrones qui sont les copies exactes de ce dernier. Ainsi, les copies produites peuvent être envoyées vers leurs destinations respectives.

Maintenant que nous disposons de tous les éléments nécessaires à l'implémentation des AMD telles qu'elles ont été spécifiées, il nous reste à proposer une implémentation de ces éléments. Cette dernière étape va permettre de concrétiser la représentation donnée à l'aide des graphes d'implantation. Concrètement, nous pourrons proposer des implémentations des AMD en connectant les éléments tels qu'ils le sont sur les graphes d'implantation. Les modèles introduits sont implémentés à l'aide d'un langage cible. Nous avons choisi à ce titre le langage de programmation Java. Nous proposons donc dans le prochain chapitre une implémentation du modèle Osagaia en Java. De plus, nous explicitons les motivations de ce choix et les inconvénients rencontrés lors d'une telle implémentation.

Chapitre 8 – Implémentation du Modèle de Composants Osagaia

« La théorie, c'est quand on sait tout et que rien ne fonctionne. La pratique, c'est quand tout fonctionne et que personne ne sait pourquoi. »

Albert Einstein, 1879-1955

La dernière étape de ce travail consiste à fournir une implémentation du modèle Osagaia proposé précédemment (cf. chapitre 6). Ainsi, nous allons concrétiser l'architecture décrite par les graphes d'implantation (cf. chapitre 7) et donc pouvoir implémenter les AMD conformément aux spécifications fonctionnelles décrites par la méthode proposée.

Les modèles que nous avons proposés nous ont permis de nous abstraire de la complexité induite par l'implémentation des AMD telles que nous les concevons. Il est temps maintenant de fournir un modèle d'exécution du modèle Osagaia conforme à ses spécifications. Pour ce faire, nous devons choisir un langage de programmation cible. Notre intérêt pour les environnements distribués fortement hétérogènes nous a naturellement conduits à choisir le langage de programmation orienté objet Java.

L'objectif de cette partie est d'évaluer le modèle de composants Osagaia en testant le déploiement des AMD. D'une part, nous allons pouvoir tester l'implémentation des propriétés fonctionnelles à l'aide des CM, d'autre part nous allons tester les propriétés non-fonctionnelles à travers l'utilisation de PE et d'opérateurs de flux synchrones. Par la même occasion, nous pourrons valider le modèle de flux de données Korrontea et ses politiques de synchronisation.

1 Introduction

Tandis que les graphes d'implantation donnent un aperçu de l'architecture d'une AMD, le modèle de composants logiciels Osagaia apporte les éléments à assembler afin de déployer ce type d'architecture.

Ainsi, l'implémentation du modèle Osagaia a pour objectif de valider son utilisation dans le déploiement d'AMD reconfigurables dynamiquement. Nous allons donc devoir mettre tout ceci en œuvre en proposant une implémentation où les composants seront supervisables par la plate-forme d'exécution présentée dans nos travaux précé-

dents [LAP06]. Cet aspect non-fonctionnel est laissé à la charge du PE dont le rôle est d'encapsuler les CM. Dans un premier temps, nous allons définir et mettre en œuvre la connexion et la déconnexion des PE d'une AMD. Ainsi, nous pourrons tester le développement des CM et leur utilisation à l'intérieur d'un PE. Nous validerons de la sorte le comportement global d'une AMD c'est-à-dire en tant qu'interconnexion de PE, d'opérateurs et de conduits.

Le second objectif concerne la gestion des flux synchrones. Ils sont définis par le modèle Korrontea sur lequel est basé le modèle Osagaia. Nous allons tester la manipulation des flux dans les AMD ainsi que les politiques de synchronisation de ces flux afin de s'assurer du respect de cette propriété.

Nous décrivons dans ce chapitre l'implémentation du modèle de composants Osagaia précédemment spécifié. Notre première tâche va être de décrire sommairement la méthode de développement que nous avons employée.

2 Méthode de Développement

Pour le développement du modèle de composants Osagaia, nous avons voulu pouvoir tester chacune des propriétés à implémenter de manière incrémentale. Nous avons trouvé intéressant de pouvoir valider chaque implémentation de propriété avant de passer à l'implémentation de la suivante et ainsi de suite. En conséquence, nous avons choisi un cycle de vie itératif [LAR05]. L'avantage est de disposer à chaque itération d'une implémentation partielle et testable des entités du modèle.

La méthode retenue est décrite sur la Figure 107. Le développement est découpé en itérations successives dans le temps. Chaque itération concerne l'implémentation d'une propriété du modèle Osagaia. Ce cycle de vie débute par une implémentation minimale des différentes entités du modèle. Chaque itération va permettre de les doter de propriétés supplémentaires. Nous commencerons par implémenter les propriétés fonctionnelles en développant des exemples de fonctionnalités atomiques d'AMD. Puis nous nous intéressons à l'implémentation des propriétés non-fonctionnelles. Ainsi, petit à petit nous définirons le CM, le PE, le conduit et les opérateurs. Chaque propriété ainsi implémentée est intégrée à l'entité du modèle qui la requiert. Cette entité est ensuite testée afin d'en valider le comportement. Ces tests consistent à implémenter des architectures simples dont l'objectif est de tester la propriété nouvellement définie dans tous les cas de configurations possibles.

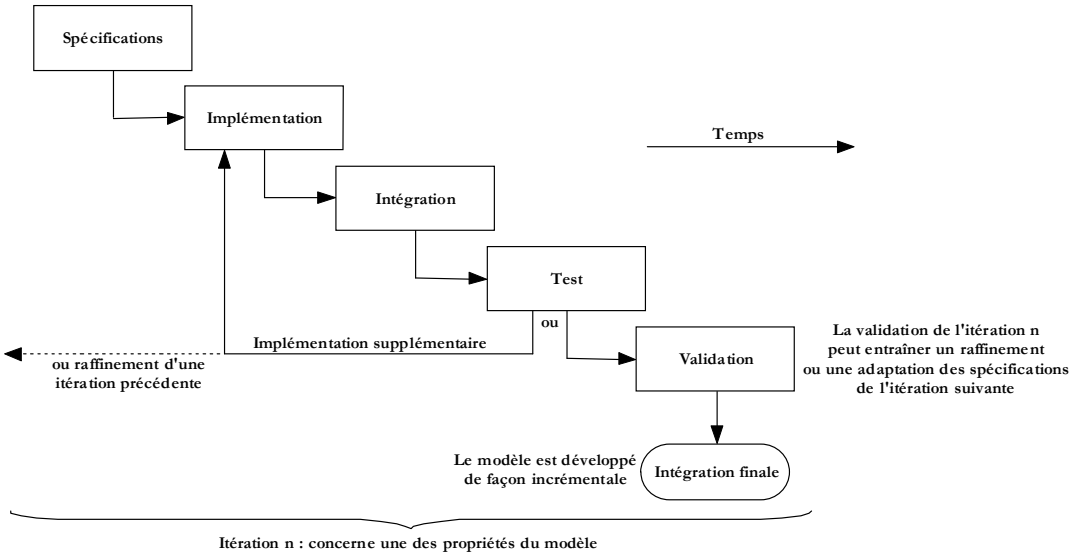


Figure 107 Cycle de développement utilisé

L'objectif est d'obtenir une implémentation complète du modèle et de ces principales propriétés :

- l'implémentation des CM d'une application ;
- la propagation des flux synchrones entre les différentes entités du modèle ;
- la transmission des flux entre les différents sites d'une application ;
- la conservation de la synchronisation intra-flux ;
- la conservation de la synchronisation inter-flux ;
- la connexion et la déconnexion des PE dans une architecture (supervision) ;
- le fonctionnement du PE en fonction des flux qu'il reçoit et qu'il produit ;
- le fonctionnement du conduit ;
- le fonctionnement des différents opérateurs ;
- etc.

3 Langage de programmation cible

Afin de réaliser cette implémentation, nous avons dû choisir un langage de programmation cible. Notre but étant d'implémenter des composants, nous nous sommes

tournés vers un langage de programmation orientée objet. Ainsi, les entités du modèle Osagaia (PE, CM, conduit, opérateurs) sont définies comme une interconnexion d'objets. Dans des travaux futurs, nous envisageons d'utiliser également des plates-formes ou des environnements de déploiement de composants logiciels.

Un critère déterminant dans le choix du langage est la possibilité de déployer les AMD sur des environnements hétérogènes. En effet, il faut que l'implémentation proposée puisse être déployée sur des environnements différents tant au niveau système qu'au niveau matériel. Nous nous sommes donc tournés vers le langage Java qui possède cette caractéristique de portabilité [ECK00] grâce à l'utilisation d'une machine virtuelle[71] qui permet de faire l'interface entre le système d'exploitation et les programmes implémentés.

D'autres caractéristiques de ce langage ont conforté ce choix. Depuis quelques années, Sun Microsystems développe la plate-forme J2ME [GIG00] qui permet d'utiliser Java sur des terminaux mobiles tels que les assistants personnels ou encore les téléphones portables. Nous prévoyons de pouvoir déployer des AMD sur ce type de périphérique et donc de pouvoir y utiliser notre plate-forme. Nous avons entamé quelques travaux dans ce sens [LOU07]. Un autre avantage de ce langage est qu'il fournit un ensemble d'API (Application Programming Interface) orienté multimédia intéressant pour nos travaux. Une API est une bibliothèque de classes et d'interfaces destinées aux développeurs d'applications afin de faciliter leurs tâches en utilisant du code déjà écrit. En général, les API s'intéressent à un domaine particulier. Ainsi, les API JAI [SUM99], Javasound [SUN00] et JMF [SUN99] permettent respectivement d'ajouter des fonctionnalités de traitement d'images, de support de l'audio et de la vidéo dans des applications Java.

Bien que nous ayons choisi le langage Java pour cette première implémentation du modèle Osagaia, nous pensons, dans nos travaux futurs, fournir plusieurs implémentations du modèle à l'aide d'autres langages ou de plates-formes à composants. Le prochain paragraphe détaille cette implémentation.

4 Développement du Modèle Osagaia

Ce modèle est développé indépendamment de la plate-forme d'exécution. Nous implémentons les points d'interactions entre les entités d'une AMD et la plate-forme tels qu'ils ont été précédemment spécifiés. Dans tous les tests que nous pratiquons,

[71] Java Virtual Machine (JVM).

nous simulons la plate-forme d'exécution à l'aide d'interfaces graphiques permettant de provoquer des reconfigurations des AMD et de récupérer les états de fonctionnement des entités qui les composent. Nos prochains travaux visent également à connecter la plate-forme développée dans [LAP06] avec des AMD développées à l'aide du modèle Osagaia afin d'évaluer le fonctionnement correct de l'ensemble.

Nous commençons par détailler le fonctionnement global des entités qui composent le modèle. Dans une seconde partie, nous présenterons le prototype d'AMD développé afin de tester des propriétés telles que la reconfiguration, la synchronisation inter-flux, etc.

4.1 Les Entités du Modèle

Les fonctionnalités atomiques des AMD sont implémentées à l'aide des CM.

4.1.1 Le Composant Métier

Le CM est décrit à l'aide d'une classe abstraite qui permet d'implémenter ses différentes propriétés. Cette classe implémente l'interface ControleComposant afin que le cycle de vie du CM puisse être contrôlé par le PE qui va le contenir. Les opérations de cette interface sont partiellement implémentées dans la classe abstraite ComposantMetier. Cette interface peut être étendue afin par exemple de rajouter des comportements particuliers pouvant être induits par certaines implémentations.

Le CM doit être implémenté comme un thread java. Un thread est une unité d'exécution java qui donne l'illusion au programmeur de s'exécuter par lui-même avec sa propre unité de traitement (CPU) [ECK00]. Pour ce faire, la classe abstraite hérite de la classe Thread.

Ainsi, pour réaliser un CM, il faut créer une classe qui étend la classe ComposantMetier. Le développeur doit dans un premier temps compléter les implémentations des opérations de l'interface ControleComposant si besoin est. Ensuite, la méthode run() propre à un thread doit être implémentée sous la forme d'une boucle infinie réalisant le traitement. La Figure 108 donne un exemple d'implémentation de la méthode run() d'un CM qui permet de générer le négatif des images lues en entrée. La boucle infinie est exécutée tant que le booléen fonctionnement est vrai. L'opération arret() de l'interface ControleComposant permet de positionner ce booléen à faux. Dans ce CM, le flux traité est un flux vidéo où chaque unité d'information contient une image reçue placée dans une unité temporelle. Les autres informations contenues dans l'unité temporelle ne peuvent pas être modifiées par le CM mais elles peuvent, dans certains cas, être utilisées pour réaliser certaines tâches comme par exemple la

restitution d'un flux synchrone en respectant la synchronisation intra-flux. Nous retrouvons dans ce code les étapes identifiées lorsque nous avons défini le CM (cf. Figure 17 du chapitre 6). En effet, un CM débute par une opération de lecture de données si tant est qu'il doive lire des données pour fonctionner[72]. La donnée lue (l'image dans l'exemple) est ensuite traitée et la donnée produite (ici une image) est de nouveau associée à l'unité temporelle qui est, pour terminer, écrite dans l'unité de sortie du PE.

```
public void run()
{
        while(fonctionnement)
        {
            ut=pe.ue.tentativeLecture(fluxTraites[0]); // Lecture d'une unité temporelle sur le flux identifié f1

            if(ut==null)
                this.yield();
            else
            {
                img=(Image)ut.echantillons();
                x=img.getWidth(fenetre);
                y=img.getHeight(fenetre);
                tableauDePixels=new int[x*y];
                pg=new PixelGrabber(img, 0, 0, x, y, tableauDePixels, 0, x);
                try
                {
                    pg.grabPixels();
                }
                catch(InterruptedException ie)
                {
                    System.err.println("Interruption pendant le traitement d'une image " + ie);
                }
                for(int i=0;i<y*x;i++)
                {
                    couleur=new Color(tableauDePixels[i]);
                    tableauDePixels[i]=(255<<24) | (255-couleur.getRed()<<16) | (255-couleur.getGreen()<<8) | (255-couleur.getBlue());
                }
                img=fenetre.createImage(new MemoryImageSource(x, y, tableauDePixels, 0, x));
                ut.modifierEchantillons(img);

                pe.us.ecrireDonneeCM(fluxProduits[0], ut, contrainteFluxProduit[0]); // Ecriture dans l'unité de sortie avec l'image modifiée
            }
        }
}
```

Traitement de l'image

Figure 108 Exemple d'implémentation de la méthode run() d'un CM

Sur le code de la Figure 108, les opérations tentativeLecture et ecrireDonneeCM prennent en paramètre l'identifiant du flux. La classe ComposantMetier contient des attributs initialisés lors de la construction d'un CM qui représentent les identifiants des flux traités et des flux produits par le CM. Il en est de même pour le dernier paramètre de la méthode ecrireDonneeCM qui définit la contrainte temporelle du flux produit. Ces attributs sont désignés dans l'exemple de la Figure 108 par fluxTraites[0], fluxProduits[0] et contrainteFluxProduit[0]. Ces informations sont contenues dans un

[72] Les CM qui créent des flux de données n'effectuent pas forcément de lecture de données pour exécuter cette production.

fichier XML qui doit être complété par le développeur. Chaque CM est donc accompagné d'un fichier XML qui définit un ensemble d'informations nécessaires à son fonctionnement. Elles sont récupérées par le CM lors de sa construction afin, par exemple, de connaître les identifiants des flux traités et des flux produits. Ce fichier est également utilisé par le PE pour récupérer des informations utiles à son exécution. Un exemple d'un tel fichier est donné par la Figure 109. Il représente une partie du fichier XML qui correspond au CM de l'exemple donné par la Figure 108. Ainsi, pour chaque CM, sont définis le nombre de flux traités et le nombre de flux produits par les balises <nbFlux>. Pour chaque flux traité, on décrit son identifiant, sa contrainte temporelle, son type et on indique s'il s'agit d'un flux prépondérant. Pour les flux produits, on se limite à donner son identifiant, sa contrainte temporelle et son type. Il est intéressant de remarquer que nous ne tenons pas compte ici de l'éventuelle appartenance des flux traités et produits à des flux composés. En effet, cette gestion est laissée à la charge du PE.

```
...
<fluxManipules>
    <fluxTraites>
        <nbFlux>1</nbFlux>
        <flux>
            <identifiant>f1</identifiant>
            <contrainte>forte</contrainte>
            <type>video</type>
            <preponderant>oui</preponderant>
        </flux>
    </fluxTraites>
    <fluxProduits>
        <nbFlux>1</nbFlux>
        <flux>
            <identifiant>f1</identifiant>
            <contrainte>forte</contrainte>
            <type>video</type>
        </flux>
    </fluxProduits>
</fluxManipules>
...
```

Figure 109 Extrait du fichier XML d'un CM de traitement négatif

Chaque CM ainsi implémenté est exécuté à l'intérieur d'un PE et son exécution est démarrée par son intermédiaire. Nous décrivons dans le paragraphe suivant l'implémentation du PE.

4.1.2 Le Processeur Elémentaire

Conformément aux spécifications définies au chapitre 6, le PE est composé de trois unités : l'unité d'entrée[73], de sortie et de contrôle. Nous avons choisi la composition pour implémenter le PE car il peut être constitué de plusieurs façons. En effet, on peut définir[74] des PE sans entrées, sans sorties et des PE avec des entrées et des sorties. Ainsi, suivant le type de PE, on y ajoutera seulement une UE, seulement une US, soit ces deux unités.

L'UE est composée principalement d'une zone de stockage temporaire des flux et d'un buffer d'événements. La zone de stockage se divise en deux parties, une pour les flux à traiter et l'autre pour ceux en transit. La tâche de l'unité d'entrée consiste à exécuter de manière continue plusieurs tâches. La première de ces tâches est de traiter les événements provenant des conduits connectés en entrée du PE. Le buffer où sont stockés ces événements est une file d'attente. L'UE traite un à un les événements de cette file. Un événement notifie à l'UE qu'elle peut aller récupérer une tranche synchrone dans le port de sortie du conduit qui a levé l'événement. Cette tranche est alors stockée dans le port d'entrée de l'UE auquel ce conduit est connecté. Une fois la tranche récupérée, l'UE émet un événement de notification au conduit concerné afin de lui spécifier que son port de sortie est vide. Ensuite, l'UE se charge de séparer les tranches synchrones stockées dans ses ports d'entrée en flux placés dans les zones prévues à cet effet. Ainsi, les flux destinés à être traités sont stockés sous la forme d'une succession d'unités temporelles tandis que les autres sont stockés sous la forme d'une succession d'unités d'information. La distinction entre ces flux se fait à l'aide du fichier XML associé au CM encapsulé dans le PE (cf. Figure 109). Le CM peut alors directement lire les unités temporelles dans ces zones de stockage. Parallèlement, les unités d'information des flux non traités sont écrites dans l'US. L'UE implémente l'interface LectureFlux qui va offrir au CM les méthodes de lecture des unités temporelles (cf. Figure 108).

L'US comporte également une zone de stockage pour placer temporairement les flux produits par le CM et ceux en transit dans le PE. Ils sont stockés dans cette zone sous la forme de flux primitifs. L'UE implémente l'interface EcritureFlux offrant les opérations qui permettent de former des tranches synchrones. Les attributs des tran-

---

73 L'unité d'entrée et l'unité de sortie font parties de l'unité d'échange chargée de la gestion des connexions d'entrée et de sortie du PE.

74 Cette définition dépend du CM que le PE va exécuter. Pour les CM d'acquisition ou de création de flux, le PE possèdera seulement une unité de sortie. Pour les CM de restitution ou de stockage de flux, le PE possèdera une unité d'entrée. Enfin, pour les CM de traitement de flux, le PE possèdera une unité d'entrée et une unité de sortie.

ches et des unités d'information sont définis différemment selon que le PE possède un flux prépondérant en entrée ou pas. Cette information est également connue à l'aide du fichier XML du CM (cf. Figure 109). La tâche de l'US consiste à appliquer les politiques de synchronisation sur les flux stockés dans la zone de stockage afin de constituer les tranches synchrones du flux composé à écrire en sortie. Une fois ces tranches constituées, elles sont écrites une à une dans le port de sortie du PE. Chaque écriture lève un événement à destination du conduit connecté en sortie.

Les ports d'entrée et de sortie sont des éléments de l'UE et de l'US. Ces unités peuvent gérer ces ports à l'aide de l'interface AccesPort qu'ils implémentent. Cette gestion est réalisée grâce aux événements que s'échangent les PE et les conduits.

L'UC permet la supervision du PE par la plate-forme d'exécution. Pour ce faire, elle implémente l'interface ControlePE. Les méthodes de cette interface permettent de gérer les connexions/déconnexions du PE mais aussi le cycle de vie du CM qu'il contient. L'UC se charge également d'envoyer les états de fonctionnement du PE à la plate-forme d'exécution sous la forme d'événements (cf. chapitre 6). Nous décrivons sur la Figure 110 un exemple de configuration d'une application simple composée de trois CM encapsulés dans trois PE. Le premier permet de produire un flux vidéo capturé à l'aide d'une WebCam. Le deuxième traite ce flux vidéo. Il s'agit d'un traitement de détection de contours qui produit en sortie un flux vidéo où seuls les contours sont représentés. La Figure 111 montre une capture d'écran de la restitution du flux traité par ce CM. Le dernier CM assure un rôle de restitution vidéo.

Lors de la mise en place de cette application, les CM sont déclarés en premier lieu et les conduits sont déclarés ensuite. Chaque conduit de sortie est déclaré comme un conduit simple. Des tableaux de conduits seront utilisés lorsqu'il faudra proposer plusieurs conduits en entrée des PE. Dans notre exemple, ces tableaux ne contiennent que le conduit de sortie du PE suivant car les PE ne produisent et ne traitent qu'un seul flux synchrone. Les PE sont constitués en fonction des CM qu'ils encapsulent. Ainsi, le CM d'acquisition vidéo sera exécuté dans un PE sans entrée (paramètre null du constructeur sur la Figure 110). Le CM d'extraction de contours est exécuté dans un PE doté d'un conduit d'entrée et d'un conduit de sortie. Enfin, le CM de restitution vidéo ne possèdera pas de sortie (paramètre null du constructeur sur la Figure 110). L'utilisation de la composition nous permet de définir plusieurs configurations pour les PE. Une fois ces déclarations faites, il est nécessaire de connecter chaque conduit aux PE qu'ils relient. Le processus de gestion est alors démarré dans chaque conduit. A l'aide des opérations de l'interface ControlePE, chaque PE et donc chaque CM peut être initialisé et mis en marche. Il faut noter que la méthode init() ne se trouve pas dans la spécification du chapitre 6. Elle est cependant nécessaire pour les

initialisations du CM. Lorsque l'application est arrêtée, chaque CM doit être arrêté à l'aide de la méthode arreterCM(). Les PE et les conduits auront été préalablement déconnectés. La Figure 111 montre la restitution du flux vidéo capturé par la WebCam et traité. Une interface dédiée permet de régler les paramètres du traitement réalisé.

```
                    •••
// Déclaration et construction des CM
AcquisitionVideo av=new AcquisitionVideo();
TraitementContours tc=new TraitementContours();
RestitutionVideo rv=new RestitutionVideo();

// Déclaration et construction des conduits
Conduit cond=new Conduit(1);
Conduit conduitSortie=new Conduit(1);
Conduit[] condEntree=new Conduit[1];
condEntree[0]=cond;
Conduit[] condEntree2=new Conduit[1];
condEntree2[0]=conduitSortie;

// Déclaration et construction des Processeurs Elémentaires
ProcesseurElementaire pe=new ProcesseurElementaire(null, av, cond);
ProcesseurElementaire pe2=new ProcesseurElementaire(condEntree, tc, conduitSortie);
ProcesseurElementaire pe3=new ProcesseurElementaire(condEntree2, rv, null);

// Connexion des conduits et des Processeurs Elémentaires
cond.connexion(pe, pe2);cond.demarrerProcessus();
conduitSortie.connexion(pe2, pe3);conduitSortie.demarrerProcessus();

// Mise en marche des CM/PE
pe3.init();
pe3.demarrerCM();
pe2.init();
pe2.demarrerCM();
pe.init();
pe.demarrerCM();
                    •••
// Arrêt des CM et déconnexion des conduits
pe3.deconnexion();
pe3.arreterCM();
pe2.deconnexion();
pe2.arreterCM();
pe.deconnexion();
pe.arreterCM();
cond.arreterProcessus();
conduitSortie.arreterProcessus();
                    •••
```

Figure 110 Exemple de Configuration

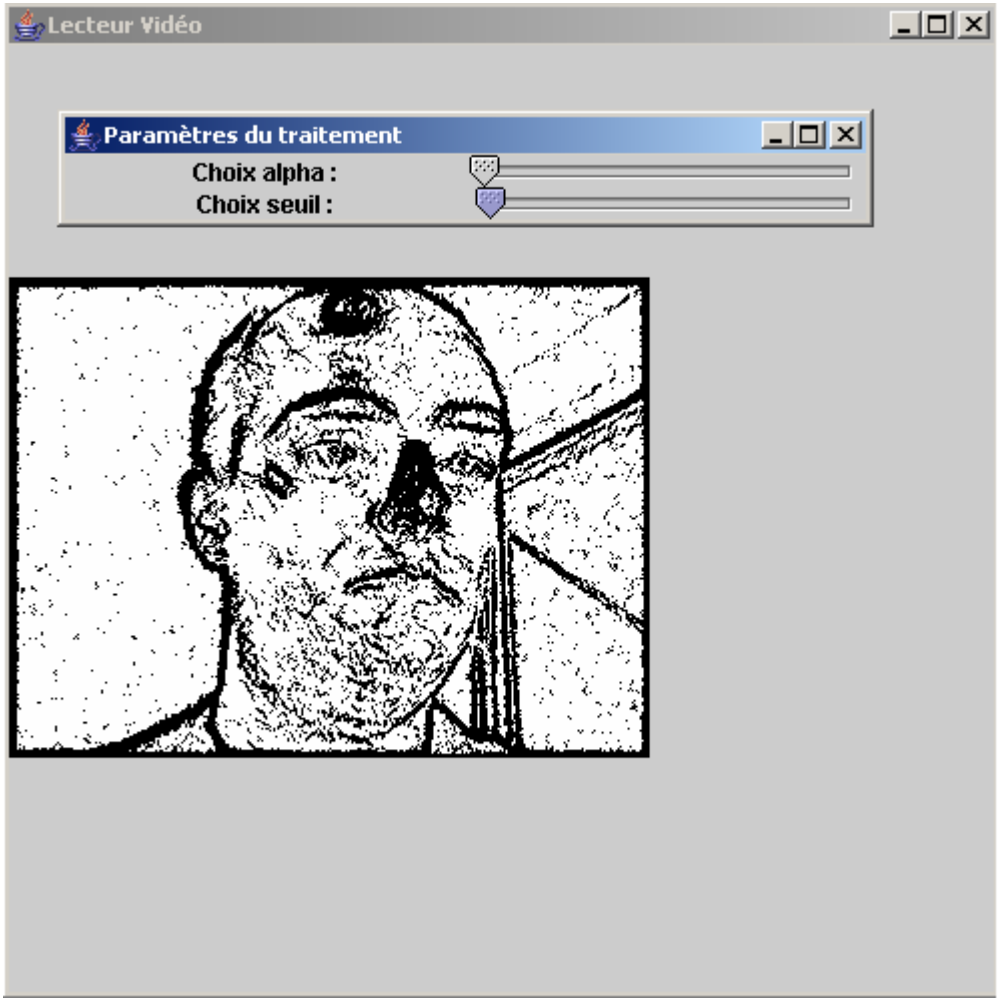


Figure 111 Restitution du Flux Vidéo traité

Nous allons maintenant présenter rapidement l'implémentation du conduit.

4.1.3 Le Conduit

Le conduit est également composé de trois unités : l'unité d'entrée, de sortie et de contrôle. Nous avons également choisi la composition pour implémenter le conduit car son architecture interne varie selon les cas d'utilisation. Ainsi, un conduit local sera utilisé pour relier deux PE se trouvant sur la même machine, il possède alors une unité d'entrée, une unité de sortie et une unité de contrôle. Un conduit distribué, quant à lui, sera séparé en deux parties. Le conduit client comporte uniquement une unité d'entrée et une unité de contrôle tandis que le conduit serveur contient une unité de sortie et une unité de contrôle. Chaque partie du conduit se trouve sur l'une des machines concernées par la transmission d'un flux synchrone. Un conduit est conçu pour transporter un seul flux synchrone qu'il soit primitif ou composé.

L'UE du conduit possède un buffer d'entrée (file d'attente) dont le but est de stocker temporairement les tranches synchrones du flux transporté. L'UE reçoit un événement du PE connecté en entrée lorsqu'une tranche est disponible sur son port de sortie. A la réception de cet événement, l'UE récupère cette tranche dans son port d'entrée et la stocke dans le buffer d'entrée.

L'US du conduit possède un buffer de sortie (file d'attente) dont le but est de stocker temporairement les tranches synchrones du flux transporté. L'US récupère les tranches stockées dans ce buffer afin de les écrire une à une dans son port de sortie. Chaque écriture déclenche la levée d'un événement à destination du PE suivant.

Un processus client/serveur permet de transférer les tranches synchrones du buffer d'entrée vers le buffer de sortie. Dans le cas d'un conduit distribué, ce transfert est réalisé à l'aide du réseau. Pour l'instant, nous avons implémenté un transfert à l'aide de sockets par l'intermédiaire du protocole TCP. La Figure 112 montre l'implémentation de ce mécanisme du côté client. La Figure 113 montre l'implémentation du processus du côté serveur.

```
                    •••
try
{
        s=new Socket(adresseIP, port);
}
catch(IOException ioe)
{
        System.err.println("Echec lors de la création de la socket.");
}
try
{
         OutputStream os=s.getOutputStream();
         oos=new ObjectOutputStream(os);
}
catch(IOException ioe)
{
         System.err.println("Echec lors de la création des flux.");
}
try
{
        // Transfert distribué du Buffer d'entrée vers le buffer de sortie
        for(int i=0;i<bufferEntree.taille();i++)
        {
                while(bufferEntree.taille()!=0)
                {
                     oos.writeObject(bufferEntree.defiler());
                     oos.flush();
                }
        }
}
catch(IOException ioe)
{
          System.err.println("Ecriture sur la socket impossible.");
}
try
{
           s.close();
}
catch(IOException ioe)
{
         System.err.println("Fermeture de la socket impossible.");
}

                    •••
```

Figure 112 Transfert distribué côté client

```
                    •••
if (ecoute != null)
{
    try
    {
       client = ecoute.accept();
    }
    catch (IOException ioe)
    {
       client=null;
       System.err.println("Echec lors de l'écoute des connexions.");
    }
    // On récupère le flux d'entrée sur la socket
    try
    {
       InputStream is = client.getInputStream();
       ois = new ObjectInputStream(is);
    }
    catch (IOException ioe)
    {
       ois = null;
       System.err.println("Impossible de créer le flux.");
    }
    try
    {
       bufferSortie.enfiler( (TrancheSynchrone)ois.readObject());
    }
    catch (Exception e)
    {
       System.err.println("Impossible de lire sur la socket.");
    }
    try
    {
       client.close();
    }
    catch(IOException ioe)
    {
       System.err.println("Impossible de fermer la socket.");
    }
}
                    •••
```

Figure 113 Transfert distribué côté serveur

L'UC permet de contrôler le cycle de vie du conduit. Elle implémente l'interface ControleProcessus qui possède des méthodes pour démarrer et stopper le transfert entre les buffers du conduit. Ainsi, à l'aide de cette interface la plate-forme peut également superviser le conduit. Elle peut également recevoir des états de fonctionnement du conduit.

4.1.4 Les Opérateurs

Les opérateurs sont définis comme une seule entité qui implémente une opération propre. On en distingue cinq qui sont : fusion, séparation, disjonction, conjonction et duplication. Chaque opérateur est doté de ports d'entrée et de sortie. Le nombre de ports varie en fonction de l'opérateur implémenté. Les ports permettent d'assurer le transit des flux dans les opérateurs. Les opérateurs reçoivent et produisent des événements de la part des conduits auxquels ils sont connectés afin d'assurer ce transfert. Les opérateurs sont introduits plus précisément dans les chapitres 6 et 7.

4.2 Présentation d'un Prototype

Nous terminons ce chapitre en présentant un prototype développé pour tester les connexions et déconnexions dynamiques de PE/CM ainsi que la conservation de la synchronisation inter-flux malgré les traitements subis par certains d'eux et les transferts par le réseau.

Ce prototype est distribué, il est composé de deux parties. La première partie est constituée de deux composants d'acquisition vidéo, soit à partir d'une WebCam, soit à partir d'un fichier. Chacun de ces deux CM produit deux flux strictement identiques qui sont intégrés dans un même flux composé transmis vers l'autre partie de l'application à l'aide d'un conduit distribué. La seconde partie est composée d'un ou de plusieurs CM suivant les configurations que l'on veut mettre en œuvre. On y trouve un CM de restitution qui permet d'afficher les deux flux vidéo reçus par le conduit distribué. On peut éventuellement avant cette restitution ajouter un ou plusieurs CM de traitement de l'un de ces deux flux vidéos. Les CM peuvent être dynamiquement ajoutés et enlevés. Cette application propose six CM de traitement : une réduction de la taille de l'image, une transformation de la vidéo en négatif, une détection de la teinte chair, une transformation de la vidéo en niveaux de gris, un flou et une détection de contours.

L'interface graphique permet de simuler le rôle de la plate-forme d'exécution en ajoutant supprimant, démarrant et stoppant des CM de traitement. Elle permet aussi de sélectionner la source des flux vidéo. Cette interface est montrée sur la Figure 114. Sur la première partie (fenêtre de gauche), des zones sont dédiées à la saisie de l'adresse IP du serveur et du port de l'autre machine. Deux boutons permettent de sélectionner la source d'acquisition des flux vidéo. Le bouton ouvrir un fichier permet de mettre en œuvre un CM d'acquisition depuis un fichier. Le bouton connecter permet de mettre en œuvre un CM d'acquisition depuis une WebCam. La seconde partie (fenêtre de droite) possède une zone de texte qui permet de saisir le numéro de port

utilisé pour la communication. Les cases à cocher en partie droite permettent de sélectionner un ou plusieurs CM de traitement placés avant celui de restitution des flux. Cette restitution est réalisée dans la partie basse de l'interface graphique. On y voit la vidéo issue du premier flux à gauche et celle du second flux à droite. Les CM de traitement agissent sur le second flux.

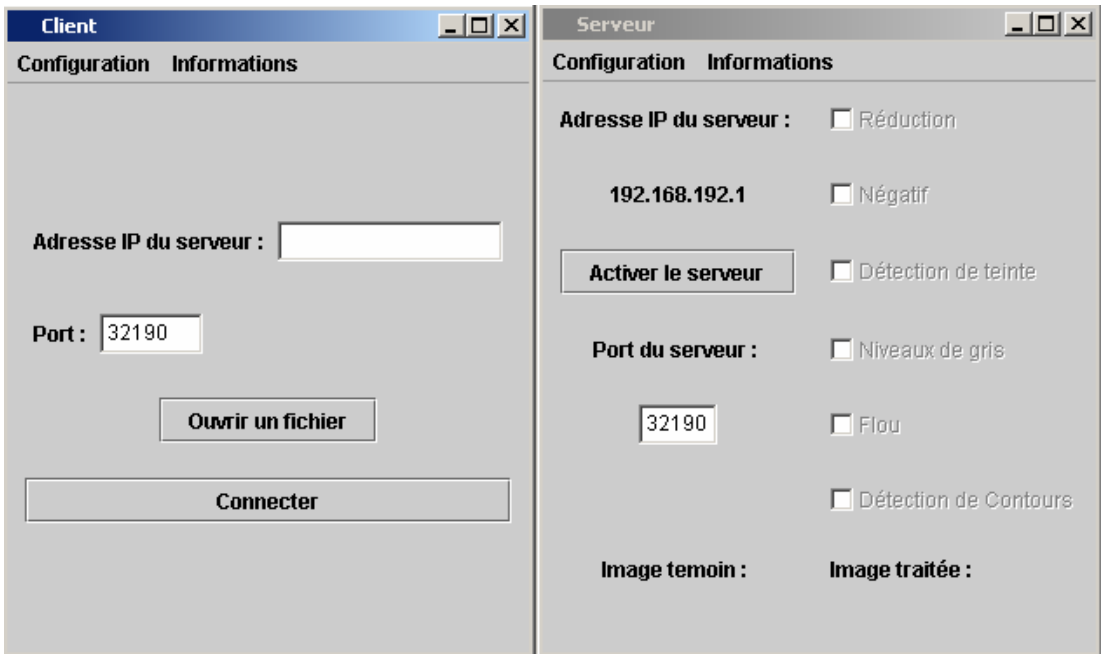


Figure 114 Interface du prototype

Le principe de ce prototype est simple : le CM d'acquisition produit un flux composé de deux flux primitifs strictement identiques et synchrones. Ces deux flux étant à contrainte temporelle forte, la politique de synchronisation forte est appliquée en sortie de ce composant ce qui permet de conserver les relations de synchronisation inter-flux entre ces deux flux vidéo. Le flux composé ainsi constitué est transmis sur le réseau. A l'arrivée il est transmis au CM de restitution après avoir éventuellement traversé un ou plusieurs CM de traitement. Les CM de traitement reçoivent un flux composé dont ils ne traitent que le second flux. La présence du premier flux sert de référence et permet de tester la conservation de la synchronisation malgré la transmission et les traitements effectués. De plus, ce prototype permet de tester l'ajout et le retrait dynamique d'un ou de plusieurs CM et donc la composition et reconfiguration dynamique d'une application.

La Figure 115 montre un exemple de composition de l'application. Nous avons ajouté un CM de détection de contours suivi d'un CM qui applique un flou. Nous pouvons voir sur les vidéos restituées que ces dernières sont synchrones malgré les deux traitements subis par la seconde. La Figure 116 montre, quant à elle, la même configuration à laquelle on a ajouté un CM de réduction de la taille de la vidéo.

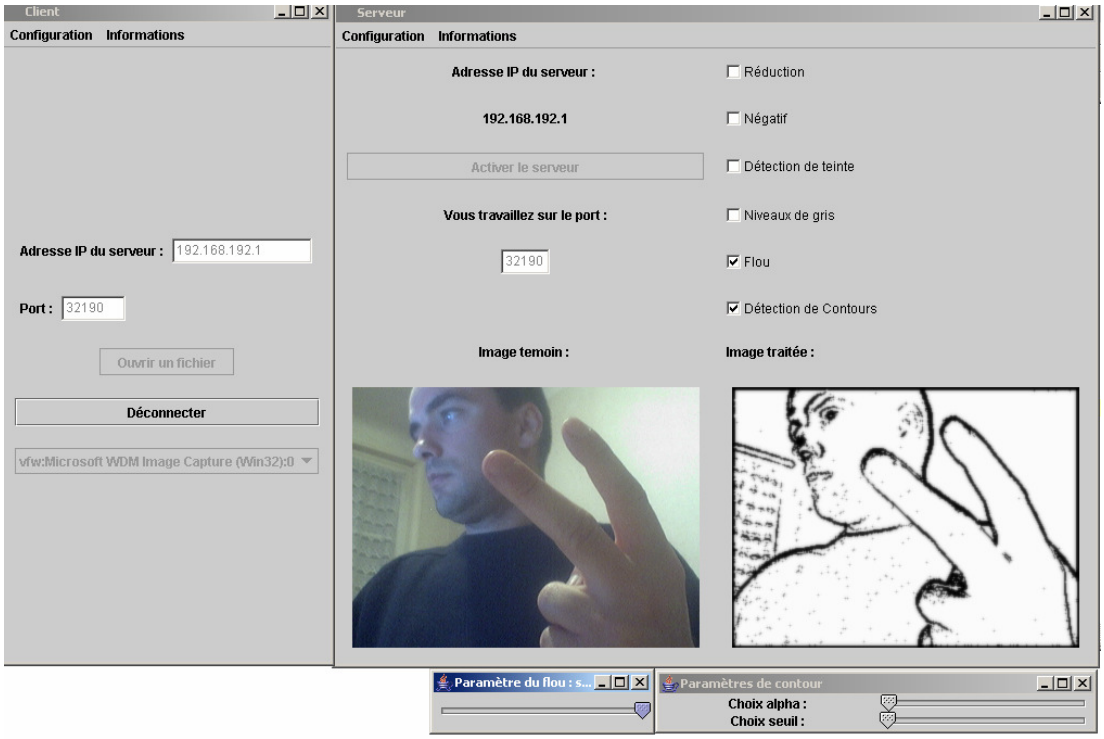


Figure 115 Détection de Contours et Flou appliqués à la seconde vidéo

On observe une nouvelle fois que les deux vidéos sont synchrones malgré les trois traitements subis par le second flux. Ce prototype permet également de retirer de l'application un ou plusieurs CM de traitement. La Figure 117 montre le même exemple duquel on a retiré le CM de détection de contours. Il ne reste plus que le flou et le composant de réduction. Encore une fois, on peut observer le synchronisme des vidéos. Le composant de restitution a été implémenté afin de respecter pour chaque flux la synchronisation intra-flux. Pour ce faire, on utilise les étiquettes temporelles des unités temporelles de chacun des flux.

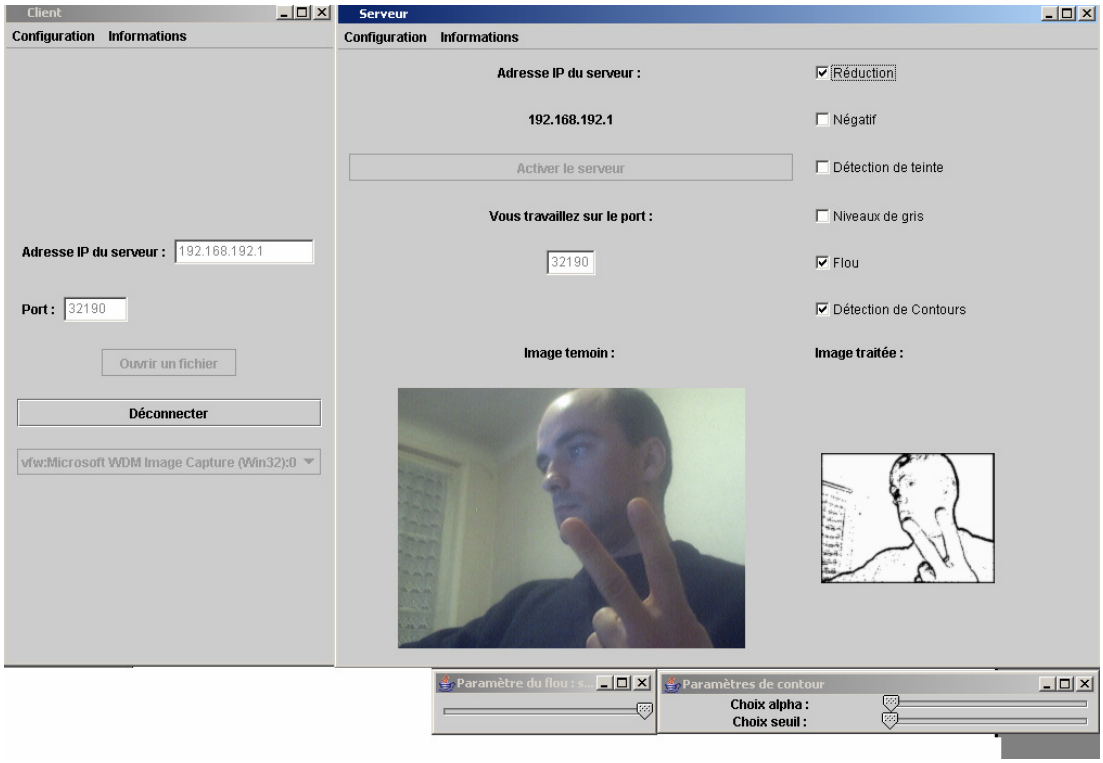


Figure 116 Ajout d'un CM de Réduction

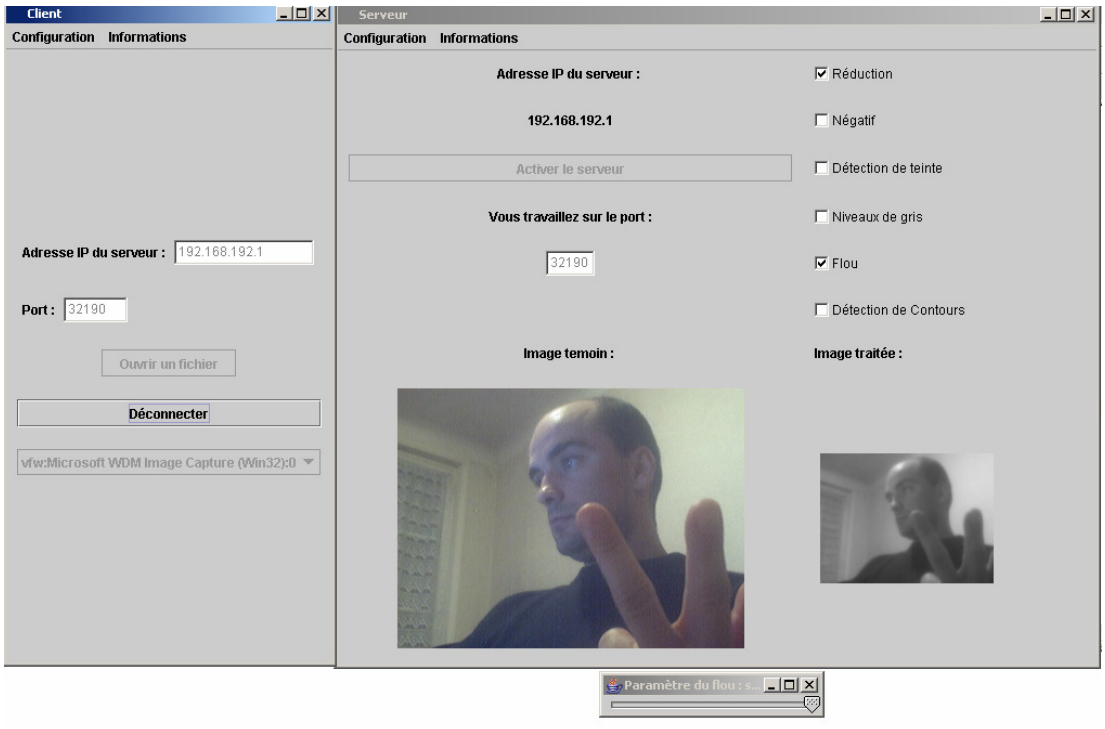


Figure 117 Retrait du CM de Détection de Contours

5 Synthèse

Nous avons présenté dans ce chapitre les principes utilisés dans l'implémentation du modèle Osagaia. L'implémentation que nous fournissons est réalisée à l'aide du langage de programmation orienté objet java. Nous avons implémenté le PE, le conduit et les opérateurs. Les fonctionnalités de l'AMD sont implémentées par des CM qui héritent de la classe ComposantMetier. Chaque CM est exécuté à l'intérieur d'un PE.

Grâce à ces entités, les propriétés non-fonctionnelles sont toutes implémentées et nous avons pu voir à l'aide du prototype que les résultats obtenus sont satisfaisants. Les flux sont gérés par le PE et la synchronisation est conservée entre les flux qui sont traités et ceux qui ne font que transiter. Le conduit permet de transporter les flux entre les PE de façon locale ou distribuée. Nous avons également testé à l'aide du prototype les connexions et les déconnexions dynamiques des PE et des conduits. Nous avons également pu tester les ajouts et suppressions de CM/PE sur le prototype. Nous avons constaté que ces modifications dynamiques de l'architecture fonctionnaient correctement.

Cela dit, le langage java et les API que nous utilisons sont assez lourds à mettre en œuvre. En effet, les initialisations de ces applications prennent un certain temps. La latence entre l'acquisition et la restitution des flux est de l'ordre de la seconde. Ceci est essentiellement dû à la machine virtuelle Java et au fait qu'il s'agit d'un langage interprété. Nous prévoyons de fournir d'autres implémentations du modèle Osagaia à l'aide d'autres langages de programmation mais aussi à l'aide de plates-formes de déploiement spécialement conçue pour la composition dynamique d'applications. L'objectif est de pouvoir comparer les résultats obtenus par ces différentes implémentations.

Conclusion et Perspectives

« This is the end

Beautiful friend

This is the end

My only friend, the end »

The End, Jim Morrison (The Doors), 1967

Le développement des applications multimédias distribuées à travers l'Internet nécessite de prendre en compte les points de vue de l'environnement d'exécution et ceux des utilisateurs si l'on veut pouvoir gérer la qualité de service de ces dernières. Dés lors que l'on veut considérer ces points de vue, on doit disposer d'une architecture logicielle capable d'agir à ces deux niveaux. Les médias constituent les données de ces applications et possèdent une importance toute particulière car les utilisateurs perçoivent directement leur contenu informationnel. Une dégradation de leur sémantique (désynchronisaion intra- et inter-flux) ou une incompréhension de ces données (par exemple langue du locuteur) sont des événements directement perceptibles par les utilisateurs. Ainsi, il est crucial de considérer leurs propriétés et de les respecter pour éviter ce genre de préjudice. De plus, elles doivent pouvoir être adaptées directement après la détection d'un problème d'accès par un utilisateur pour cause de handicaps par exemple. Des réactions à ces événements ne peuvent être mises en œuvre que si l'on possède une intégration forte des médias dans les applications. Le point de vue de l'environnement d'exécution peut être pris en compte dés lors que l'on est capable de mettre en œuvre des fonctionnalités permettant de se rapprocher des capacités fournies. Par exemple, avec des composants de traitement paramétrables, on peut s'adapter à la bande passante d'une liaison réseau en proposant différents taux de compression proposant des qualités de rendu différentes. Ces composants peuvent résoudre une grande partie de l'adaptation au contexte d'exécution. Il faut bien évidemment pour cela que l'application soit capable d'appréhender son environnement.

Notre objectif est de gérer la qualité de service des applications multimédias distribuées selon ces deux points de vue que nous avons définis à l'aide d'une méthode de conception et d'un modèle de qualité de service. Ces remarques nous ont amenés à proposer dans cette thèse une architecture logicielle qui répond à ces principes.

De nombreux travaux abordent l'un ou l'autre des aspects et choisissent la réservation de ressources plutôt que l'adaptation des applications pour la gestion de la qualité de service. L'absence de réservation de ressources sur l'Internet grand public nous impose de choisir l'adaptation des applications comme méthode de gestion. Les points clés qui nous ont intéressés dans cette thèse concernent la définition d'une architecture logicielle permettant ce type d'adaptation. L'intérêt que nous portons aux applications multimédias nous a amené à considérer les médias et plus largement les données de ces applications comme un concept central.

Nous avons débuté notre approche par la définition d'un modèle de flux de données permettant la prise en compte des propriétés importantes des médias. Ce modèle facilite la gestion des flux de données dans ces applications (tout est flux de données). Ainsi, on associe à chaque type de données des propriétés de séquence et des relations de synchronisation. Les données sont séparées en deux catégories qui permettent de les distinguer afin de permettre une gestion plus efficace. Cette classification s'opère en fonction des relations de synchronisation intra-flux (comportement temporel) des flux. Nous définissons donc les flux à contrainte temporelle forte et les flux à contrainte temporelle faible. La manipulation d'une structure unique sous la forme de flux facilite également la définition des relations de synchronisation inter-flux. Ce modèle introduit des politiques de synchronisation utilisées pour définir des relations de synchronisation entre flux provenant du même site d'une application. Ces politiques permettent de conserver le caractère important des propriétés des données afin d'assurer une diffusion correcte des données aux utilisateurs. De plus, il permet une intégration forte des médias, ce qui permet sans aucun doute de se conforter dynamiquement aux besoins émergeants des utilisateurs. L'intérêt de cette approche réside dans le fait que les propriétés des données sont conservées pendant tout leur transit dans une application alors que beaucoup de travaux définissent plutôt des modèles de présentation de données synchrones. Ce modèle permet de répondre en partie au point de vue des utilisateurs.

Enfin, nous avons également défini un modèle de composants basé sur l'utilisation du modèle de flux de données. Les entités qui le composent sont justifiées par l'intégration du modèle de flux mais aussi par les spécifications fonctionnelles amenées par la méthode de conception. Les applications multimédias distribuées sont définies comme un ensemble de composants logiciels interconnectés entre eux par des conduits. Nous faisons un pas vers la réutilisabilité et proposons de distinguer les aspects fonctionnels et non-fonctionnels. Nous avons donc défini les composants fonctionnels et les opérateurs. Ces derniers effectuent des opérations sur des flux synchrones permettant la synchronisation de plusieurs flux dans un flux composé et des

opérations permettant le traitement synchrone de plusieurs flux issus d'un même flux composé. Les composants fonctionnels sont destinés à l'implémentation des rôles atomiques logiciels. Ces composants sont exécutés au sein d'un processeur élémentaire dont l'avantage est de proposer un ensemble de services non-fonctionnels nécessaires à leur exécution. Cette structure permet de conserver la synchronisation entre les flux traités et ceux qui ne le sont pas. Chaque PE propose de reconstituer en sortie les flux composés à l'aide des politiques de synchronisation. Cette manière de faire peut paraître redondante mais elle est nécessaire pour éviter les pertes de synchronisation et aussi pour minimiser les retards de certains flux par rapport à d'autres. Le conduit est une autre entité fonctionnelle qui permet le transport synchrone des flux à travers l'application. Par la définition de ces deux entités, nous répondons aux deux sources de désynchronisation identifiées auparavant. Un autre avantage de cette structure est qu'elle permet dans certaines conditions de traitement de créer des relations de synchronisation inter-flux artificielles qui se justifient dés les phases de conception. Afin de permettre une adaptation des applications, ces entités sont entièrement supervisables par la plate-forme d'exécution. Elles constituent donc des solutions pour l'implémentation des applications mais aussi des solutions de gestion de la qualité de service par la définition de composants paramétrables ou par la définition de traitements spécialisés pour l'adaptation des flux synchrones au contexte d'exécution. Un autre point fort du modèle est que les entités fonctionnelles sont capables d'appréhender leur environnement afin d'avertir la plate-forme d'exécution de problèmes liés à des saturations/famines de buffers, à des sous- ou sur-emploi des composants métier.

La définition de tels modèles permet de définir des architectures logicielles flexibles et fiables par la définition d'entités entièrement supervisables par une plate-forme d'exécution. Une telle structure permet de mettre en œuvre des architectures reconfigurables dynamiquement.

Des perspectives de ces travaux vont consister à généraliser le modèle Osagaia afin de l'utiliser dans d'autres types de réseau comme les réseaux ad-hocs. On va de la sorte tenter de prendre en compte les périphériques de type CDC/CLDC/Smart Sensor contraints en énergie, en capacités de traitement, en capacités de transmission ou en capacités de taille d'affichage. L'objectif de telles recherches est d'adapter le modèle Osagaia afin qu'il soit utilisable dans ce type d'environnements et aussi avoir un modèle unique et ce quelle que soit l'implémentation. Ces adaptations peuvent consister par exemple à retirer ou délocaliser des services non-fonctionnels du modèle, à délocaliser le PE ou même, plus intéressant, à définir des délocalisations ou des compo-

sitions dynamiques des services en fonction des contraintes disponibles. Le modèle Korrontea fournit une approche intéressante pour ce type de réseau, il doit également être étendu. Les réseaux ad-hocs peuvent intégrer des capteurs sans-fil qui produisent des données en continu à contrainte temporelle faible. Ce modèle sait tenir compte de ce type de flux. Il peut également être intéressant de coupler sémantiquement les données issues de différents capteurs et ainsi de proposer des relations de synchronisation inter-flux dans ce type de réseaux. Dans ce cas là, le modèle temporel doit être remanié afin d'intégrer des relations de synchronisation entre flux provenant de sites différents. On peut cibler, par exemple, un domaine d'applications précis.

Actuellement, telles qu'elles sont définies les applications multimédias distribuées ne fournissent pas de rendus sur l'environnement physique de ces dernières (luminosité, bruit, etc.). Les capteurs sans fils sont des dispositifs matériels qui permettent d'obtenir ce type d'informations. On va donc s'intéresser à la définition des applications multimédias distribuées en environnements contraints avec prise en compte de l'environnement physique. Or, la conception à grande échelle d'applications multimédias intégrant des capteurs ne peut se faire qu'à l'aide d'un modèle conceptuel unique. En effet, les obligations de reconfiguration sont liées à la structure même de ces applications. Cette définition de modèle unifié va se dérouler par la généralisation des modèles Osagaia et Korrontea. Le modèle Osagaia doit s'ouvrir aux capteurs. Ainsi, ces applications sont décrites comme une interconnexion de rôles atomiques. Chaque rôle peut être réalisé soit de manière logicielle, soit de manière matérielle, soit de manière logicielle et matérielle. Le but est d'intégrer ces possibilités dans des processeurs élémentaires afin de leur proposer des services non-fonctionnels. Le modèle Korrontea doit également être généralisé afin de considérer les caractéristiques et les types de données produites par les capteurs.

Références Bibliographiques

« Il n'y a qu'une méthode pour inventer, qui est d'imiter. Il n'y a qu'une méthode pour bien penser, qui est de continuer quelque pensée ancienne et éprouvée. »

Propos sur l'éducation, Alain, 1932

[ABA95] B. Abali, C. B. Stunkel. « Time synchronization on SP1 and SP2 parallel systems ». In *Proceedings of 9th International Parallel Processing Symposium*, Santa Barbara, Etats-Unis, 25-28 avril 1995.

[ACC02] Accord. *Etat de l'art sur les Langages de Description d'Architectures (ADLs)*. Rapport Technique, Projet Accord – RNTL, juin 2002.

[ALD02] J. Aldrich, C. Chambers, D. Notkin. « ArchJava: Connecting Software Architecture to Implementation ». In *Proceedings of the International Conference on Software Engineering*, Orlando, Etats-Unis, 19-25 mai 2002.

[ALL83] J. F. Allen. « Maintaining Knowledge about Temporal Intervals ». *Communications of the ACM*, vol. 26, n°11, pp. 832-843, novembre 1983.

[AND93] D. P. Anderson. « Metascheduling for Continuous Media ». *ACM Transactions on Computer Systems*, vol. 11, n°3, pp. 226-252, août 1993.

[ANN93] M. Andreesen. *NCSA Mosaic Technical Summary*. Rapport Technique, National Center for Supercomputing Applications, Urbana-Champaign, Etats-Unis, 1993.

[ART06] ARTE. Arte VOD. http://www.artevod.com/home.do.

[BAI96] V. Baiceanu, C. Cowan, D. McNamee, C. Pu, J. Walpole. « Multimedia Applications Require Adaptive CPU Scheduling ». In *Proceedings of the 1996 Workshop on Resource Allocation Problems in Multimedia Systems*, Washington DC, Etats-Unis, 3 décembre 1996.

[BEU99] A. Beugnard, J-M. Jézéquel, N. Plouzeau, D. Watkins. « Making Components Contract Aware ». *IEEE Computer*, vol. 13, n°7, pp. 38-45, juillet 1999.

[BEU05] A. Beugnard. *Contributions à l'étude de l'assemblage de logiciels*. Habilitation à Diriger des Recherches, Université de Bretagne Sud, Brest, France, décembre 2005.

[BLA96] G. Blakowski, R. Steinmetz. « A Media Synchronization Survey: Reference Model, Specification, and Case Studies ». *IEEE Journal on Selected Areas in Communications*, vol. 14, n°1, pp. 5-35, janvier 1996.

[BLA02] A. P. Black, J. Huang, R. Koster, J. Walpole, C. Pu. « Infopipes: An abstraction for multimedia streaming ». *Multimedia Systems*, vol. 8, n°5, pp. 406-419, 2002.

[BOB01] J-F. Bobier, A. Fontaine, S. Vignes. *Microsoft .NET : Architecture et Services.* Rapport de fin d'études, Ecole Nationale Supérieure des Télécommunications, Paris, France, juillet 2001.

[BOO00] G. Booch, J. Rumbaugh, I. Jacobson. *Le guide de l'utilisateur UML*, Editions Eyrolles, 2000.

[BOU00] G. Bourguin. *Un support informatique à l'activité coopérative fondé sur la Théorie de l'Activité : le projet DARE*. Thèse de Doctorat, Université des Sciences et Technologies de Lille, Lille, France, juillet 2000.

[BOU01] N. M. N. Bouraquadi-Saâdani, T. Ledoux. « Le point sur la programmation par aspects ». *Technique et Science Informatiques*, vol. 20, n°4, pp. 505-528, 2001.

[BOU03] N. Bouillot. « Un algorithme d'auto synchronisation distribuée de flux audio dans le concept virtuel réparti ». In *Proceedings of 3ème Conférence Française sur les Systèmes d'Exploitation*, La Colle sur Loup, France, 14-17 octobre 2003.

[BOU05] E. Bouix, M. Dalmau, P. Roose, F. Luthon. « A Multimedia Oriented Component Model ». In *Proceedings of 19th IEEE International Conference on Advanced Information Networking and Applications*, Taipei, Taiwan, 28-30 mars 2005.

[BOX05] E. Bouix, M. Dalmau, P. Roose, F. Luthon. « A Component Model for transmission and processing of Synchronized Multimedia Data Flows ». In *Proceedings of the 1st International Conference on Distributed Frameworks for Multimedia Applications*, Besançon, France, 6-9 février 2005.

[BRU02] E. Bruneton, T. Coupaye, J.B. Stefani. « Recursive and Dynamic Software Composition with Sharing ». In *Proceedings of 7th International Workshop on Component-Oriented Programming*, Malaga, Espagne, 10-14 juin 2002.

[BRU03] E. Bruneton. *Developing with Fractal.* Tutoriel v.1.0, France Télécom R&D, septembre 2003.

[BUS45] V. Bush. « As We May Think ». *The Atlantic Monthly*, juillet 1945.

[CAM92] A. Campbell, G. Coulson, F. Garcia, D. Hutchison. « A Continuous Media Transport and Orchestration Service ». In *Proceedings of the Conference on Communications architectures and protocols*, Baltimore, Etats-Unis, 17-20 août 1992.

[CAR04] J. Carlström, T. Bodén. « Synchronous Dataflow Architecture for Network Processors ». *IEEE Micro*, vol. 24, n°5, pp. 10-18, septembre 2004.

[CCI89] CCITT. *General Aspects of Quality of Service and Network Performance in Digital Networks*. Recommandation I.350, International Telecommunication Union, Genève, Suisse, 1989.

[CEN97] S. Cen. *A Software Feedback Toolkit and its Application in Adaptive Multimedia Systems*. Thèse de Doctorat, Institut des Sciences et Technologies, Portland, Etats-Unis, octobre 1997.

[CES97] S. Cen, C. Pu, J. Walpole. « Flow and Congestion Control for Internet Media Streaming Applications ». In *Proceedings of Multimedia Computing and Networks*, janvier 1998.

[CHA00] M. Chaumont. *Rapport sur les techniques de transport de flux multimédia sur une architecture à différentiation de services*. Rapport de DEA, Université de Rennes 1, Rennes, France, juin 2000.

[CHE95] Z. Chen, S-M. Tan, R. H. Campbell, Y. Li. « Real Time Video and Audio in the World Wide Web ». In *Proceedings of 4th International World Wide Web Conference*, Boston, Etats-Unis, 11-14 décembre 1995.

[CHE96] H-Y. Chen, J-L. Wu. « MultiSync: A Synchronization Model for Multimedia Systems ». *IEEE Journal on Selected Areas in Communications*, vol. 14, n°1, pp. 238-248, janvier 1996.

[CHE03] S-C. Chen, M-L. Shyu, N. Zhao, C. Zhang. « Component-Based Design and Integration of a Distributed Multimedia Management System ». In *Proceedings of the 2003 IEEE International Conference on Information Reuse and Integration*, Las Vegas, Etats-Unis, 27-29 octobre 2003.

[COU96] J-P. Courtiat, L. F. R. Da Costa Carmo, R. C. De Oliveira. « A General-purpose Multimedia Synchronization Mechanism based on Causal Relations ». *IEEE Journal on Selected Areas in Communications*, vol. 14, n°1, pp. 185-195, janvier 1996.

[CPU01] C. Pu, K. Schwan, J. Walpole. « Infosphere Project: System Support for Information Flow Applications ». *ACM SIGMOD*, vol. 30, n°1, pp. 25-34, mars 2001.

[CRN02] I. Crnkovic, B. Hnich, T. Jonsson, Z. Kiziltan. « Specification, Implementation, and Deployment of Components ». *Communications of the ACM*, vol. 45, n°10, pp. 35-40, octobre 2002.

[DAL02] M. Dalmau, P. Roose, F. Luthon. « A Distributed Architecture for Cooperative and Adaptive Multimedia Applications ». In *Proceedings of 26th IEEE International Computer Software and Applications Conference*, Oxford, Angleterre, 26-29 août 2002.

[DIA93] M. Diaz, P. Sénac. « Time stream Petri nets, a model for multimedia streams synchronization ». In *Proceedings of 1st International Conference on Multimedia Modeling*, Singapour, République de Singapour, 9-12 novembre 1993.

[DIO95] C. Diot. « Adaptive Applications and QoS Guaranties ». In *Proceedings of the International Conference on Multimedia and Networking*, Aizu, Japon, 27-29 septembre 1995.

[DUC02] F. Duclos. *Environnement de Gestion de Services Non Fonctionnels dans les Applications à Composants*. Thèse de Doctorat, Université Joseph Fourier, Grenoble, France, octobre 2002.

[DUM99] C. Dumontier, F. Luthon, J-P. Charras. « Real-Time DSP Implementation for MRF-Based Video Motion Detection ». *IEEE Transactions on Image Processing*, vol. 8, n°10, pp. 1341-1347, octobre 1999.

[ECK00] B. Eckel. *Thinking in Java, 2nd edition*, 2000.

[EID02] V. S. W. Eide, F. Eliassen, O. Lysne, O-C. Granmo. « Real-time Processing of Media Streams: A Case for Event-Based Interaction ». In *Proceedings of 1st International Workshop on Distributed Event-Based Systems*, Vienne, Autriche, 2-3 juillet 2002.

[ELE00] P. Eles, A. Doboli, P. Pop, Z. Peng. « Scheduling with Bus Access Optimization for Distributed Embedded Systems ». *IEEE Transactions on VLSI Systems*, vol. 8, n°5, pp. 472-491, octobre 2000.

[ELS02] J. Elson, L. Girod, D. Estrin. « Fine-Grained Network Time Synchronization using Reference Broadcasts ». In *Proceedings of 5th Symposium on Operating Systems Design and Implementation*, Boston, Etats-Unis, 9-11 décembre 2002.

[FID88] C. Fidge. « Timestamps in message-passing systems that preserve the partial ordering ». *Australian Computer Science Communications*, vol. 10, n°1, pp. 56-66, février 1988.

[FIE99] R. Fielding *et al. Hypertext Transfer Protocol-HTTP/1.1*. RFC 2616, juin 1999.

[FIN01] L. Finch. « So Much OO, So Little Reuse ». *Dr Dobb's Portal*, http://www.ddj.com/184410808, 22 juillet 2001.

[GAA01] E. Garcia. *Une plate-forme de développement pour applications coopératives multimédia intégrant la gestion de la qualité de service*. Thèse de Doctorat, Université de Franche-Comté, Besançon, France, novembre 2001.

[GAN94] D. Garlan, R. Allen, J. Ockerbloom. « Exploiting Style in Architectural Design Environments ». In *Proceedings of 2nd ACM SIGSOFT Symposium on Foundations of Software Engineering*, Nouvelle-Orléans, Etats-Unis, 6-9 décembre 1994.

[GAR94] D. Garlan, M. Shaw. *An Introduction to Software Architecture*. Rapport Technique, School of Computer Science, Carnegie Mellon University, Pitsburgh, Etats-Unis, janvier 1994.

[GAR01] D. Garlan, B. Spitznagel. « A Compositional Approach for Constructing Connectors ». In *Proceedings of the 2001 Working IEEE/IFIP Conference on Software Architecture*, Amsterdam, Pays-Bas, 28-31 août 2001.

[GAU98] L. Gautier, C. Diot. « Design and Evaluation of MiMaze, a Multi-Player Game on the Internet ». In *Proceedings of the 1998 IEEE International Conference on Multimedia Computing and Systems*, Austin, Etats-Unis, 28 juin-1er juillet 1998.

[GHI98] G. Ghinea, J. Thomas. « QoS impact on user perception and understanding of multimedia video clips ». In *Proceedings of 6th ACM International Conference on Multimedia*, Bristol, Angleterre, 12-16 septembre 1998.

[GIB94] S. Gibbs, D. Tsichritzis. *Multimedia Programming: Objects, Environments and Frameworks*, Addison-Wesley/ACM Press, 1994.

[GIG00] E. Giguere. *Java 2 Micro Edition: Professional Developer's Guide*, John Wiley & Sons, 2000.

[GIS94] S. Gibbs, C. Breiteneder, D. Tsichritzis. « Data Modeling of Time-Based Media ». In *Proceedings of ACM SIGMOD International Conference on Management of Data*, Minneapolis, Etats-Unis, 24-27 mai 1994.

[GRI03] R. Grigoras. *Supervision de flux pour les contenus hypermédia : optimisation de politiques de préchargement et ordonnancement causal*. Thèse de Doctorat, Ecole Nationale Supérieure d'Electrotechnique, d'Electronique, d'Informatique, d'Hydraulique et des Télécommunications, Toulouse, France, décembre 2003.

[GUX02] X. Gu, K. Nahrstedt, W. Yuan, D. Wichadakul, D. Xu. « An XML-based Quality of Service Enabling Language for the Web ». *Journal of Visual Language and Computing*, vol. 13, n°1, pp. 61-95, février 2002.

[HAF98] A. Hafid, G. von Bochmann, R. Dssouli. « Distributed Multimedia Application and Quality of Service: A Review ». *Electronic Journal on Networks and Distributed Processing*, n°6, pp. 1-50, 1998.

[HAG02] D. Hagimont, N. Layaida. « Adaptation d'une application multimédia par un code mobile ». *Technique et Science Informatiques*, vol. 21, n°6, pp. 877-898, 2002.

[HAM72] C. Hamblin. « Instants and Intervals ». In *Proceedings of 1st Conference of the International Society for the Study of Time*, 1972.

[HAM04] R. Hammi, K. Chen. « Dynamic rate control in wireless video communications ». In *AS 150 – Systèmes Répartis et réseaux adaptatifs au contexte*, Paris, France, 1er avril 2004.

[HEI01] G. T. Heineman, W. T. Councill. *Component-Based Software Engineering: Putting the Pieces Together*, Addison-Wesley, 2001.

[HEM99] M. Hemy, U. Hengartner, P. Steenkiste, T. Gross. « MPEG System Streams in Best-Effort Networks ». In *Proceedings of the 1999 Packet Video Workshop*, New York City, Etats-Unis, 26-27 avril 1999.

[IEE04] IEEE 1588. *Precision clock synchronization protocol for networked measurement and control systems*. IEC 61588, janvier 2004.

[ILR68] D. M. Ilroy. « Mass Produced Software Components ». In *Proceedings of a conference sponsored by the NATO Science Committee*, Garmish, Allemagne, 7-11 octobre 1968.

[ISO95] ISO/IEC/JTC1/SC21/WG1. *QoS Framework*. International Standard Organization, 1995.

[JAM04] D. James. *Buying Information Systems: Selecting, Implementing and Assessing Off-The-Shelf Systems*, Gower Publishing Company, 2004.

[JIN03] C. Jin, D. X. Wei, S. H. Low. *FAST TCP for High-Speed Long-Distance Networks*. Internet Engineering Task Force, juin 2003.

[JOU05] Le Journal du Net. « La population internaute mondiale ». *Le Journal du Net*, http://www.journaldunet.com/diaporama/0509tourdumonde/01population-mondiale.shtml, juillet 2005.

[KAP03] U. J. Kapasi *et al.* « Programmable Stream Processors ». *IEEE Computer*, vol. 36, n°8, pp. 54-62, août 2003.

[KIC97] G. Kiczales *et al.* « Aspect-Oriented Programming ». In *Proceedings of 11th European Conference on Object-Oriented Programming*, Jyväskylä, Finlande, 9-13 juin 1997.

[KON00] F. Kon, R. H. Campbell, K. Nahrstedt. « Using Dynamic Configuration to Manage A Scalable Multimedia Distribution System ». *Computer Communications*, vol. 24, n°1, pp. 105-123, 2001.

[KRA01] C. Krasic, K. Li, J. Walpole. « The Case for Streaming Multimedia with TCP ». In *Proceedings of 8th International Workshop on Interactive Distributed Multimedia Systems*, Lancaster, Angleterre, 4-7 septembre 2001.

[KUR03] J. Kurose, K. Ross. *Analyse Structurée des réseaux*, Pearson Education France, Paris, France, 2003.

[LAI03] M. Lai. « Urbanismes, nouvelles architectures et e-SI ». In *Proceedings of 8ème colloque de l'Association Information et Management*, Grenoble, France, 21-23 mai 2003.

[LAM78] L. Lamport. « Time, Clocks, and the Ordering of Events in a Distributed System ». *Communications of the ACM*, vol. 21, n°7, pp. 558-565, juillet 1978.

[LAP03] S. Laplace, M. Dalmau, P. Roose. « New formal approach for QoS management in Distributed Multimedia Applications ». In *Proceedings of 6th IFIP/IEEE International Conference on Management of Multimedia Networks and Services*, Belfast, Irlande du Nord, 7-10 septembre 2003.

[LAP06] S. Laplace. *Conception d'Architectures Logicielles pour Intégrer la Qualité de Service dans les Applications Multimédias Réparties*. Thèse de Doctorat, Université de Pau et des Pays de l'Adour, IUT de Bayonne – Pays Basque, Bayonne, France, mai 2006.

[LAR04] Larousse. *Le Petit Larousse Grand Format*, Editions Larousse, 2004.

[LAR05] C. Larman. *UML 2 et les design patterns*, Pearson Education, 2005.

[LAY04] O. Layaida, S. B. Atallah, D. Hagimont. « Reconfiguration-based QoS Management in Multimedia Streaming Applications ». In *Proceedings of 30th EUROMICRO Conference*, Rennes, France, 1-3 septembre 2004.

[LEE94] B. Lee, A. R. Hurson. « Dataflow Architectures and Multithreading ». *IEEE Computer*, vol. 27, n°8, pp. 27-39, août 1994.

[LEE99] T. Lee *et al.* « Querying Multimedia Presentations Based on Content ». *IEEE Transactions on Knowledge and Data Engineering*, vol. 11, n°3, pp. 361-385, mai 1999.

[LIN03] A. Lins, E. F. Nakamura, A. A. F. Loureiro, C. J. N. Coelho. « Bean Watcher: A Tool to Generate Multimedia Monitoring Applications for Wireless Sensor Networks ». In *Proceedings of 6th IFIP/IEEE International Conference on Management of Multimedia Networks and Services*, Belfast, Irlande du Nord, 7-10 septembre 2003.

[LIT90] T. D. C. Little, A. Ghafoor. « Synchronization and Storage Models for Multimedia Objects ». *IEEE Journal on Selected Areas in Communications*, vol. 8, n°3, pp. 413-427, avril 1990.

[LIT93] T. D. C. Little. « A Framework for Synchronous Delivery of Time-Dependent Multimedia Data ». *Multimedia Systems*, vol. 1, n°2, pp.87-94, mars 1993.

[LIU03] A. Liu, I. Gorton. « Accelerating COTS Middleware Acquisition: The i-Mate Process ». *IEEE Software*, vol. 20, n°2, pp. 72-79, mars-avril 2003.

[LOP95] C. Lopes, W. Hursch. *Separation of Concerns*. Rapport technique, College of Computer Science, Northeastern University, Boston, Etats-Unis, février 1995.

[LOU07] C. Louberry, P. Roose, M. Dalmau. « Towards Sensor Integration into Multimedia Applications ». In *Proceedings of 4th European Conference on Universal Multiservice Networks*, Toulouse, France, 14-16 février 2007.

[MAG95] J. Magee, N. Dulay, S. Eisenbach, J. Kramer. « Specifying Distributed Software Architectures ». In *Proceedings of 5th European Software Engineering Conference*, Sitges, Espagne, 25-28 septembre 1995.

[MAL94] T. W. Malone, K. Crowston. « The Interdisciplinary Study of Coordination ». *ACM Computing Surveys*, vol. 26, n°1, pp. 87-119, mars 1994.

[MAO01] Z. M. Mao, H-S. W. So, B. Kang. « Network Support for Mobile Multimedia Using a Self-adaptive Distributed Proxy ». In *Proceedings of 11th International Workshop on Network and Operating Systems Support for Digital Audio and Video*, Port Jefferson, Etats-Unis, 25-26 juin 2001.

[MAR99] R. Marvie. *CORBA Components : la proposition unifiée, du modèle d'objets au modèle de composants*. Rapport Technique, Laboratoire d'Informatique Fondamentale de Lille, Lille, France, mai 1999.

[MAT88] F. Mattern. « Virtual Time and Global States of Distributed Systems ». In *Proceedings of International Workshop on Parallel And Distributed Algorithms*, Château de Bonas, France, 3-6 octobre 1988.

[MED00] N. Medvidovic, R. N. Taylor. « A Classification and Comparison Framework for Software Architecture Description Languages ». *IEEE Transactions on Software Engineering*, vol. 26, n°1, pp. 70-93, janvier 2000.

[MEH00] N. R. Mehta, N. Medvidovic, S. Phadke. « Towards a Taxonomy of Software Connectors ». In *Proceedings of 22nd International Conference on Software Engineering*, Limerick, Irlande, 4-11 juin 2000.

[MER00] B. Meyer. *Conception et Programmation orientées objet*, Eyrolles, 2000.

[MUN99] S. Mungee, N. Surendran, D. C. Schmidt. « The Design and Performance of CORBA Audio/Video Streaming Service ». In *Proceedings of 32nd Hawaii International Conference on System Sciences*, Hawaii, Etats-Unis, 5-8 janvier 1999.

[NIE95] O. Nierstrasz, D. Tsichritzis. *Object-Oriented Software Composition*, Prentice-Hall, 1995.

[OBS06] Observatoire Economique Aquitain des TIC. *Rapport 2006 sur le secteur des Technologies de l'Information et de la Communication en Aquitaine*. Rapport, Chambre de Commerce et d'Industrie de Bordeaux, Bordeaux, France, 2006.

[OCC03] A. Occello, A-M. D. Pinna, M. B. Fornarino, M. Riveill. « Contrôles des adaptations d'applications à base de composants ». In *Proceedings of Journée du Groupe Objets, Composants et Modèles*, Vannes, France, 5 février 2003.

[OMG03] OMG. *UML 2.0 Superstructure Specification*. 2003.

[OMG05] OMG. *UML Profile for Schedulability, Performance and Time Specification*. Version 1.1, janvier 2005.

[OWE96] P. Owezarski, M. Diaz. « Models for enforcing multimedia synchronization in visioconference applications ». In *Proceedings of 3rd International Conference on Multimedia Modeling*, Toulouse, France, 12-15 novembre 1996.

[OWE00] P. Owezarski, C. Chassot. « La couche Transport – services et protocoles – et ses évolutions ». In *Logiciel et Réseaux de Communication : compte rendu du groupe Logiciels et Réseaux de Communication de l'Observatoire Français des Techniques Avancées*, Editions Lavoisier, mai 2000.

[OWE03] P. Owezarski, M. Boyer. « Modélisation et implémentation d'architectures multimédias ; application au cas de la visioconférence à qualité de service garantie ». In *Les Réseaux de Pétri : vérification et mise en œuvre*, Editions Hermès, janvier 2003.

[OWI96] P. Owezarski. *Conception et formalisation d'une application de visioconférence coopérative – Application et extension pour la téléformation*. Thèse de Doctorat, Université Toulouse III, Toulouse, France, décembre 1996.

[PAD99] J. Padhye, J. Kurose, D. Towsley, R. Koodli. « A model based TCP-friendly rate control protocol ». In *Proceedings of International Workshop on Network and Operating System Support for Digital Audio and Video 1999*, Basking Ridge, Etats-Unis, 23-25 juin 1999.

[PIL06] D. Pilone, N. Pitman. *UML 2 en concentré*, O'Reilly, 2006.

[POS80] J. Postel. *User Datagram Protocol*. RFC 768, août 1980.

[POS81] J. Postel. *Transmission Control Protocol*. RFC 793, septembre 1981.

[RAN96] P. V. Rangan, S. S. Kumar, S. Rajan. « Continuity and Synchronization in MPEG ». *IEEE Journal on Selected Areas in Communications*, vol.14, n°1, pp.52-60, janvier 1996.

[RAY87] V. Rayward-Smith. « UET scheduling with unit interprocessor communication delays ». *Discrete Applied Mathematics*, vol. 18, pp. 55-71, 1987.

[RIV03] P. Rivière. « Qui profite des progrès de la communication ? ». *L'Atlas du Monde Diplomatique*, pp. 10-11, janvier 2003.

[ROM06] F. Roméo, C. Ballagny, F. Barbier. « PauWare : un modèle de composant basé état ». In *Proceedings of Journées Composants*, Perpignan, France, 4-6 octobre 2006.

[SAM97] J. Sametinger. *Software Engineering with Reusable Components*, Springer-Verlag, 1997.

[SCH96] H. Schulzrinne, S. Casner, R. Frederick, V. Jacobson. *RTP: A Transport Protocol for Real-Time Applications*. RFC 1889, janvier 1996.

[SEG02] M-T. Segarra, F. André. « Un modèle de composants pour l'adaptation dynamique à l'environnement ». *Revue des Sciences et Technologies de l'Information, série l'Objet*, vol. 8, n°1-2, pp. 231-245, 2002.

[SHA94] M. Shaw, D. Garlan. *Characteristics of Higher-level Languages for Software Architecture*. Rapport technique, Carnegie Mellon University, Pittsburgh, Etats-Unis, décembre 1994.

[SHA95] M. Shaw *et al.* « Abstractions for Software Architecture and Tools to Support Them ». *IEEE Transactions on Software Engineering*, vol. 21, n°4, pp. 314-335, avril 1995.

[SIN99] F. Singhoff. *Spécification temporelle modulaire et support pour les applications multimédias réparties*. Thèse de Doctorat, Ecole Nationale Supérieure des Télécommunications, Paris, France, décembre 1999.

[SMI94] B. C. Smith. *Implementation Techniques for Continuous Media Systems and Applications*. Thèse de Doctorat, University of California at Berkeley, Berkeley, Etats-Unis, 1994.

[STE93] J. B. Stefani. « Computational Aspects of QoS in an object-based, distributed systems architecture ». In *Proceedings of 3rd Workshop on Responsive Computer Systems*, Lincoln, Etats-Unis, septembre 1993.

[STE96] R. Steinmetz. « Human Perception of Jitter and Media Synchronization ». *IEEE Journal on Selected Areas in Communications*, vol.14, n°1, pp. 61-72, janvier 1996.

[STO93] D. L. Stone, K. Jeffay. « Queue Monitoring: A Delay Jitter Management Policy ». In *Proceedings of 4th International Workshop on Network and Operating System Support for Digital Audio and Video*, Lancaster, Angleterre, 3-5 novembre 1993.

[SUM99] Sun Microsystems. *Java Advanced Imaging*. novembre 1999.

[SUN99] Sun Microsystems. *Java Media Framework API Guide*. novembre 1999.

[SUN00] Sun Microsystems. *Java Sound Programmer Guide*. 2000.

[SUN03] Sun Microsystems. *Enterprise Java Beans Specification 2.1 Final Release*, 2003.

[SZY02] C. Szyperski, D. Gruntz, S. Murer. *Component Software – Beyond Object-Oriented Programming*, Addison-Wesley, 2002.

[TEN90] D. L. Tennenhouse. « Layered Multiplexing Considered Harmful ». In *Protocols for High-Speed Networks*, Elsevier Science Publishers B. V., Pays-Bas, 1990.

[TOK92] H. Tokuda, Y. Tobe, S. T-C. Chou, J. M. F. Moura. « Continuous Media Communication with Dynamic QOS Control Using ARTS with an FDDI Network ». In *Proceedings of the Conference on Communications Architecture and Protocols*, Baltimore, Etats-Unis, 17-20 août 1992.

[TOT05] T. Totozafiny, O. Patrouix, F. Luthon, J-M. Coutellier. « Motion Reference Image JPEG 2000: Road Surveillance Application with Wireless Device ». In *Proceedings of the Visual Communications and Image Processing*, Beijing, Chine, 12-15 juillet 2005.

[TOT07] T. Totozafiny. *Compression d'Images Couleurs pour Application à la Télésurveillance Routière par Transmission Vidéo à très bas débit*. Thèse de Doctorat, Université de Pau et des Pays de l'Adour, Bidart, France, juillet 2007.

[VIG96] M. R. Vigder, W. M. Gentleman, J. Dean. *COTS Software Integration: State of the art*. Rapport Technique, Conseil national de recherches du Canada, janvier 1996.

[VIL03] J. Villalobos. *Fédération de Composants : une Architecture Logicielle pour la Composition par Coordination*. Thèse de Doctorat, Université Joseph Fourrier, Grenoble, France, juillet 2003.

[VOG94] A. Vogel, B. Kerhervé, G. v. Bochmann, J. Gecsei. « Distributed Multimedia Applications and Quality of Service: A Survey ». In *Proceedings of the 1994 Con-*

ference of the Centre for Advanced Studies on Collaborative research, Toronto, Canada, 31 octobre-3 novembre 1994.

[WEI98] J. Weiss. *Télévision : Perception visuelle humaine v.1.00*. Rapport Technique, Supélec, Rennes, France, octobre 1998.

[ZHU01] W. Zhu, N. D. Georganas. « JQoS: Design and Implementation of a QoS-based Internet Videoconferencing System using the Java Media Framework (JMF) ». In *Proceedings of the IEEE Canadian Conference on Electrical and Computer Engineering*, Ontario, Canada, 13-16 mai 2001.